

DataInterchange Programmer's Reference Version 3 Release 1

DataInterchange Programmer's Reference Version 3 Release 1

SB34-2001-04

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xv.

Fifth Edition (December 1998)

This book is a major revision and replaces SB34-2001-03. New editions will reflect changes in procedure or technical details.

This edition applies to Version 3 Release 1, Modification Level 0, of the DataInterchange for MVS licensed program number 5655-B29 and DataInterchange for CICS licensed program number 5655-B30, and to all subsequent releases and modifications until otherwise indicated in new editions. Use this publication only for the purposes stated in “About This Book” on page xvii.

Books are not stocked at the address given below. Request for copies should be made to your marketing representative.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to: IBM, Department RD10, P.O. Box 20143, Tampa, FL, 33633-0872, U.S.A.

You may use this form to communicate your comments about this publication with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1993, 1998. All rights reserved.**

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	vii
Tables	ix
Notices	xv
Trademarks and Service Marks	xv
Softcopy Agreement	xvi
About This Book	xvii
Purpose of This Book	xvii
Changes after Publication	xvii
Audience for This Book	xvii
How This Book Is Organized	xvii
Terms Used in This Book	xviii
Syntax Conventions Used in This Book	xix
How to Read a Syntax Diagram	xix
Related Books	xx
Chapter 1. Using the DataInterchange Utility	1-1
Command Language Syntax	1-1
Error Filtering	1-4
Overriding the Utility Condition Codes	1-6
Outbound Processing	1-6
Fixed-to-Fixed (Runtime)	1-16
SAP Status Tracking in DataInterchange	1-18
Inbound Processing	1-20
Managing Data	1-27
Reporting and Data Extraction	1-39
Reporting Using Management Reporting and Transaction Store	1-48
Customizing	1-61
Managing Mailboxes	1-63
Profile Maintenance	1-64
Continuous Receive Functions	1-65
Persistent Environment Debugging	1-67
Keyword, Option, Criteria, and Range Criteria Descriptions	1-69
Running the DataInterchange Utility in MVS	1-116
Chapter 2. File Formats and DataInterchange Utility Records	2-1
Transaction Store Input and Output Files	2-1
Export/Import Utility Function	2-12
Export/Import Control File (CTLFILE)	2-12
The Export/Import Files	2-15
Export/Import of Trading Partner Profile	2-18
Export/Import of Application Data Formats	2-37
Export/Import of Trading Partner Transactions	2-39
Export/Import of Table Definitions	2-45
Additional Profile Layouts	2-48
Importing the New Definitions to DataInterchange	2-59
Mapping Migration Input Control File (INCNTL)	2-59
Mapping Migration Output Control File (OUTCNTL)	2-60

DataInterchange Utility Records Format	2-61
Management Reporting	2-76
Transaction Store Data Extract Information Categories	2-80
Chapter 3. Using the DataInterchange Application Program Interface	3-1
API Languages	3-1
API Business Tasks	3-12
Translation Services	3-20
Enveloping Services	3-69
Data Extraction Services	3-101
Communication Services	3-105
Status Update Services	3-126
SYNCPPOINT Services	3-132
Chapter 4. Exit Routines	4-1
Return Codes	4-1
DataInterchange User Extensions	4-1
Independent Programs	4-32
Chapter 5. Using DataInterchange in the CICS Environment	5-1
Running the DataInterchange Utility in CICS	5-1
CICS Startup Considerations	5-12
Running DataInterchange for CICS in a HOT-DI Environment	5-12
Format of DataInterchange Utility Control Information	5-18
Continuous Receive Considerations	5-23
Response Applications	5-27
Persistent Environment	5-31
Reserved Temporary Storage and Transient Data Queues	5-33
Interface Between DataInterchange for CICS, Expedite/CICS and Information Exchange	5-35
DataInterchange Supplied Transactions	5-43
Performance Monitor User Exit	5-44
Using EDIW to Invoke the DataInterchange Utility	5-45
Chapter 6. Interfacing to Other Networks and Applications	6-1
Generalized Network	6-2
Point-to-Point Network	6-4
Application-to-Network Flow	6-6
Parameters Passed to the Communication Routine	6-9
Building Network Commands	6-9
Message Handler	6-13
Special Communication Routine for CICS	6-14
Continuous Receive Interface (CICS only)	6-19
Interfacing with the In*Touch Gateway Communications Package	6-21
Interfacing DataInterchange for CICS with SDM LinkPlus Interactive	6-34
SAP Status Tracking in DataInterchange	6-34
Interfacing DataInterchange with MQSeries	6-37
Appendix A. DataInterchange Control Blocks	A-1
Service Name Block (SNB)	A-1
Common Control Block (CCB)	A-2
Function Control Block (FCB)	A-4
Translator Control Block (TRCB)	A-6

Translator Input Data Block (TRIDB)	A-29
Translator Output Data Block (TRODB)	A-31
Communication Interface Control Block (CMCB)	A-34
Trading Partner Profile Block (TPPDB)	A-42
Communication Data Block (DATABLK)	A-52
Network Profile Block (NPDB)	A-53
Requestor Profile Block (REQDB)	A-56

Appendix B. DataInterchange Utility Condition Codes and API Return

Codes	B-1
Return Codes	B-1
DataInterchange Utility Condition Codes	B-2
Translation Condition Codes	B-4
Communications Condition Codes	B-14
Combination Command Condition Codes	B-16
Application Programming Interface Return Codes	B-17
Communication Return Codes	B-27

Appendix C. Copy Books and Include Files

Copy Books for COBOL	C-2
Include Files for PL/I	C-13
Include Files for C	C-25
Copy Books for ASSEMBLER	C-36

Appendix D. Sample Programs

Creating Tagged Import Files from Fixed Flat Files	D-1
Initializing and Terminating DataInterchange	D-1
Query the Transaction Store using COBOL	D-9
Query the Transaction Store in PL/I	D-11
Translate and Queue for Send	D-12
Send Queued Data	D-15
End Translation	D-18
Receive From the Network	D-19
Translate Received Data	D-22
End DataInterchange	D-26
Data Extract Report Generator (EDISAMR1)	D-27
Data Extract Report Generator (EDISAMS1)	D-35
Network Activity Report Generator (EDISAMT1)	D-44
Initializing, Invoking, and Terminating HOT-DI	D-49
Invoking Response Programs	D-56
Field Exit Program	D-57
Test for Filter Type	D-65
Filtration Exit Examples	D-68
Authentication Examples	D-85
Encryption Examples	D-95
Get Envelope Service Example	D-117
Put Envelope Service Example	D-118
Inbound Envelope Program Example	D-120
Outbound Envelope Program Example	D-121
VANICICS Network Program Example	D-122

Appendix E. Using Sample JCL

DataInterchange Utility (EDIUTILV) JCL	E-1
--	-----

	Utility Data Sets Required	E-17
	Archive VSAM event log entries (EDIELARV)	E-18
	Archive DB2 event log entries (EDIELARD)	E-20
	 Appendix F. Space Calculation Examples	F-1
	Space Requirements for DataInterchange Tables and Files	F-1
	DataInterchange Allocation Tables	F-24
	Space Calculation Scenario	F-29
	Worksheets	F-37
	 Appendix G. Performance Considerations	G-1
	DataInterchange Optimization	G-1
	 Glossary	X-1
	 Index	X-7

Figures

3-1.	Load Module Relationships	3-4
3-2.	Translate, Envelope, and Send Process	3-24
3-3.	Receive, Deenvelope, and Translate	3-48
6-1.	Application Interface with the Network	6-6
6-2.	Sample DataInterchange/Gateway Configuration	6-21
6-3.	TIGD31 Network Profile Sample	6-22
6-4.	TIGV31 Network Profile Sample	6-24
6-5.	Building SYSIN	6-26

Tables

1-1.	SAP Extract Keywords	1-20
1-2.	Keywords For Removing SAP Status	1-20
1-3.	Various Ways to Remove and Archive Event Log Entries	1-34
1-4.	Continuous Receive Status Report Statuses	1-66
1-5.	Continuous Receive Status Report Record Format	1-66
2-1.	Fields in the Enveloping Options File for Functional Acknowledgments	2-10
2-2.	Export/Import Control File	2-13
2-3.	Export/Import Common Control Record (OC1/OC2) Fields	2-16
2-4.	Export/Import Common End of Group Record (000) Fields	2-17
2-5.	Trading Partner Profile Header Record	2-19
2-6.	Trading Partner Profile Members	2-19
2-7.	TP Contacts Definition	2-33
2-8.	TP Control Numbers (7P4)	2-33
2-9.	Comments Definition	2-33
2-10.	Contacts Definition	2-33
2-11.	Standard/Envelope Definition	2-35
2-12.	Standard Transaction Definition	2-35
2-13.	Standard Segment Usage	2-36
2-14.	Standard/Envelope Segment Definition	2-36
2-15.	Standard/Envelope Data Element Usages	2-36
2-16.	Standard/Envelope Data Element Definition	2-37
2-17.	Application Data Format Definition	2-38
2-18.	Application Data Format Structure Header	2-38
2-19.	Application Data Format Structure Entry	2-39
2-20.	Application Data Format Field Entry	2-39
2-21.	Trading Partner Transaction	2-40
2-22.	Trading Partner Mapping Segment	2-41
2-23.	Trading Partner Mapping Data Element	2-42
2-24.	Trading Partner Mapping Rules	2-42
2-25.	Trading Partner Send Usage	2-43
2-26.	Trading Partner Receive Usage	2-44
2-27.	Table Definition	2-46
2-28.	Table Entry	2-48
2-29.	Requestor Profile	2-49
2-30.	Security Profile	2-49
2-31.	Network Profile	2-50
2-32.	Network Operations Profile	2-50
2-33.	Activity Log Profile	2-51
2-34.	Application Definition Profile	2-51
2-35.	User Program Information Profile	2-52
2-36.	Language Profile	2-52
2-37.	EDIFACT Profile	2-53
2-38.	ICS Profile	2-54
2-39.	UNTDI Profile	2-55
2-40.	UCS Profile	2-56
2-41.	X12 Profile	2-56
2-42.	CONTRECV Member	2-57
2-43.	System Profile	2-58
2-44.	MQSeries Queue	2-59

2-45.	C Record for Translating to Standard Format	2-61
2-46.	D Record — All Structures Passed Together	2-70
2-47.	D Record — Structures Passed Separately	2-71
2-48.	Z Record	2-72
2-49.	Raw Data Record	2-72
2-50.	Information Record	2-74
2-51.	Interchange Header	2-75
2-52.	Group Header	2-75
2-53.	Transaction Set Header	2-76
2-54.	Queuing Totals	2-76
2-55.	Trading Partner Information Data Extract	2-77
2-56.	Trading Partner Capability Information Data Extract	2-77
2-57.	Network Traffic Data Extract	2-78
2-58.	Transaction Activity Report Data	2-79
2-59.	Format of the Common Key	2-81
2-60.	Interchange Data Extract Record Layout	2-81
2-61.	Group Data Extract Record Layout	2-83
2-62.	Transaction Data Extract Record Layout	2-84
2-63.	Application Data Extract Record Layout	2-86
2-64.	Image Record Layout	2-87
3-1.	EDITSIN Keywords	3-2
3-2.	Control Block Abbreviations	3-13
3-3.	Overview of Environmental Services	3-14
3-4.	Overview of Translation Services	3-14
3-5.	Overview of Enveloping Services	3-15
3-6.	Overview of Data Extraction Services	3-15
3-7.	Overview of Communications Services	3-15
3-8.	Overview of Status Update Services	3-16
3-9.	Overview of SYNCPOINT Services	3-17
3-10.	Service Name Block (SNB) Initialization for Translate to Standard	3-28
3-11.	Function Code Block (FCB) Initialization for Translate to Standard	3-28
3-12.	Translator Control Block (TRCB) Initialization for Translate to Standard	3-28
3-13.	Translator Output Data Block (TRODB) Initialization for Translate to Standard	3-30
3-14.	Initialization - First Call for Transaction - TRCB	3-31
3-15.	Initialization - First Call for Transaction - TRODB	3-34
3-16.	Fields in TRCB Returned on First Call for a Transaction	3-35
3-17.	Fields in TRCB Returned When an Interchange is Written	3-35
3-18.	Fields in TRCB Returned on Translation	3-37
3-19.	Fields in TRCB Related to Interchange Header and Trailer	3-38
3-20.	Fields in Translator Control Block (TRCB) Related to Group Header and Trailer	3-38
3-21.	Fields in the Translator Control Block (TRCB) Related to Transaction Header and Trailer	3-39
3-22.	Service Name Block (SNB) Initialization for Translate-to-Application	3-50
3-23.	Function Code Block (FCB) Initialization for Translate to Application	3-50
3-24.	Translator Control Block (TRCB) Initialization for Translate-to-Application	3-50
3-25.	Translator Input Data Block (TRIDB) Initialization for Translate-to-Application	3-51
3-26.	Translator Output Data Block (TRODB) Initialization for Translate-to-Application	3-51

3-27.	Initialization - First Call for Transaction	3-52
3-28.	Initialization - First Call for Transaction	3-52
3-29.	Transaction Attribute Fields in the Translator Control Block (TRCB)	3-53
3-30.	Application Related Fields in Translator Control Block (TRCB)	3-54
3-31.	Application Data	3-54
3-32.	Interchange Header/Trailer Fields in the Translator Control Block (TRCB)	3-55
3-33.	Group Header/Trailer Fields in the Translator Control Block (TRCB)	3-55
3-34.	Transaction Header/Trailer Fields in the Translator Control Block (TRCB)	3-56
3-35.	Service Name Block (SNB) Initialization for Translate-to-Application	3-58
3-36.	Function Code Block (FCB) Initialization for Translate-to-Application	3-58
3-37.	Translator Control Block (TRCB) Initialization for Translate-to-Application	3-58
3-38.	Transaction Input Data Block (TRIDB) Initialization for Translate-to-Application	3-60
3-39.	Translator Output Data Block (TRODB) Initialization for Translate-to-Application	3-60
3-40.	Initialization - First Call for Transaction - TRCB	3-61
3-41.	Initialization - First Call for Transaction - TRODB	3-61
3-42.	Transaction Attribute Fields in TRCB	3-62
3-43.	Application Related Fields in TRCB	3-63
3-44.	Application Data	3-64
3-45.	Fields Related to Functional Acknowledgments	3-64
3-46.	Interchange Header/Trailer Fields in TRCB	3-64
3-47.	Group Header/Trailer Fields in the TRCB	3-66
3-48.	Transaction Header/Trailer Fields in the TRCB	3-67
3-49.	SNB Initialization for Enveloping Function	3-74
3-50.	FCB Initialization for Enveloping Function	3-74
3-51.	TRCB Initialization for Enveloping Function	3-74
3-52.	TRIDB Initialization for Enveloping Function	3-76
3-53.	TRODB Initialization for Enveloping Function	3-76
3-54.	Initialization - For Each Transaction	3-76
3-55.	Transaction Attribute Fields in TRCB	3-77
3-56.	Application Related Fields in TRCB	3-77
3-57.	Fields in TRCB Related to Interchange Header or Trailer	3-78
3-58.	Fields in TRCB Related to Group Header or Trailer	3-79
3-59.	Fields in TRCB Related to Transaction Header or Trailer	3-80
3-60.	Fields in TRCB Returned When an Interchange is Written	3-80
3-61.	Fields in TRCB Returned When an Interchange is Written	3-85
3-62.	SNB Initialization for Deenvelope	3-87
3-63.	FCB Initialization for Deenvelope	3-87
3-64.	TRCB Initialization for Deenvelope	3-87
3-65.	TRIDB Initialization for Deenvelope	3-90
3-66.	TRODB Initialization for Deenvelope	3-90
3-67.	Transaction Attribute Fields in TRCB	3-91
3-68.	Application-Related Fields in TRCB	3-92
3-69.	Fields Related to Functional Acknowledgments	3-92
3-70.	Interchange Header/Trailer Fields in TRCB	3-92
3-71.	Group Header/Trailer Fields in the TRCB	3-95
3-72.	Transaction Header/Trailer Fields in the TRCB	3-96
3-73.	SNB Initialization for Data Extraction	3-101
3-74.	FCB Initialization for Data Extraction	3-102

3-75.	TRCB Initialization for Data Extraction	3-102
3-76.	TRIDB Initialization for Data Extraction	3-102
3-77.	TRODB Initialization for Data Extraction	3-102
3-78.	Initialization for Each Transaction	3-102
3-79.	Returned Information on Extract Request	3-104
3-80.	Returned Information in Output Data Block	3-104
3-81.	Trading Partner Fields Used by Communications	3-107
3-82.	Network Capability Fields in CMCB	3-108
3-83.	Initialization for Send Transaction Data	3-110
3-84.	Fields Returned from 211 Request	3-113
3-85.	Initialization for Send Transaction Data	3-115
3-86.	Fields Returned from 221 Request	3-117
3-87.	Initialization for Receive	3-118
3-88.	Fields Returned from 232 Request	3-120
3-89.	Initialization for Cancel	3-122
3-90.	Fields Returned from 233 Request	3-122
3-91.	Initialization for Query Filename	3-124
3-92.	Fields Returned from 300 Request	3-124
3-93.	Initialization for Process Network Acknowledgments	3-125
3-94.	Full Key Using Trading Partner Nickname	3-128
3-95.	Full Key Using Interchange ID and Qualifier	3-129
3-96.	Full Key Using Account Number and User ID	3-129
3-97.	Transaction Handle	3-129
3-98.	Alternate Key Using Account Number and User ID	3-130
3-99.	Alternate Key Using Interchange Qualifier and ID	3-130
3-100.	Interchange Fields Established During Status Update	3-131
3-101.	DataInterchange DB/2 Recovery Decision Table	3-133
4-1.	Data Extract Exit Control Block	4-33
5-1.	Multiple TSQ Control Block	5-5
5-2.	Unit of Work for Each DataInterchange Utility Command	5-7
5-3.	Format of DataInterchange Utility Control Information	5-19
5-4.	DataInterchange Utility Control Information Field Descriptions	5-20
5-5.	Format of Performance Monitor Commarea	5-44
5-6.	Active Function Keys for DataInterchange for CICS Utility Invocation	5-46
5-7.	Command Fields for DataInterchange for CICS Utility Invocation	5-47
6-1.	Definition of the Network Operation Profile	6-10
6-2.	Network Operation Example	6-11
6-3.	Description of FSUPPORT Bytes	6-12
6-4.	Format of Network Program Control Information	6-15
6-5.	Format of Returned Network Program Control Information	6-16
6-6.	Format of Message Handler Program Control Information	6-17
6-7.	Format of Returned Message Handler Program Control Information	6-18
6-8.	Format of Control Information Given to EDICRIN	6-19
6-9.	Network Command Tags for Fields of the Network profile	6-30
6-10.	Network Command Tags for Fields of the Requestor profile	6-31
6-11.	Network Command Tags for Fields of the Trading Partner profile	6-31
6-12.	Network Command Tags for Fields of the Common control block	6-32
6-13.	SAP Status Codes Supported by DataInterchange	6-37
A-1.	SNB Definition	A-1
A-2.	CCB Definition	A-3
A-3.	FCB Definition	A-4
A-4.	TRCB Definition	A-6
A-5.	Unique Error Codes from Translator	A-28

A-6.	TRIDB Functions and Initialization Requirements	A-30
A-7.	TRIDB Definition with 2-Byte Lengths	A-31
A-8.	TRIDB Definition with 4-Byte Lengths	A-31
A-9.	TRODB Functions and Initialization Requirements	A-32
A-10.	TRODB Definition with 2-Byte Lengths	A-33
A-11.	TRODB Definition with 4-Byte Lengths	A-33
A-12.	Definition of Communication Interface Control Block	A-34
A-13.	Definition of the Trading Partner Profile Block	A-42
A-14.	DATABLK Up to 32 K	A-52
A-15.	DATABLK Over 32 K	A-52
A-16.	Definition of the Network Profile Block (NPDB)	A-53
A-17.	Definition of the Requestor Profile Block (REQDB)	A-56
B-1.	DataInterchange Utility Condition Codes for Noncommunication and Nontranslation Functions	B-2
B-2.	DataInterchange Utility Condition Codes from Invalid Application Input	B-6
B-3.	DataInterchange Utility Condition Codes for Translation with Normal or Warning Conditions	B-7
B-4.	DataInterchange Utility Condition Codes for Data Element Translation Errors	B-7
B-5.	DataInterchange Utility Condition Codes for Segment Translation Errors	B-8
B-6.	DataInterchange Utility Condition Codes for Transaction Translation - EDI Syntax Errors	B-9
B-7.	DataInterchange Utility Condition Codes for Transaction Translation - Environmental Errors	B-10
B-8.	DataInterchange Utility Condition Codes for Group Translation Errors	B-11
B-9.	DataInterchange Utility Condition Codes for Interchange Translation Errors	B-12
B-10.	DataInterchange Utility Condition Codes Set by Invalid Data in the Input File	B-13
B-11.	DataInterchange Utility Condition Codes for Environmental and Program Errors	B-13
B-12.	DataInterchange Utility Condition Codes for Communications Services in MVS	B-15
B-13.	DataInterchange Utility Condition Codes for Communications Services in CICS	B-15
B-14.	Initialization Return Codes	B-17
B-15.	Negative Return Codes and Messages	B-18
B-16.	Termination Return Codes	B-18
B-17.	Translation with Normal or Warning Return Codes	B-19
B-18.	Data Element Level Translation Errors	B-20
B-19.	Segment Level Translation Errors	B-21
B-20.	Transaction Level Syntax Errors	B-22
B-21.	Transaction Level Environmental Errors	B-22
B-22.	Group Level Translation Errors	B-24
B-23.	Interchange Level Translation Errors	B-24
B-24.	Invalid Data in the Input File	B-25
B-25.	Environmental and Program Errors	B-26
B-26.	Communications—Warnings	B-28
B-27.	Communications—Nonsevere Errors	B-28
B-28.	Communications—Severe Errors	B-28
B-29.	Status Update Return Codes	B-29
B-30.	Initialize SYNC Return Codes	B-29

B-31.	COMMIT Work Return Codes	B-30
B-32.	ROLLBACK Work Return Codes	B-30
C-1.	Library Members	C-1
C-2.	Library Members	C-2
E-1.	JCL Sections Required by the DataInterchange Utility	E-17
F-1.	DataInterchange Database Records	F-1
F-2.	DataInterchange Supplied Database Allocation for DB2	F-24
F-3.	DataInterchange Supplied Database Allocation for VSAM	F-27
F-4.	DataInterchange/DB2 Database Allocation Required for Space Calculation Scenario	F-30
F-5.	DataInterchange/VSAM Database Allocation Required for Space Calculation Scenario	F-35
F-6.	DataInterchange/DB2 Database Allocation Worksheet	F-37
F-7.	DataInterchange/VSAM Database Allocation Worksheet	F-40

Notices

This publication was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only an IBM product, program or service may be used. Any functionally equivalent product, program, or service that does not infringe on any of IBM Global Services' intellectual property rights may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Trademarks and Service Marks

The following terms are trademarks of the International Business Machines Corporation in the United States and other countries:

Expedite
IBM
IBM Global Network
IBM Global Services

All other trademarks are the property of their respective owners.

Softcopy Agreement

For softcopy versions of this book, we authorize you to:

- Copy, modify, and print the documentation contained on the media, for use within your enterprise, provided you reproduce the copyright notice, all warning statements, and other required statements on each copy or partial copy.
- Transfer the original unaltered copy of the documentation when you transfer the related IBM product (which may be either machines you own, or programs, if the program's license terms permit a transfer). You must, at the same time, destroy all other copies of the documentation.

You are responsible for payment of any taxes, including personal property taxes, resulting from this authorization.

THERE ARE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT WILL IBM BE RESPONSIBLE FOR ANY CONSEQUENTIAL OR INCIDENTAL DAMAGES ASSOCIATED WITH USE OF THE PROGRAM OR DOCUMENTATION, EVEN IF IBM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Your failure to comply with the terms above terminates this authorization. Upon termination, you must destroy your machine readable documentation.

About This Book

This book provides guidance in DataInterchange for MVS and DataInterchange for CICS Version 3 Release 1.

Purpose of This Book

This book provides information for the DataInterchange application programmer. It describes general-use programming and provides reference information for developing application programs that use DataInterchange.

Changes after Publication

The README file on the product tape may contain additional information or changes made after this book was published.

Audience for This Book

This book is intended for the electronic data interchange (EDI) programmer who implements a computer system for electronic exchange of business information. The programmer should be familiar with or have a working knowledge of:

- Multiple Virtual Storage (MVS)
- Customer Information Control System (CICS)
- DataInterchange
- A programming language such as COBOL, C, or Assembler
- An IBM relational database management system such as Database 2 (DB2)
- Virtual Storage Access Method (VSAM)

How This Book Is Organized

This book contains the following information:

Chapter 1, "Using the DataInterchange Utility," provides information on using PERFORM commands.

Chapter 2, "File Formats and DataInterchange Utility Records," describes the input and output record formats used by the Transaction Store, Mapping Migration, DataInterchange Utility, Management Reporting, Transaction Store Data Extract, Close Mailbox, Export/Import, and Custom Layout.

Chapter 3, "Using the DataInterchange Application Program Interface," describes how to write application programs that directly call DataInterchange services.

Chapter 4, "Exit Routines," describes how to write routines for DataInterchange to call. This chapter includes field, security, pre-translation, and post-translation exit routines.

Chapter 5, "Using DataInterchange in the CICS Environment," describes how to write response programs for a CICS environment.

Chapter 6, "Interfacing to Other Networks and Applications," describes how to write programs that communicate with applications and networks.

Appendix A, "DataInterchange Control Blocks," contains the definition and explanation of DataInterchange control blocks needed to use the DataInterchange application programming interface.

Appendix B, "DataInterchange Utility Condition Codes and API Return Codes," contains condition code and return code information.

Appendix C, "Copy Books and Include Files," contains language specific include files and copy book definitions of the control blocks.

Appendix D, "Sample Programs," contains sample programs that illustrate how to define DataInterchange control blocks, how to initialize and terminate DataInterchange, and so forth. This Appendix also contains sample exit routines that illustrate how to use field exit routines, how to test for filter types, and so forth.

Appendix E, "Using Sample JCL," contains sample DataInterchange and archival JCL.

Appendix F, "Space Calculation Examples," contains an example of space calculations.

Appendix G, "Performance Considerations," contains information on tuning and general optimization techniques for DataInterchange for MVS and DataInterchange for CICS.

This book also includes a glossary and an index.

Terms Used in This Book

The following acronyms and terms are used in this book:

Abbreviation/Term	Definition
Applications	Programs that process information
CICS	Customer Information Control System
Data set	Basic unit of data storage for MVS
DataInterchange	DataInterchange for MVS and DataInterchange for CICS
EDI	Electronic data interchange
Electronic transmission	Media by which information is transmitted, such as a public network
MVS	Multiple Virtual Storage
Profile	Collection of descriptive or key information stored in the profiles part of the DataInterchange product
TD	Transient data queue
Trading Partners	Parties who agree to exchange information

Translation	Process of converting business information from an application-specific format to a standard format, or from a standard format to an application-specific format
TS	Temporary storage queue
VS	VSAM entry sequences data set

Syntax Conventions Used in This Book

The following syntax conventions are used throughout this book.

- Uppercase letters represent values that you must type without change (these values are not case-sensitive). For example:

EDI PRINT(FILE)

- Lowercase italicized letters represent variable parameters for which you supply the values. For example:

SYSID(*system-name*)

- Italicized letters are also used for emphasis within textual descriptions. For example:

In the example *system-name* is the resource name.

- Italicized mixed letters represent field names from panels. For example:

Trans data queue

- Bold letters represent control block field names. For example:

FILEID

- Bold letters are also used to provide additional emphasis to certain words. For example:

Do this **or** do that...

How to Read a Syntax Diagram

The syntax diagram shows you how to specify a command so that the operating system can correctly interpret your entry. Read the syntax diagram from left to right and from top to bottom, following the horizontal line (the main path).

Symbols and Punctuation

The following symbols are used in syntax diagrams.

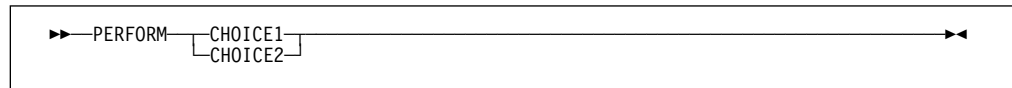
Symbol	Description
▶▶	Marks the beginning of the command syntax
▶	The command syntax is continued
	Marks the beginning and end of a fragment or part of the command syntax
◀◀	Marks the end of the command syntax

You must include all punctuation, such as colons, semicolons, commas, quotation marks, and minus signs that are shown in the syntax diagram.

Syntax that is displayed in all capital letters must be entered as shown.

Variables are italicized, appear in lowercase letters, and represent names or values you supply.

Stacked items represent choices. For example, the following shows that you can select the first or second choice:



An arrow above the main path that returns to a previous point means the sequence of items included by the arrow can be specified more than once.

The following example indicates that you can enter multiple file names.



Related Books

The following list contains the publications referred to in this book, as well as other available publications that contain related information.

- *Customizing and Developing Applications with Expedite/CICS*, GC34-3304

This book provides a description of the CICS (interactive) interface for the IBM Global Network.

- *DataInterchange Administrator's Guide*, SB34-2002

This book provides information for the electronic data interchange (EDI) administrator. This book provides information about entering, sending, and receiving EDI transactions and other documents interactively.

- *DataInterchange Client User's Guide*, SB34-2010

This book provides information on the DataInterchange Client/Server user interface.

- *DataInterchange Installation Guide*, GB09-8070

This book provides information about installing DataInterchange on MVS and CICS systems.

- *DataInterchange Messages and Codes*, SB34-2000

This book provides information to assist you in diagnosing errors.

- *Expedite/MVS Host Programming Guide*, GC34-2242

This book provides a description of the MVS (batch) interface for the IBM Global Network.

- *Expedite Base/MVS Programming Guide*, GC34-2204

This book provides a description of the Expedite Base/MVS communications program IEBASE.

- *Harbinger Gateway User Guide*

This book provides information about the Harbinger In*Touch Gateway.

- *Information Exchange Interface Programming Guide*, GC34-2222

This book provides information about programming for the Information Exchange interface.

- *Using EDI VAN Interconnect*, GC34-2263

This book provides information about using EDI VAN Interconnect to connect to multiple networks.

- *Using Expedite/CICS Display Application*, GC34-3303

This book provides information about the Expedite/CICS Display Application.

- *Using Information Exchange Administration Services*, GC34-2221

This book explains all the major administrative tasks and describes how to use them.

- *VS COBOL II Application Programming Guide*, SC26-4045

This book provides information on run-time parameters for improving performance.

Chapter 1. Using the DataInterchange Utility

Command Language Syntax	1-1
Command Structure	1-1
Command Language Validation	1-4
Error Filtering	1-4
Overriding the Utility Condition Codes	1-6
Outbound Processing	1-6
TRANSLATE TO STANDARD Command	1-6
ENVELOPE and REENVELOPE Commands	1-7
Sending EDI Data	1-10
Sending Non-EDI Data	1-10
Restart Send of EDI Data	1-11
Translating and Enveloping Transactions	1-11
Enveloping and Sending or Reenveloping and Sending Transaction	1-12
Translating and Sending Transactions	1-15
Fixed-to-Fixed (Runtime)	1-16
TRANSLATE TO STANDARD Command	1-16
ENVELOPE Command	1-16
REENVELOPE Command	1-17
SEND command	1-17
TRANSLATE AND ENVELOPE Command	1-17
TRANSLATE AND SEND Command	1-17
ENVELOPE AND SEND Command	1-18
REENVELOPE AND SEND command	1-18
SAP Status Tracking in DataInterchange	1-18
SAP Status Extract	1-19
Removing SAP Status From Database	1-20
Inbound Processing	1-20
Receiving EDI Data	1-21
Receiving Non-EDI Data	1-21
Restart Receive of EDI Data	1-22
Deenveloping EDI Data	1-22
Translating or Retranslating EDI Data to Application Format	1-23
Receiving and Deenveloping EDI Data	1-25
Deenveloping and Translating EDI Data	1-25
Receiving and Translating EDI Data	1-26
Managing Data	1-27
Reconstructing Envelopes	1-28
Update Network Status	1-29
Processing Received Network Acknowledgments	1-29
Hold, Purge, Release, and Unpurge	1-30
Purging Documents From the Transaction Store	1-32
Removing and Archiving Event Log Entries	1-33
Updating Management Reporting Statistics	1-38
Removing Management Reporting Statistics	1-38
Reporting and Data Extraction	1-39
Printing Transaction Store Reports	1-39
Query Transaction	1-47
Reporting Using Management Reporting and Transaction Store	1-48
Steps for Creating Management Reporting Reports	1-49
Management Reporting Trading Partner Profile Data Extract	1-50

Management Reporting Trading Partner Capability Data Extract	1-51
Management Reporting Transaction Activity Data Extract	1-53
Management Reporting Network Activity Data Extract	1-55
Transaction Store Envelope or Transaction Data Extract Overview	1-57
Creating Transaction Store Reports	1-58
Customizing	1-61
Migrating Trading Partner Transactions	1-61
Exporting and Importing	1-62
Exporting Setup Information	1-62
Importing Setup Information	1-63
Managing Mailboxes	1-63
Profile Maintenance	1-64
Querying Profiles	1-64
Deleting Profile Members	1-64
Continuous Receive Functions	1-65
Starting Continuous Receives	1-65
Stopping Continuous Receives	1-65
Reporting Continuous Receive Statuses	1-66
Persistent Environment Debugging	1-67
Obtaining a Persistent Environment Dump	1-67
Obtaining a Persistent Environment Trace	1-68
Keyword, Option, Criteria, and Range Criteria Descriptions	1-69
Running the DataInterchange Utility in MVS	1-116
Examples of Using Optional Parameters	1-117

Chapter 1. Using the DataInterchange Utility

The DataInterchange Utility provides command level access to DataInterchange services. These services can be split into the following categories:

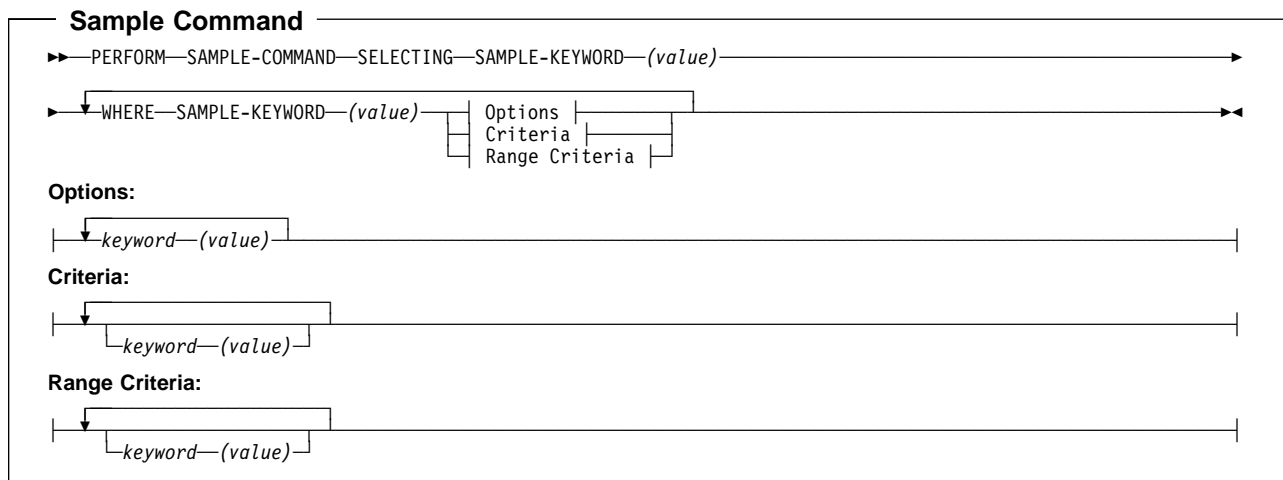
- Outbound processing of EDI documents:
 - Translating to the EDI standard format
 - Enveloping
 - Sending
- Inbound processing of EDI documents:
 - Receiving
 - Deenveloping
 - Translating to the application format
- Managing data including:
 - Updating records
 - Removing records
 - Controlling status of records
- Reporting and data extraction:
 - Formatting and printing reports
 - Printing application data
 - Extracting data
- Customizing:
 - Migrating maps
 - Exporting and importing administrative data
- Managing mailboxes (CICS only):
 - Closing mailboxes

Command Language Syntax

This section provides general information about using the DataInterchange Utility command language. You can invoke the DataInterchange Utility from an MVS environment or an MVS-CICS environment. For additional information about invoking the DataInterchange Utility in an MVS environment, see “Running the DataInterchange Utility in MVS” on page 1-116. For additional information about invoking the DataInterchange Utility in a CICS environment, see Chapter 5, “Using DataInterchange in the CICS Environment.”

Command Structure

The DataInterchange Utility command language consists of PERFORM statements, and WHERE and SELECTING clauses. It is graphically represented as follows:



The PERFORM statement defines the action DataInterchange takes. The following statement consists of a PERFORM keyword followed by a TRANSLATE TO APPLICATION command:

```
PERFORM TRANSLATE TO APPLICATION
```

The WHERE clause supplies selection criteria and information DataInterchange needs to process your request. A WHERE clause consists of the word WHERE, one or more keywords, and their associated values. Associated values must be enclosed in parentheses. For example:

```
WHERE TPNICKN (PISCES)
```

Note: The parentheses delimiter can be redefined with the DLM parameter. For more information, see “Running the DataInterchange Utility in MVS” on page 1-116.

A statement can include more than one WHERE clause. Each WHERE clause can contain more than one keyword. You cannot repeat the same keyword in one clause. The following is an example of an entire command statement:

```
PERFORM TRANSLATE TO APPLICATION
WHERE TPNICKN (PISCES) TRXDATE (12/14/97)
```

The previous statement tells DataInterchange to translate to application format all EDI documents that were received from a trading partner whose name is PISCES, and whose documents were placed in the Transaction Store on December 14, 1997. DataInterchange processes only those documents that meet both of the conditions specified in the WHERE clause. When you specify all of your conditions in one WHERE clause, DataInterchange limits its processing to only those objects that satisfy all the conditions. If you want DataInterchange to process EDI documents with either condition, use two separate WHERE clauses. For example:

```
PERFORM TRANSLATE TO APPLICATION
WHERE TPNICKN (PISCES)
WHERE TRXDATE (12/14/97)
```

Some statements use a SELECTING clause. Like the WHERE clause, it provides additional selection criteria. However, you can only include one SELECTING clause in the PERFORM statements. The following is an example of an entire command statement that uses a SELECTING clause:

```
PERFORM ENVELOPE DATA EXTRACT SELECTING
INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y)
WHERE TPNICKN(PISCES) INTCTLNO(000008888) DIR(S)
```

The command language format is free-form. You can leave blanks between elements of the statement. Characters can be in upper, lower, or mixed case. Most fields are not case sensitive. To determine if a field is case sensitive, see “Keyword, Option, Criteria, and Range Criteria Descriptions” on page 1-69.

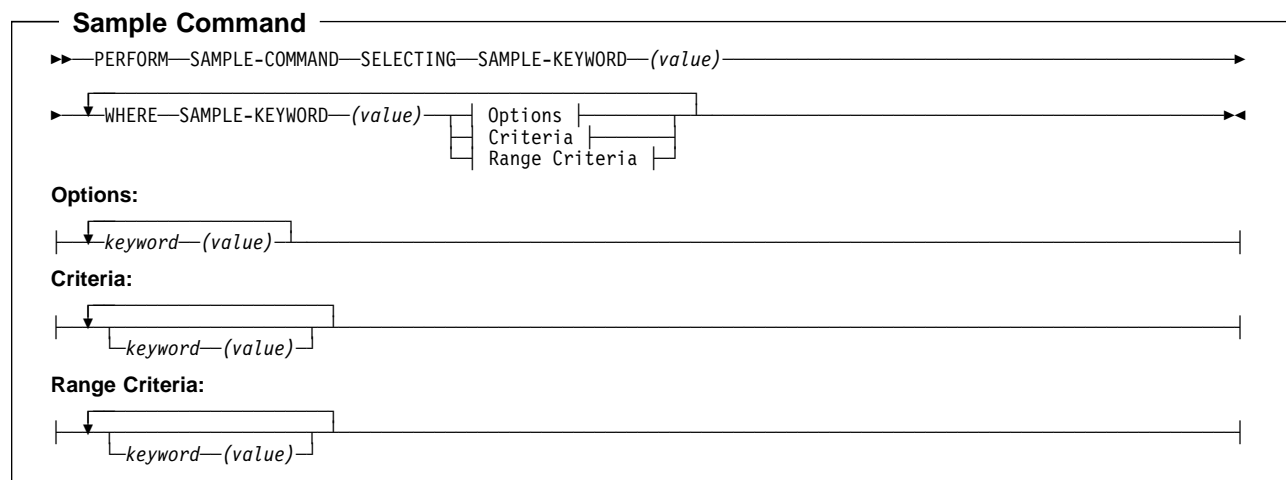
To insert comments between commands, place an asterisk (*) in column one and type the text on the same line. For example:

```
//SYSIN DD *
*
*This command translates my data into standard format
*
PERFORM TRANSLATE TO STANDARD
WHERE APPFILE(APDATA01)
/*
```

Your input can include more than one PERFORM statement. The DataInterchange Utility handles each statement separately. It also verifies the syntax of a statement and processes it before starting on the next statement. However, processing stops if a nonzero return code is received.

Options, Criteria, and Range Criteria Keywords

The DataInterchange Utility has special keyword groups that are graphically represented as follows:



The keyword groups have the following restrictions:

Keyword Group Keyword Restrictions

- | | |
|----------------|---|
| Options | Each keyword can be used only once in a WHERE clause. |
| Criteria | Each keyword can be used only once in a WHERE clause. |
| Range Criteria | Each keyword can be used only once in a WHERE clause. You can follow the keyword with a pair of values separated by the keyword TO. The pair defines a range of values for selecting the transactions you want to use. The first value indicates the low end of the range. The second value indicates the high end of the range. You can also specify a single value by omitting the keyword TO and the second value. |

Date, Time, and Handle Format Keywords

Several command keywords are used to identify date or time. The format of the date and time values must conform to the Date mask and Time mask fields in the language profile. You can use an asterisk (*) for the current date or time. For the date, you can also use an *-n , where n is the number of days before today's date.

The HANDLE keyword value format is:

YYYYMMDDHHMMSSnnnnnn

You can use an asterisk (*) to assign the current date to YYYYMMDD, or you can use an *-n , where n is the number of days before today's date.

Command Language Validation

DataInterchange performs basic validation on the command language input. This includes:

- Each PERFORM command (such as PERFORM TRANSLATE TO STANDARD) is validated. DataInterchange generates an error if the command is not valid.
- All keywords are validated. DataInterchange generates an error if a keyword is not valid.
- If a keyword is specified, then an associated value must also be specified. DataInterchange generates an error if you do not specify a keyword value.
- The length of each keyword value is validated against the maximum allowable length for the corresponding keyword. DataInterchange generates an error if the length of the value you supply is greater than the maximum allowable length for the keyword.
- Keywords cannot be duplicated in the same WHERE clause. However, you can specify multiple WHERE clauses. DataInterchange generates an error if you specify a keyword more than once in the same WHERE clause.
- Beginning and ending delimiters must be specified around each keyword value. DataInterchange generates an error if you do not enclose a keyword value with delimiters. The default delimiters are open and closed parentheses.
- Date and time fields are validated. DataInterchange generates an error if the date or time values you specify are not valid.
- When a keyword has an associated default value, keyword values that are not valid are ignored, and defaults are used. An example of a keyword value that is not valid is RAWTEST(X). In this situation, DataInterchange ignores the value of X and does not generate an error. Check each keyword for possible default values.
- If a keyword is used as part of a selection criteria on the Transaction Store, the associated value is accepted. If the associated value is not valid, DataInterchange uses the value as part of the selection criteria, and does not find a transaction match. Check each keyword used for Transaction Store selection criteria for associated valid values.

Error Filtering

A special named variable called DIERRFILTER may be used in the DataInterchange mapping setup to tell the translator to ignore certain error conditions. DIERRFILTER may also be used as a keyword on the Utility PERFORM command.

When the DIERRFILTER variable is either SET or SAVED during the DataInterchange mapping process,

the translator will take special notice and parse the value of the variable that will indicate which errors should be ignored.

The value of the DIERRFILTER variable should be a list of error codes that should be ignored. The error code values that should be specified are the unique codes documented in “Translator Unique Error Code Values” on page A-28 and in Appendix B, “DataInterchange Utility Condition Codes and API Return Codes,” where translator errors are documented.

For example, if you are not interested in the mandatory data element missing error (TR0001-101), or the data element is too short error (TR0003-103), then these can be filtered out with:

```
&SET DIERRFILTER 101,103
```

A range of values is allowed within the variable to allow many errors to be filtered without typing each one of them. For example, all warning messages, in addition to the 101 and 103, would be eliminated with:

```
&SET DIERRFILTER 101,103,1-99
```

DIERRFILTER is a named variable, and operations like the following could be used to eliminate validation errors, along with those errors previously set:

```
&SET DIERRFILTER &E(DIERRFILTER + ',116')
```

There are certain errors that occur prior to any translation being done, even before a map is determined. Most of these errors fall into the warning category. Therefore, you can use the following PERFORM command to initially eliminate all the warning errors.

```
PERFORM TRANSLATE TO STANDARD WHERE DIERRFILTER(1-99)
```

The DIERRFILTER variable has a transaction scope and therefore the value set from one transaction will not propagate to the next transaction. The translator will initially set the DIERRFILTER variable to the value specified in the PERFORM command at the start and end of each transaction.

There are certain errors that you cannot turn off. For example, TR0053 (unique code of 204) cannot be filtered because the message is too important, as it indicates data is being ignored. There will not be any error message given if an attempt is made to filter an error that is not filterable. The request will just be ignored.

It could be that a map contains sets of DIERRFILTERs, but something is not working correctly and it is necessary to see all possible error messages. The map will not have to be changed, but rather the DIERRFILTER keyword with a value of IGNORE can be used to tell DataInterchange that all errors should be reported.

```
PERFORM TRANSLATE TO STANDARD WHERE DIERRFILTER(IGNORE)
```

When an error is filtered, it does not mean the error did not occur. Eliminating the message and its corresponding condition code does not alter the fact that the error occurred. For example, a TR0004 error message can be issued if a validation edit fails. This error can be filtered so as not to produce the TR0004 message, but doing so does not alter the fact that the edit failed. Filtering only serves to suppress the error message and its condition code. Other than this, translation processing is unaffected.

However, if an error is filtered that would normally generate a functional acknowledgment, filtering the error eliminates the functional acknowledgment as well. If it is not considered an error because of filtering, it will not be considered an error to be reported in a functional acknowledgment.

Also, messages TR0101, TR0103, TR0151, TR0153, TR0203 and TR0205 are special. These errors indicate that the ST/SE, GS/GE, or ISA/IEA are inconsistent with respect to control numbers or control

counts. If these errors are filtered, then DataInterchange will process the message, group, or interchange even though the inconsistency exists.

Overriding the Utility Condition Codes

The DataInterchange Utility returns a condition code to MVS that can be tested in the JCL. In those cases where you want to ignore certain condition codes, the keywords IFCC and SETCC can be used on any PERFORM statement to override up to 10 different condition codes.

Note: When overriding condition codes, do not ignore important error conditions.

These keywords are entered as follows:

- IFCC(c1,c2,c3,c4,c5,c6,c7,c8,c9,c10)
The values c1 - c10 are the utility condition codes to be checked.
- SETCC(n1,n2,n3,n4,n5,n6,n7,n8,n9,n10)
The values n1 - n10 are the values that will be used to override the IFCC condition codes c1–c10, respectively.
- If SETCC keyword is not specified, all codes specified in the IFCC keyword are overridden to zero (0).

This is an example of an override of an empty application file condition on TRANSLATE TO STANDARD:

```
PERFORM TRANSLATE TO STANDARD WHERE APPFILE(APDATA01) IFCC(6) SETCC(0)
```

Note: Since zero (0) is the default override code, you can omit the SETCC keyword and get the same results.

Outbound Processing

The following set of utility commands serves these functions:

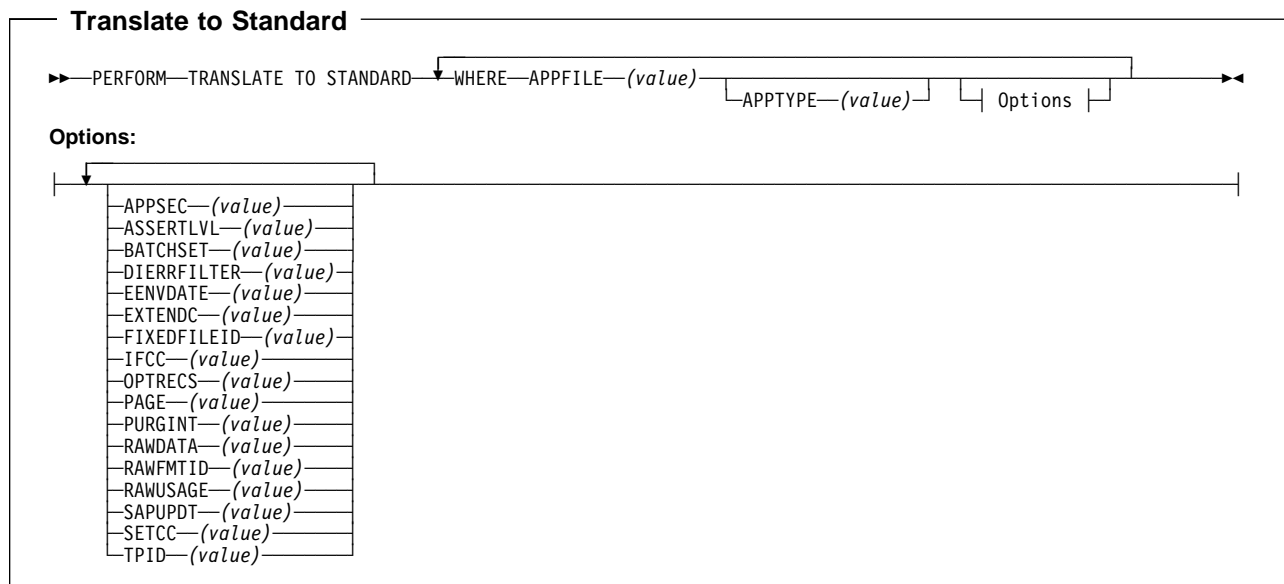
- Translate application data into a standard format and store this data into the Transaction Store.
- Envelope standard transactions or messages so they are ready to be sent.
- Send enveloped data to trading partners.

The commands are supplied to do each of these three steps independently or in combination. The combination commands can be used to improve performance in some cases. See the description of each command for more details.

| This version of DataInterchange allows you to designate print files, exception files, application files, etc. as
| MQSeries Queues.

| TRANSLATE TO STANDARD Command

The TRANSLATE TO STANDARD command translates application data to a standard format and places the results in the Transaction Store. The application data can be in C and D record format or in raw data format.



TRANSLATE TO STANDARD Command Examples

Example 1: Translate the transactions in a file named INVOICES from a raw data format to a standard format and place the results in the Transaction Store. The data format SENDINVOICES describes the raw data, Enter:

```
PERFORM TRANSLATE TO STANDARD
WHERE APPFILE(INVOICES) RAWFMTID(SENDINVOICES)
```

Example 2: Translate the transactions in two application files from C and D record format to standard format and place the results in the Transaction Store. Assign the batch ID 121497 to these transactions, Enter:

```
PERFORM TRANSLATE TO STANDARD
WHERE APPFILE(APDATA01) BATCHSET(121497)
WHERE APPFILE(APDATA02) BATCHSET(121497)
```

ENVELOPE and REENVELOPE Commands

The ENVELOPE command takes the EDI transactions from the Transaction Store that were previously enveloped, envelopes them, and puts the results in an envelope file. The envelope file is either the transaction data queue from the network profile member or a file specified by the command. Selection criteria determine which transactions go into the interchange envelope. The enveloper sorts the transactions to create the fewest number of functional groups and interchange envelopes. For more information, see the *DataInterchange Administrator's Guide*.

Envelope

►►—PERFORM—ENVELOPE—►WHERE— Criteria and/or Range Criteria | Options |

Criteria and/or Range Criteria:

APPLID—(value)	ACFIELD—(value)	TO—(value)
BATCH—(value)	EPURDATE—(value)	TO—(value)
FORMAT—(value)	HANDLE—(value)	TO—(value)
NETID—(value)	TRXDATE—(value)	TO—(value)
STDTRID—(value)	TRXTIME—(value)	TO—(value)
TPID—(value)		
TPNICKN—(value)		
TRERLVL—(value)		
TRXSTAT—(value)		

Options:

DIERRFILTER—(value)
ENVPRBREAK—(value)
FILEID—(value)
FIXEDFILEID—(value)
IACCESS—(value)
IAREA—(value)
IEXIT—(value)
ITYPE—(value)
IFCC—(value)
INMEMTRANS—(value)
ITPBREAK—(value)
OPTRECS—(value)
PAGE—(value)
RAWDATA—(value)
RECOVERY—(value)
SAPUPDT—(value)
SERVICESEGVAL—(value)
SETCC—(value)
VERIFY—(value)

The REENVELOPE command takes the EDI transactions from the Transaction Store that were previously enveloped, envelopes them, and places the results in an envelope file.

Reenvelope

PERFORM REENVELOPE WHERE Criteria and/or Range Criteria Options

Criteria and/or Range Criteria:

APPLID—(value)	ACFIELD—(value)	T0—(value)
APPRECID—(value)	ENVDATA—(value)	T0—(value)
APPSNDID—(value)	ENVTIME—(value)	T0—(value)
BATCH—(value)	EPURDATE—(value)	T0—(value)
ENVTYPE—(value)	GRPCTLNO—(value)	T0—(value)
FORMAT—(value)	HANDLE—(value)	T0—(value)
FUNACKP—(value)	INTCTLNO—(value)	T0—(value)
INTRECID—(value)	SNDDATE—(value)	T0—(value)
INTSNDID—(value)	SNDTIME—(value)	T0—(value)
NETACKP—(value)	TRXCTLNO—(value)	T0—(value)
NETID—(value)	TRXDATE—(value)	T0—(value)
NETSTAT—(value)	TRXTIME—(value)	T0—(value)
STDTRID—(value)		
TPID—(value)		
TPNICKN—(value)		
TRERLVL—(value)		
TRXSTAT—(value)		

Options:

DIERRFILTER—(value)
ENVPRBREAK—(value)
FILEID—(value)
FIXEDFILEID—(value)
IACCESS—(value)
IAREA—(value)
IEXIT—(value)
IFCC—(value)
ITYPE—(value)
INMEMTRANS—(value)
ITPBREAK—(value)
OPTRECS—(value)
PAGE—(value)
RAWDATA—(value)
RECOVERY—(value)
SAPUPDT—(value)
SERVICESEGVAL—(value)
SETCC—(value)
VERIFY—(value)

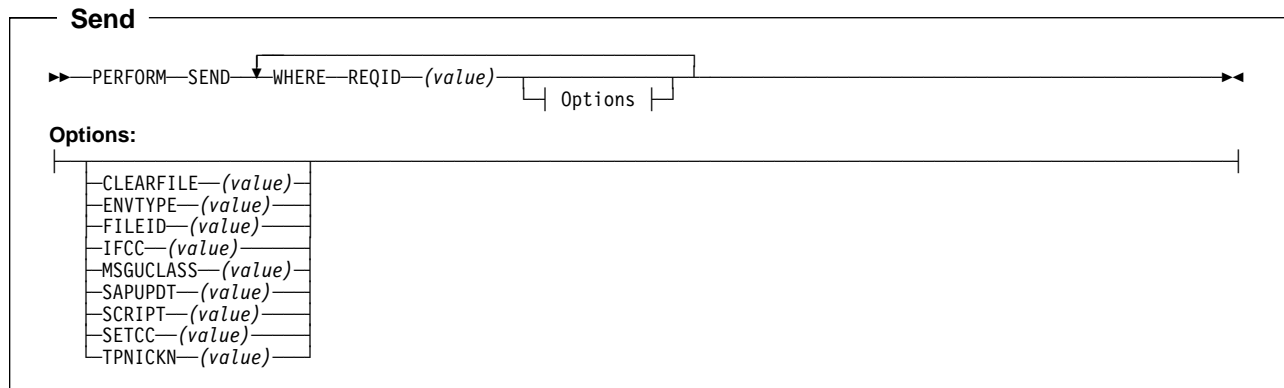
Example 1: Envelope all the EDI documents in the Transaction Store that are associated with application data format ID PO850 or CORP861, and with network IINB1. Put the results in a file named NETQUEUE. The NETID keyword is included in both WHERE clauses to make sure that only IINB1 transactions are selected. (Each WHERE clause is a separate set of selection criteria.)

Example 2: Reenvelope the EDI document with the Transaction Store handle 19971214101533000001. Place the results in the network transaction data queue.

Chapter 1. Using the DataInterchange Utility **1-9**

Sending EDI Data

The SEND command sends EDI transactions from an envelope file to the network. The envelope file is either the transaction data queue field from the network profile member or a file named in the command.



SEND Command Examples

Example 1: Send the transactions in the transaction data queue. Use the network specified by requestor profile member IINREQ.

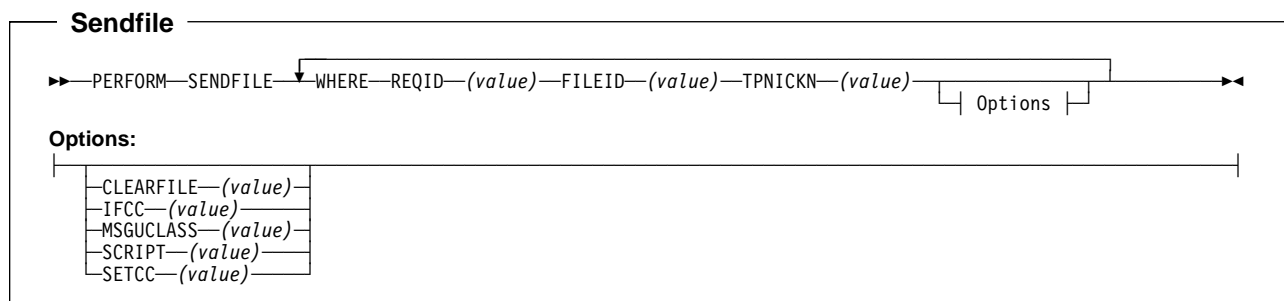
```
PERFORM SEND  
WHERE REQID(IINREQ)
```

Example 2: Send the transactions in the file named NETQUEUE. Use the network specified by requestor profile member IINREQ.

```
PERFORM SEND  
WHERE REQID(IINREQ) FILEID(NETQUEUE) ENVTYP(E)
```

Sending Non-EDI Data

The SENDFILE command sends a file (application data) that does not need an interchange header or contain EDI data.



SENDFILE Command Examples

Example 1: Send the data in the file named FLATFILE to trading partner MYTP.

```
PERFORM SENDFILE  
WHERE REQID(ME) FILEID(FLATFILE) TPNICKN(MYTP)
```

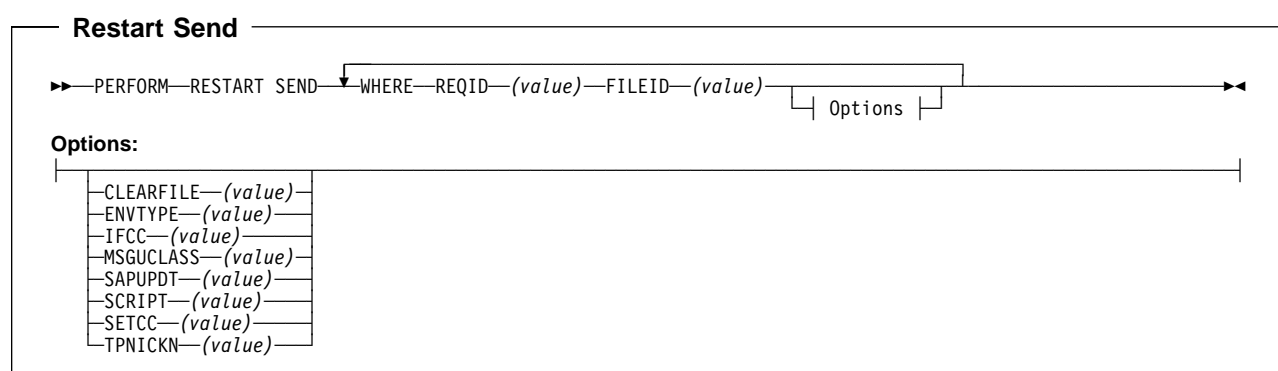
Example 2: Send the data in the file named BULK to trading partner YOURTP with a network message user class of SPECIAL.

```
PERFORM SENDFILE
WHERE REQID(ME) FILEID(BULK) TPNICKN(YOURTP) MSGUCLASS(SPECIAL)
```

Restart Send of EDI Data

The RESTART SEND command is valid only when the network supports restart and you specified checkpoint-level recovery when initially sending the data. If an error causes the network processing to enter a restart situation, the RESTART SEND can be used to restart and complete the send. For more information on checkpoint recovery, see the *Expedite Base/MVS Programming Guide*.

Note: The envelope file and the requestor ID must be the same values as specified on the initial send. Also, this command is not supported in CICS.



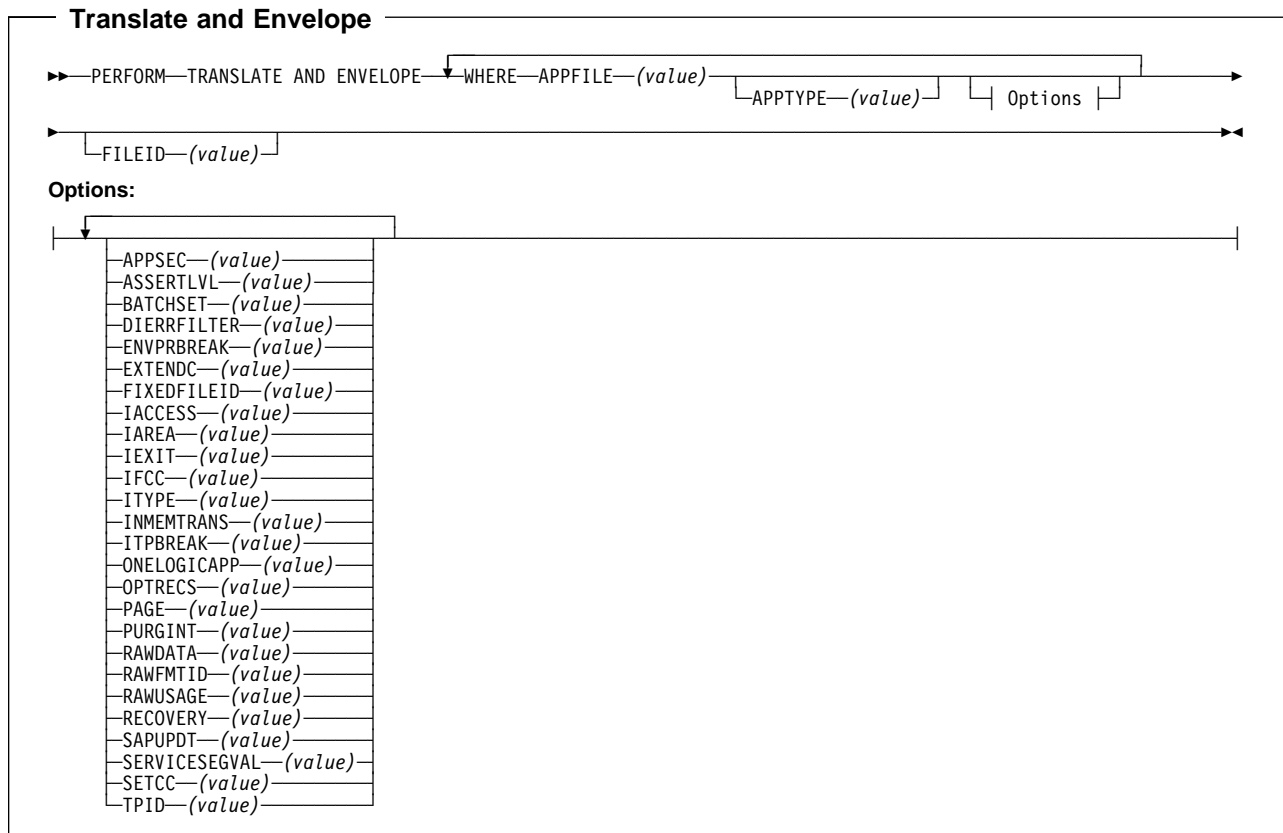
RESTART SEND Command Examples

Example 1: Restart the send of data from file NETQUEUE to the network defined in requestor profile IINREQ.

```
PERFORM RESTART SEND
WHERE REQID(IINREQ) FILEID(NETQUEUE)
```

Translating and Enveloping Transactions

The TRANSLATE AND ENVELOPE command combines the services of the TRANSLATE TO STANDARD and ENVELOPE commands. The command takes EDI transactions from an application file, translates them to EDI format, puts the results in the Transaction Store, and creates envelopes for the transactions. It puts the enveloped transactions in the transaction data queue or in a file specified in the command. This command performs faster than the separate commands, but it does not provide the optimum enveloping achieved with separate commands.



TRANSLATE AND ENVELOPE Command Examples

Example 1: Translate and envelope the transactions in a file with the data definition name (ddname) APPDATA. Place the enveloped transactions in the transaction data queue specified by the network profile. Return the information record created during translation.

```
PERFORM TRANSLATE AND ENVELOPE
WHERE APPFILE(APPDATA) OPTRECS(I)
```

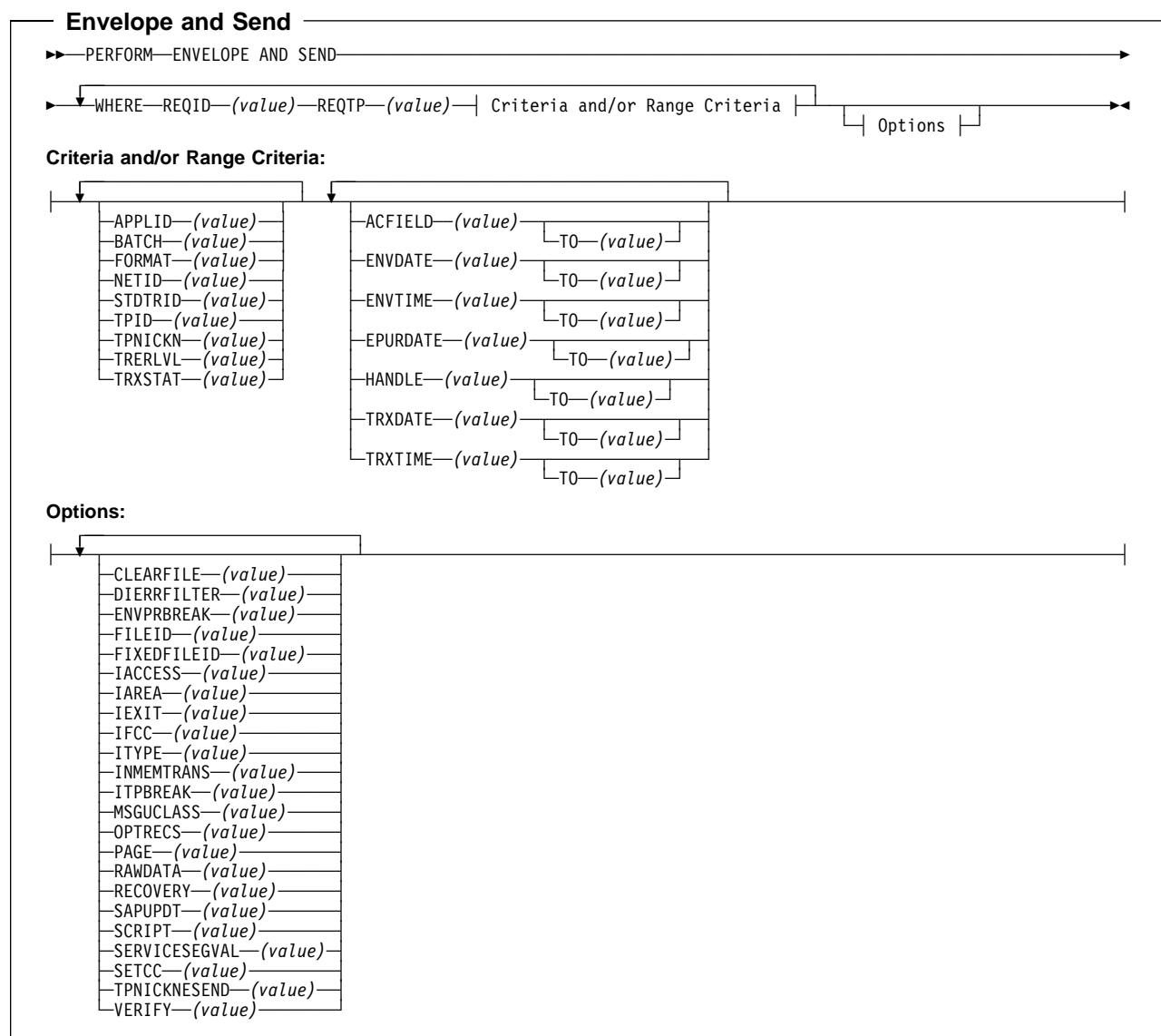
Example 2: In CICS, translate and envelope the transactions in a transient data queue named AP01. Place the enveloped transactions in the temporary storage queue named ENVDATA. Return the information record and the envelope headers created during translation.

```
PERFORM TRANSLATE AND ENVELOPE
WHERE APPFILE(AP01) APPTYPE(TD) FILEID(ENVDATA) OPTRECS(IE)
```

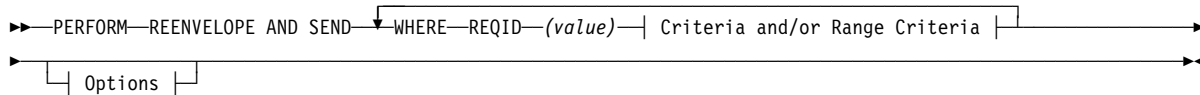
Enveloping and Sending or Reenveloping and Sending Transaction

The ENVELOPE AND SEND command combines the services of the ENVELOPE and SEND commands. It takes EDI transactions from the Transaction Store, envelopes them, and puts the results in a file with a *ddname* specified by the Transaction data queue field of the network profile member or in a file specified by the command. Selection criteria determine which transactions go into the interchange envelope. The enveloper sorts the transactions to create the fewest possible number of functional groups and interchange envelopes. (For more information on determining the number of groups and envelopes needed, see the *DataInterchange Administrator's Guide*.) The DataInterchange Utility then sends the enveloped transactions to the network.

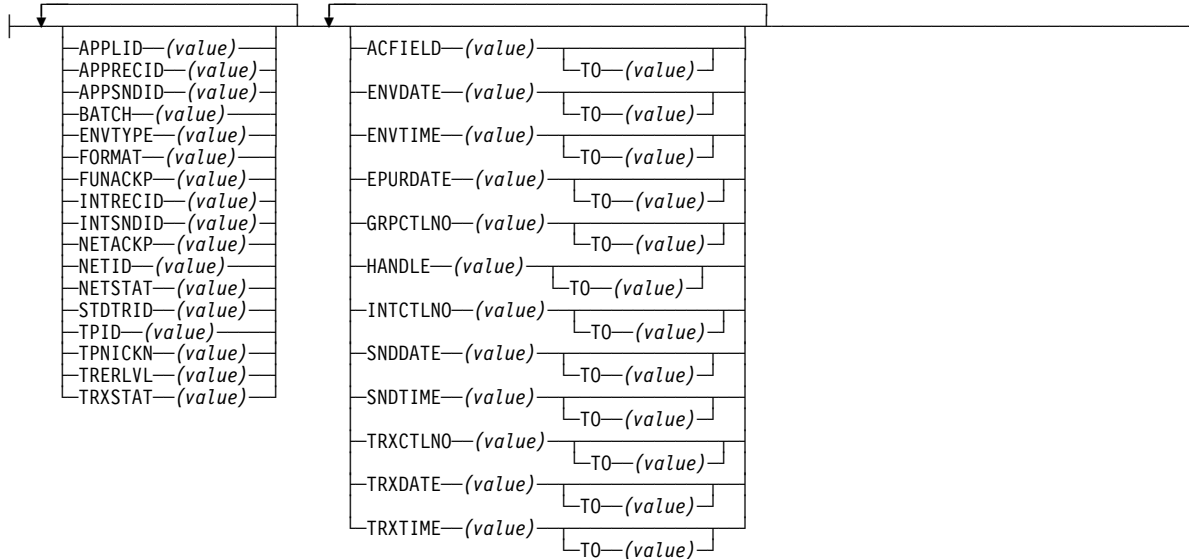
The REENVELOPE AND SEND command provides the same services, but for EDI data that was previously enveloped. It combines the services of the REENVELOPE and SEND commands. A requestor ID must be supplied for each network for which data is enveloped.



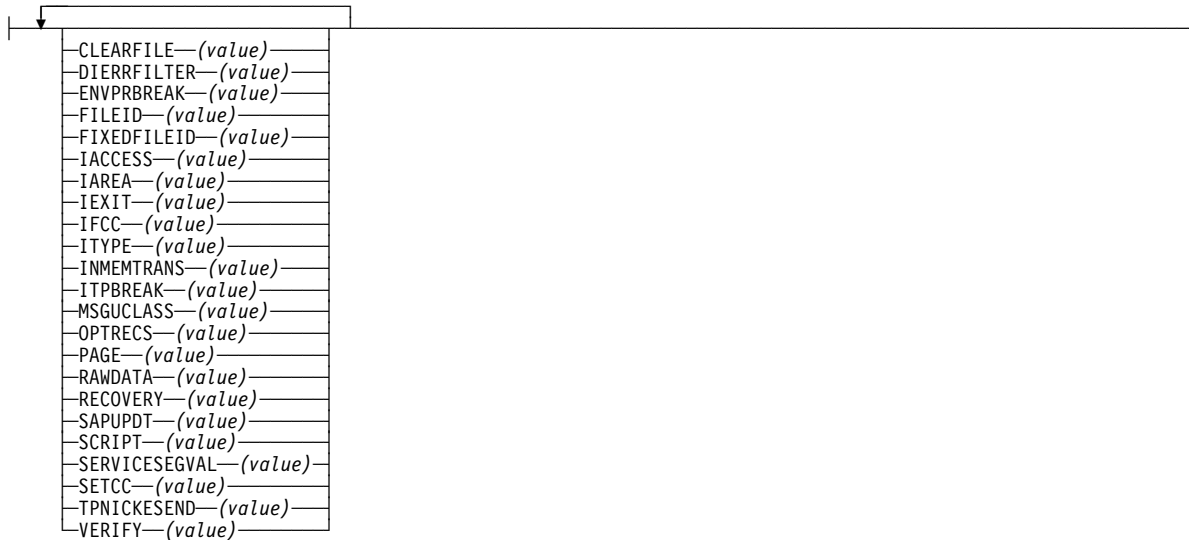
Reenvelope and Send



Criteria and/or Range Criteria:



Options:



ENVELOPE AND SEND Command Examples

Example 1: Envelope and send all documents in the Transaction Store that have a destination of trading partner PISCES.

```
PERFORM ENVELOPE AND SEND
WHERE REQID(IINREQ) TPNICKN(PISCES)
```

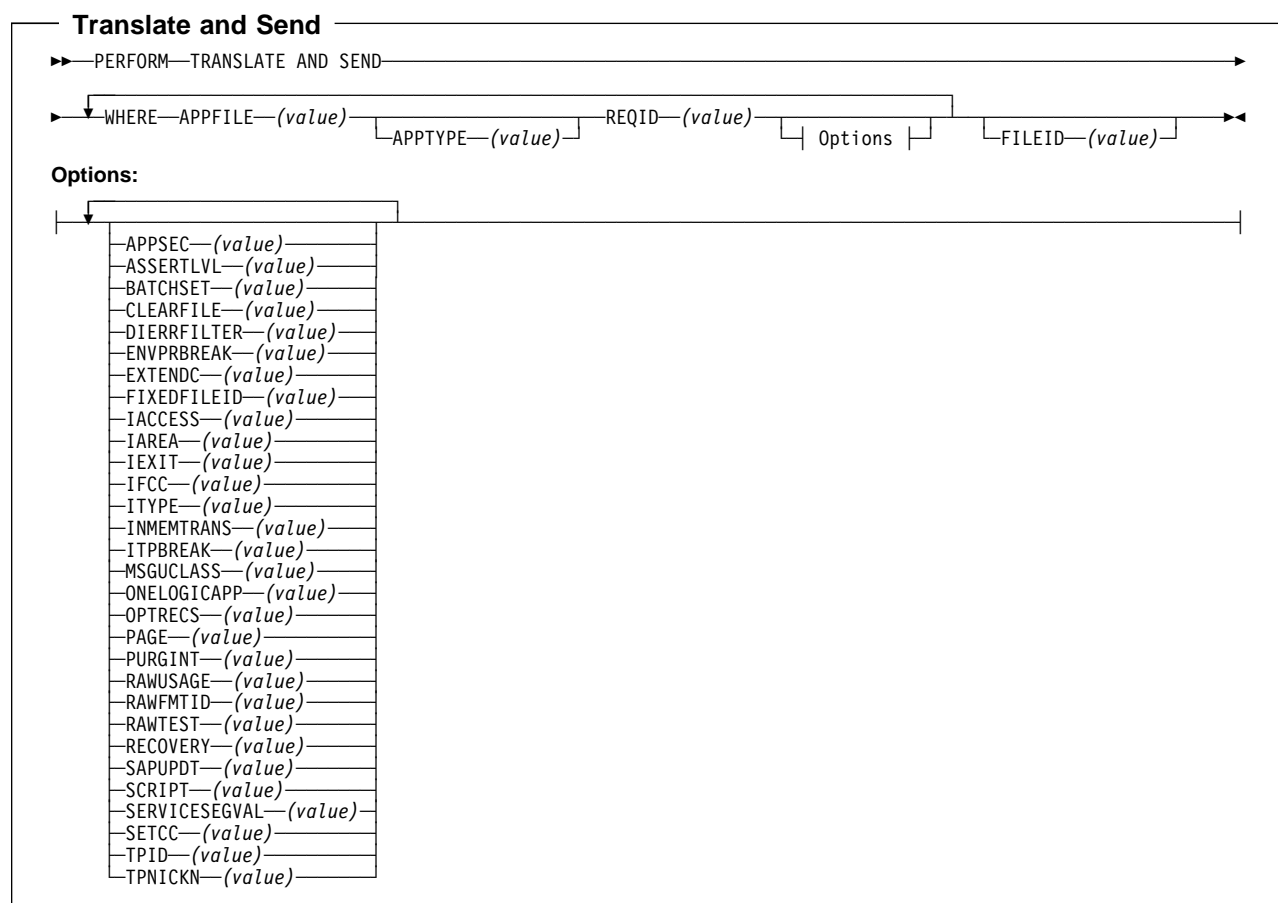
Example 2: Envelope and send all documents in the Transaction Store that you assigned the batch ID 121497.

```
PERFORM ENVELOPE AND SEND
WHERE REQID(IINREQ) BATCH(121497)
```


Note: These transactions will be sent using different networks.

Translating and Sending Transactions

Note: This command will not work properly with direct connection networks such as Point-to-Point or the Gateway if the APPFILE contains transactions for more than one trading partner.



Example 1: Translate and send the transactions in the file whose *ddname* is APPDATA. Assign the batch number 121497 to these transactions.

Chapter 1. Using the DataInterchange Utility **1-15**

Example 2: Translate and send the transactions in the file whose *ddname* is RAWFILE. Data format POSEND describes these transactions, which are in raw data format.

```
PERFORM TRANSLATE AND SEND  
WHERE APPFILE(RAWFILE) REQID(IINREQ) RAWFMTID(POSEND)
```

Fixed-to-Fixed (Runtime)

The processes that are used for Fixed-to-Fixed translation and enveloping are the same as those used for standard translations. Therefore, all of the outbound processing works basically the same for Fixed-to-Fixed as they do for standard translations. The following describes how Fixed-to-Fixed fits into each outbound process.

TRANSLATE TO STANDARD Command

There is no difference in operation if Fixed-to-Fixed translation is involved. It is the mapping that determines a Fixed-to-Fixed translation based on the standard used in the mapping. If the standard has an envelope type of F, then a Fixed-to-Fixed translation is done. If the standard has any other envelope type, then a standard translation is done. The application data being translated can be a mixture of data; some of which is destined for Fixed-to-Fixed translation; some of which is destined for standard translation. The results of the translation and image of the data produced will be saved in the Transaction Store. These outputs will be saved in the C and D record format. They will be eligible for a PERFORM ENVELOPE command. The RAWDATA keyword controls the type of data (raw or the C and D format), which is output from the ENVELOPE command. For more information on the command syntax and command examples, see “TRANSLATE TO STANDARD Command” on page 1-6.

ENVELOPE Command

The ENVELOPE command basically goes to the Transaction Store and retrieves all the transactions that are eligible to be enveloped and that match the selection criteria. It sorts them to obtain the fewest interchanges and groups. Then it builds and writes the interchanges to the appropriate file. The appropriate file for standard translated data in order of precedence is:

1. The file identified by the FILEID keyword
2. The file specified in the *Trans data queue* name of the network profile member (with E or U suffix)
3. QDATA by default (with E or U suffix)

The appropriate file for standard Fixed-to-Fixed translate data in order of precedence is:

1. The file identified by the FIXEDFILEID keyword
2. The *ddname* formed by the concatenation of the standard ID (which was the *Application file name* in the application data format that created the standard) with the File suffix

Fixed-to-Fixed translated data can either be written to the file in a C and D record format or in a RAWDATA format. The RAWDATA keyword on the PERFORM command controls which format is used. RAWDATA(Y) results in the rawdata format and anything else results in a C and D record format. If the target ADF has raw data specifications, a structure will be initialized with the Record ID value. If the structure contains the field specified to be the internal trading partner ID value, that field will be initialized with the Internal Trading Partner ID value from the usage. These values may be overlaid during the mapping process. The initialization takes place even when RAWDATA output has not been requested.

The D record format produced is always the format designed for use when there are multiple D records. This format consists of the Record ID value of D, followed by the name of the structure for 16 bytes, followed by the structure data itself. If the fixed translated data does not have an associated ADF, the segment ID from the standard definition is used as the structure name. For more information on the command syntax and command examples, see “ENVELOPE and REENVELOPE Commands” on page 1-7.

REENVELOPE Command

The REENVELOPE command performs the same as the ENVELOPE command. For more information on the command syntax and command examples, see “ENVELOPE and REENVELOPE Commands” on page 1-7.

SEND command

When SEND command is performed in a composite command such as ENVELOPE and SEND, DataInterchange remembers all the files created during an Envelope and will send each of them using the appropriate request to the network to send the data. Standard data will be sent with a request to send transaction data using either SENDEDI, SENDX12, SENDUCS, or SENDFILE (based on the envelope type used). Fixed translated data without an interchange layer (for example, ISA, UNB, and so forth) will be sent with a request to send a file using the SENDFILE command. Fixed translated data with an interchange layer will be sent with a request to send transaction data using either SENDEDI, SENDX12, SENDUCS, or SENDFILE (based on the envelope type used).

If you are actually doing the SEND single command, then you should do a SEND for the transaction data or fixed data with an interchange. For fixed data without an interchange, the new SENDFILE command should be used. The SENDFILE command sends a file to a specific trading partner. For more information on the command syntax and command examples, see “Sending EDI Data” on page 1-10 and “Sending Non-EDI Data” on page 1-10.

TRANSLATE AND ENVELOPE Command

The TRANSLATE AND ENVELOPE command works much the same as the TRANSLATE command, followed by the ENVELOPE command. The only difference is that since enveloping is occurring at the same time as translation, you will probably end up with more and smaller interchanges unless you sort the application data beforehand. See “Translating and Enveloping Transactions” on page 1-11 for the command syntax and command examples.

TRANSLATE AND SEND Command

The TRANSLATE AND SEND command works much the same as the TRANSLATE command, followed by the SEND command. The only difference is that since enveloping is occurring at the same time as translation, you will probably end up with more and smaller interchanges unless you sort the application data beforehand.

Also, with the combined function, DataInterchange keeps track of what needs to be sent. With separate functions, you need to keep track of what files were created and need to be sent. See “Translating and Sending Transactions” on page 1-15 for the command syntax and command examples.

ENVELOPE AND SEND Command

The ENVELOPE AND SEND command works the same as the ENVELOPE command followed by the SEND command, except that when they are combined, DataInterchange keeps track of what needs to be sent. If they are separated, you need to keep track of what files were created and need to be sent. See “Enveloping and Sending or Reenveloping and Sending Transaction” on page 1-12 for the command syntax and command examples.

REENVELOPE AND SEND command

The REENVELOPE AND SEND command works the same as the ENVELOPE AND SEND command. See “Enveloping and Sending or Reenveloping and Sending Transaction” on page 1-12 for the command syntax and command examples.

SAP Status Tracking in DataInterchange

SAP is a client/server application which supports business processes that include sales, materials management, and distribution. SAP was developed in Germany and supports an interface to an EDI subsystem. SAP has both mainframe and UNIX solutions.

- | SAP generates application data in the SAP Intermittent Document (IDOC) layout. The file is sent to the EDI subsystem (or translator) using a file transfer product such as File Transfer Protocol (FTP) or Transmission Control Protocol/Internet Protocol (TCP/IP) from UNIX to mainframe.

The mapping required to perform the translation of the inbound and outbound IDOCs is the user's responsibility. To assist in mapping the IDOC, a new mapping literal keyword called &THANDLE is provided to enable mapping of the DataInterchange archive key to the SAP IDOC for inbound processing. A new special DataInterchange variable name called DISAPSEQ allows saving of the IDOC record sequence number on the first error during outbound processing. The variable DISAPSEQ will be captured in the SAP status record to indicate the first record in error.

- | DataInterchange provides the capability to capture the SAP status information during different phases of the EDI process by specifying a new keyword, SAPUPDT, on the utility PERFORM commands. The SAP status tracking is only supported with the DataInterchange Utility.

- | A utility PERFORM command allows you to extract or remove the SAP status records from the database based on selection criteria, and write them (in SAP EDI_DS record format) to a sequential file for transfer to the SAP system. DataInterchange currently supports the SAP status EDI_DS record at IDOC releases 2, 3, and 4.

SAP Status Codes Supported by DataInterchange

The following status codes are supported by DataInterchange:

Code	Description
04	Error within control information of EDI subsystem
05	Error during translation process
06	Translation successful
09	Error during interchange handling
10	Interchange handling successful

- 11 Error during dispatch
- 12 Dispatch successful
- 16 Functional acknowledgment positive
- 17 Functional acknowledgment negative
- 22 Dispatch successful, acknowledgment still due

Outbound Processing

The capture of SAP status information is under the user's control using a new keyword, SAPUPDT, on the utility PERFORM statements. If SAPUPDT(Y) is specified, the SAP status along with the DataInterchange archive key is captured in a new database (EDIVSSTK). The status is updated at various key points during EDI processing.

The SAPUPDT(Y) keyword is required on the following DataInterchange Utility PERFORM commands for outbound processing when using the SAP status tracking:

```
PERFORM TRANSLATE TO STANDARD
PERFORM TRANSLATE AND ENVELOPE
PERFORM TRANSLATE AND SEND
PERFORM ENVELOPE
PERFORM REENVELOPE
PERFORM ENVELOPE AND SEND
PERFORM REENVELOPE AND SEND
PERFORM SEND
PERFORM RESTART SEND
```

Inbound Processing

The capture of SAP status information for functional acknowledgments is under the user's control using the new keyword, SAPUPDT, on the utility PERFORM statements. If SAPUPDT(Y) is specified, the functional acknowledgment status is captured in the database. When the Transaction Store is active, you can capture the SAP status information.

The SAPUPDT(Y) keyword is required on the following DataInterchange Utility PERFORM commands for inbound processing when using the SAP status tracking for functional acknowledgments:

```
PERFORM DEENVELOPE
PERFORM DEENVELOPE AND TRANSLATE
PERFORM RECEIVE AND DEENVELOPE
PERFORM RECEIVE AND TRANSLATE
```

SAP Status Extract

The following PERFORM statement is used to extract SAP status records from the DataInterchange database, and write them to the file specified in keywords OUTFILE and OUTTYPE. The output file defines a record length that is at least as large as the SAP status EDI_DS record. The utility returns a completion code of 792 if no records meet the selection criteria. If the records are truncated in the output file, an error occurs with a return code of 796.

```
PERFORM SAP STATUS EXTRACT
WHERE OUTFILE() OUTTYPE() CLIENT() SAPSTAT() DAYS()
```

Table 1-1. SAP Extract Keywords

Keyword	Description
OUTFILE	The file name to which the extracted SAP status records are written. Default is SAPOUT.
OUTTYPE	The file type (MQ, TD, TM, TS, VS). In MVS, the default file name is the ddname of a sequential file. In CICS, the default is TS.
CLIENT	Extract records by SAP client ID. Default is all.
SAPSTAT	The SAP status value to extract or used with TO(), for a range of values to extract. Acceptable values are 04–22. Default is all status. For example, SAPSTAT(04) TO(22) would extract those status values.
DAYS	The date of the records to extract or used with TO(YYYYMMDD), for a range of dates to extract. The range is inclusive. Default is all days. For example, DAYS(*–5) TO(*) would represent the previous 5 days.
TO	When used with the SAPSTAT or DAYS, this specifies the end value of a range of values.

Removing SAP Status From Database

The following PERFORM statement removes the SAP status from the DataInterchange database. The utility returns a completion code of 796 if an error occurs.

```
PERFORM SAP STATUS REMOVE
WHERE CLIENT() PRIORTO() SAPSTAT() TO()
```

Table 1-2. Keywords For Removing SAP Status

Keyword	Description
CLIENT	Remove records by SAP client ID. Default is all.
SAPSTAT	The SAP status value to remove or used with TO(), for a range of values to remove. Acceptable values are 04–22. Default is all status. For example, SAPSTAT(04) TO(22) would remove those status values.
PRIORTO	The date prior to which the records are to be removed. Default is all dates.
TO	When used with the SAPSTAT, this specifies the end value or range of values.

Inbound Processing

The following set of utility commands serves these functions:

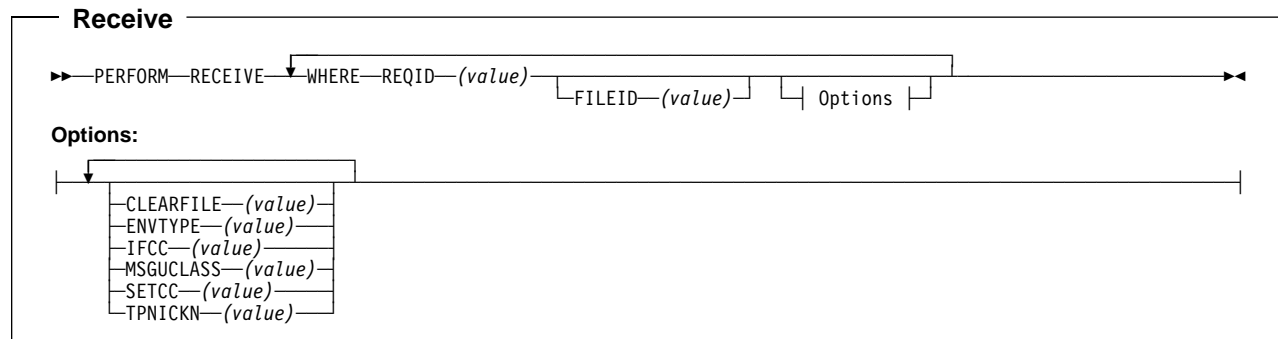
- Receive data from trading partners.
- Deenvelope interchanges and place the standard transactions or messages into the Transaction Store.
- Translate standard transactions or messages into application formats.

Commands are supplied to do each of these three steps independently or in combination. The combination commands normally improve overall performance. However, PERFORM RECEIVE combination commands cannot be used in the CICS environment when Management Reporting is turned on and SYNCVAL is -1. See “Format of DataInterchange Utility Control Information” on page 5-18 for more information on SYNCVAL.

This version of DataInterchange allows you to designate print files, exception files, application files, etc. as MQSeries Queues.

Receiving EDI Data

The RECEIVE command receives EDI data from the network indicated in the requestor profile member. It puts the EDI data in the receive file specified in the requestor profile or in the override file specified in the command.



RECEIVE Command Examples

Example 1: Receive data for requestors defined by requestor profile members DEPTA7F and DEPTB8R. Place the data into the receive files specified by the profile members. Clear both files before the data is received. The data is in X12 format.

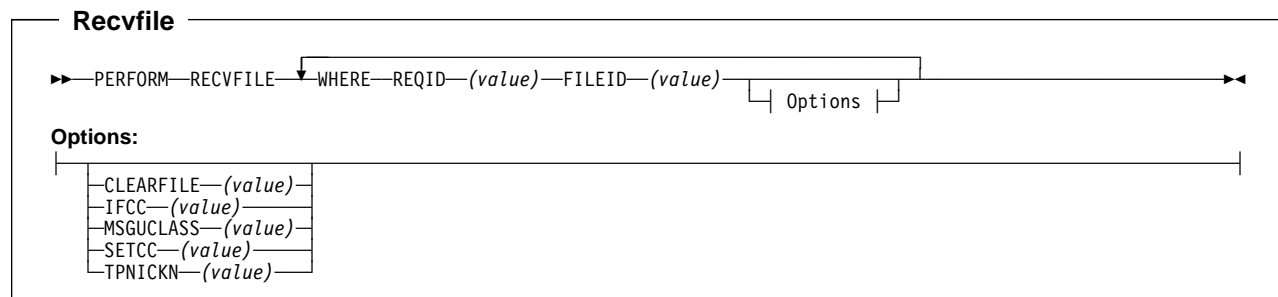
```
PERFORM RECEIVE
WHERE REQID(DEPTA7F) CLEARFILE(Y)
WHERE REQID(DEPTB8R) CLEARFILE(Y)
```

Example 2: In CICS, receive data for the requestor defined by requestor profile member DEPTA7F and place the data in the CICS temporary storage queue named RECVPO. The data is in EDIFACT format.

```
PERFORM RECEIVE
WHERE REQID(DEPTA7F) FILEID(RECVPO) ENVTYP(E)
```

Receiving Non-EDI Data

The RECVFILE command receives a file that does not need an interchange header or contain EDI data.



RECVFILE Command Examples

Example 1: Receive data for the requestor defined by requestor profile member DEPTA7F. Place the data into file RCVFILE. Clear RCVFILE before the data is received. The data is not necessarily in any standard format.

```
PERFORM RCVFILE
WHERE REQID(DEPTA7F) FILEID(RCVFILE) CLEARFILE(Y)
```

Restart Receive of EDI Data

The RESTART RECEIVE command is valid only when the network supports restart and you have specified checkpoint-level recovery when initially receiving the data. If an error causes the network processing to enter a restart situation, the RESTART RECEIVE can be used to restart and complete the receive. For more information on checkpoint recovery, see the *Expedite Base/MVS Programming Guide*.

Note: The requestor ID must be the same value as specified on the initial receive. This command is not supported in the CICS environment.

Restart Receive

```
►►—PERFORM—RESTART RECEIVE—┐ WHERE—REQID—(value) ┘
```

RESTART RECEIVE Command Examples

Example 1: Restart the receive of data from the network defined in requestor profile DEPTA7F.

```
PERFORM RESTART RECEIVE
WHERE REQID(DEPTA7F)
```

Deenveloping EDI Data

The DEENVELOPE command takes EDI transactions from the envelope file, removes the envelope segments, and puts the results in the Transaction Store. The envelope file is either the receive file specified in the requestor profile or a file named in the command.

Deenvelope

```
►►—PERFORM—DEENVELOPE—┐ WHERE—REQID—(value) ┘
                        └ FILEID—(value) ┘ Options ┘
```

Options:

```
┌ DIERRFILTER—(value) ─┐
├ DUPENV—(value) ─┐
├ EXTENDC—(value) ─┐
├ FADELAY—(value) ─┐
├ FORCETEST—(value) ─┐
├ FUNACKFILE—(value) ─┐
├ FUNACKREQ—(value) ─┐
├ IAREA—(value) ─┐
├ IEXIT—(value) ─┐
├ IFCC—(value) ─┐
├ INMEMTRANS—(value) ─┐
├ MRREQID—(value) ─┐
├ OPTRECS—(value) ─┐
├ PAGE—(value) ─┐
├ PURGINT—(value) ─┐
├ RECOVERY—(value) ─┐
├ SAPUPDT—(value) ─┐
├ SERVICESEGVAL—(value) ─┐
└ SETCC—(value) ─┐
```


DEENVELOPE Command Examples

Example 1: Deenvelope the transactions previously received for requestor DEPTA7F and put the results in the Transaction Store. Indicate that these transactions are to be marked for purging after 10 days in the store.

```
PERFORM DEENVELOPE
WHERE REQID(DEPTA7F) PURGINT(10)
```

Example 2: Deenvelope the transactions in the file whose ddname is A7FIN and put the results in the Transaction Store. Do not allow duplicate interchanges to be processed.

```
PERFORM DEENVELOPE
WHERE FILEID(A7FIN) DUPENV(N)
```

Translating or Retranslating EDI Data to Application Format

The TRANSLATE TO APPLICATION command takes EDI documents from the Transaction Store, translates them to application format, and puts the results in the file specified by the data format or its override in the trading partner usage for the transaction mapping. The data can be formatted as C and D records or as raw data.

The RETRANSLATE TO APPLICATION command provides the same services, but for transactions that were previously translated.

Note: In CICS, you can also deliver the data to a program or CICS transaction, as specified in the data format fields **Application file name** and **Application file type**.

Translate to Application

►►—PERFORM—TRANSLATE TO APPLICATION—WHERE— Criteria and/or Range Criteria | Options |►►

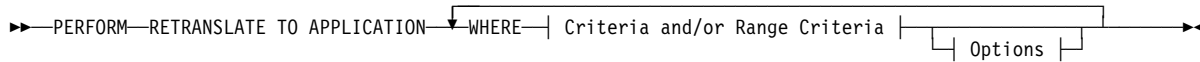
Criteria and/or Range Criteria:

APPSNDID—(value)	ACFIELD—(value)	TO—(value)
APPRECID—(value)	EPURDATE—(value)	TO—(value)
ENVTYPE—(value)	GRPCTLNO—(value)	TO—(value)
FORMAT—(value)	HANDLE—(value)	TO—(value)
INTRECID—(value)	INTCTLNO—(value)	TO—(value)
INTSNDID—(value)	TRXCTLNO—(value)	TO—(value)
NETID—(value)	TRXDATE—(value)	TO—(value)
STDTRID—(value)	TRXTIME—(value)	TO—(value)
TPID—(value)		
TPNICKN—(value)		
TRXSTAT—(value)		

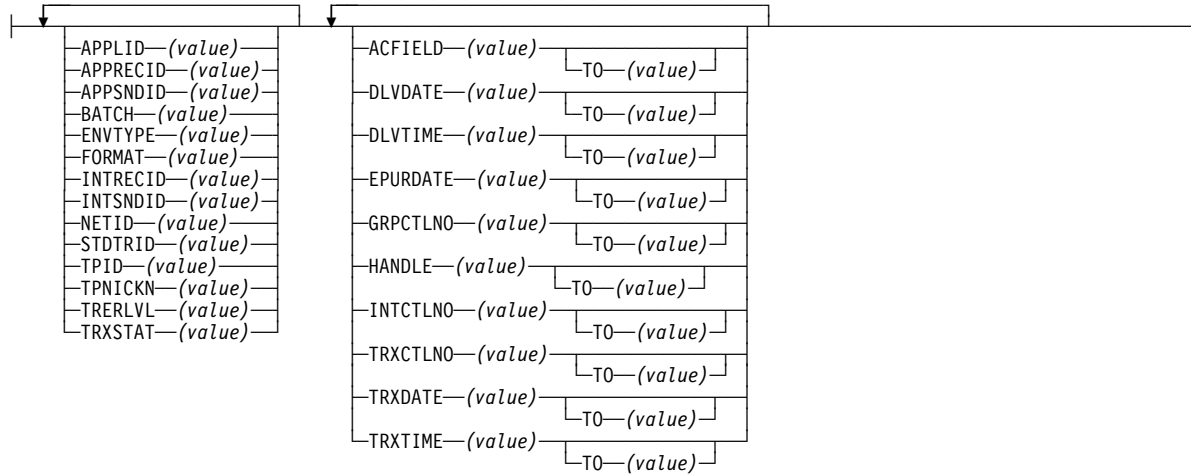
Options:

ASSERTLVL—(value)
BATCHSET—(value)
CCEXCEPTION—(value)
DIERRFILTER—(value)
EXTENDC—(value)
FORCETEST—(value)
IFCC—(value)
OPTRECS—(value)
PAGE—(value)
RAWDATA—(value)
SETCC—(value)

Retranslate to Application



Criteria and/or Range Criteria:



Options:



TRANSLATE TO APPLICATION Command Examples

Example 1: Translate all EDI documents in the Transaction Store that were received on December 14, 1997 with network ID IINB1. Put the data in the application file specified by the data format or the override file specified in the receive usage mapping for the trading partner.

```
PERFORM TRANSLATE TO APPLICATION
WHERE TRXDATE(97/12/14) NETID(IINB1)
```

Example 2: Translate all EDI documents in the Transaction Store that were received between December 14, 1997 and today's date. Put the data in the application file specified by the data format or the override file specified in the receive usage mapping for the trading partner.

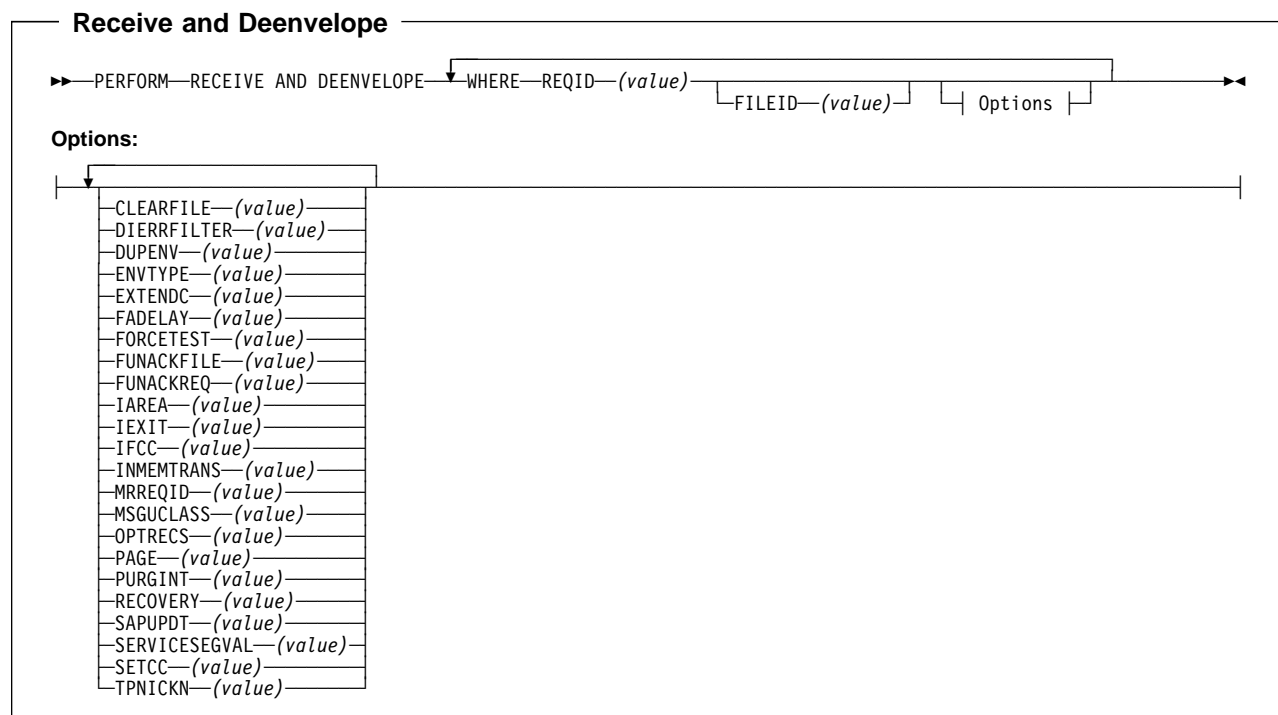
PERFORM TRANSLATE TO APPLICATION
WHERE TRXDATE(97/12/14) TO (*)

Example 3: Translate all EDI documents in the Transaction Store that were received on December 14, 1997 with network ID IINB1. Also translate data received on December 14, 1997 from trading partner PISCES. Put the data in the application file specified by the data format or the override file specified in the receive usage mapping for the trading partner.

```
PERFORM TRANSLATE TO APPLICATION
WHERE TRXDATE(97/12/14) NETID(IINB1)
WHERE TRXDATE(97/12/14) TPNICKN(PISCES)
```

Receiving and Deenveloping EDI Data

The RECEIVE AND DEENVELOPE command combines the services of the RECEIVE and DEENVELOPE commands. It receives EDI data from the network indicated in the requestor profile member. It puts the data in the receive file specified by the requestor profile member or the override file specified by the command. It then removes the envelope segments and puts the results in the Transaction Store.



RECEIVE AND DEENVELOPE Command Examples

Example 1: Receive EDI documents for requestor DEPTA7F, deenvelope them, and put them in the Transaction Store. The documents are in X12 format.

```

PERFORM RECEIVE AND DEENVELOPE
WHERE REQID(DEPTA7F)
  
```

Example 2: Receive EDI documents for requestor DEPTA7F, deenvelope them, and put them in the Transaction Store. Set the purge interval for these documents at 20 days. The documents are in EDIFACT format.

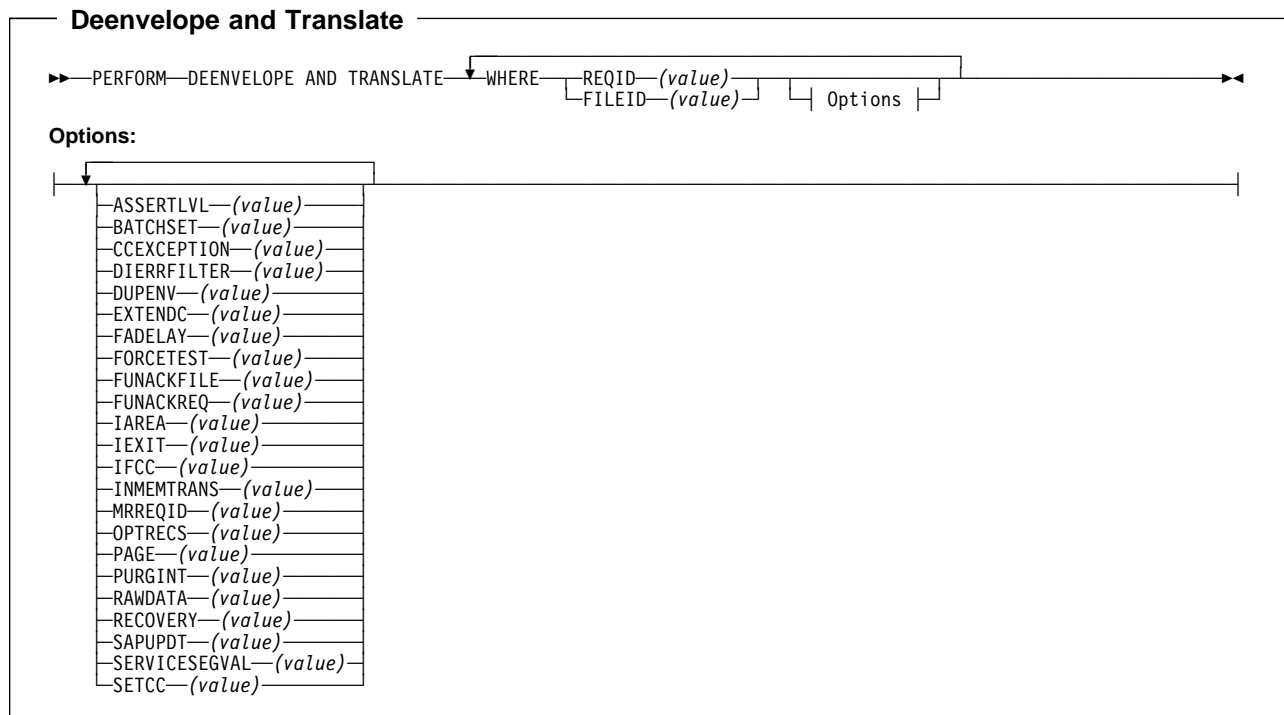
```

PERFORM RECEIVE AND DEENVELOPE
WHERE REQID(DEPTA7F) PURGINT(20) ENVTYPE(E)
  
```

Deenveloping and Translating EDI Data

The DEENVELOPE AND TRANSLATE command combines the services of the DEENVELOPE and TRANSLATE TO APPLICATION commands. It takes EDI documents from the receive file specified in the requestor profile member, or an override file specified in the command, removes the envelope segments, and puts the results in the Transaction Store. It then translates the documents to an application format and puts the results in the application file specified by the data format or its override in the trading partner usage. This command performs faster than the separate commands.

Note: In CICS, you can also deliver the data to a program of CICS transaction, as specified in the data format fields **Application file name** and **Application file type**.



DEENVELOPE AND TRANSLATE Command Examples

Example 1: Deenvelope and translate EDI documents in the receive file specified by requestor profile member DEPTA7F.

```

PERFORM DEENVELOPE AND TRANSLATE
WHERE REQID(DEPTA7F)

```

Example 2: Deenvelope and translate EDI documents in the receive file specified by requestor profile member DEPTA7F. Provide the translated documents in raw data format and return the information records to the exception file.

```

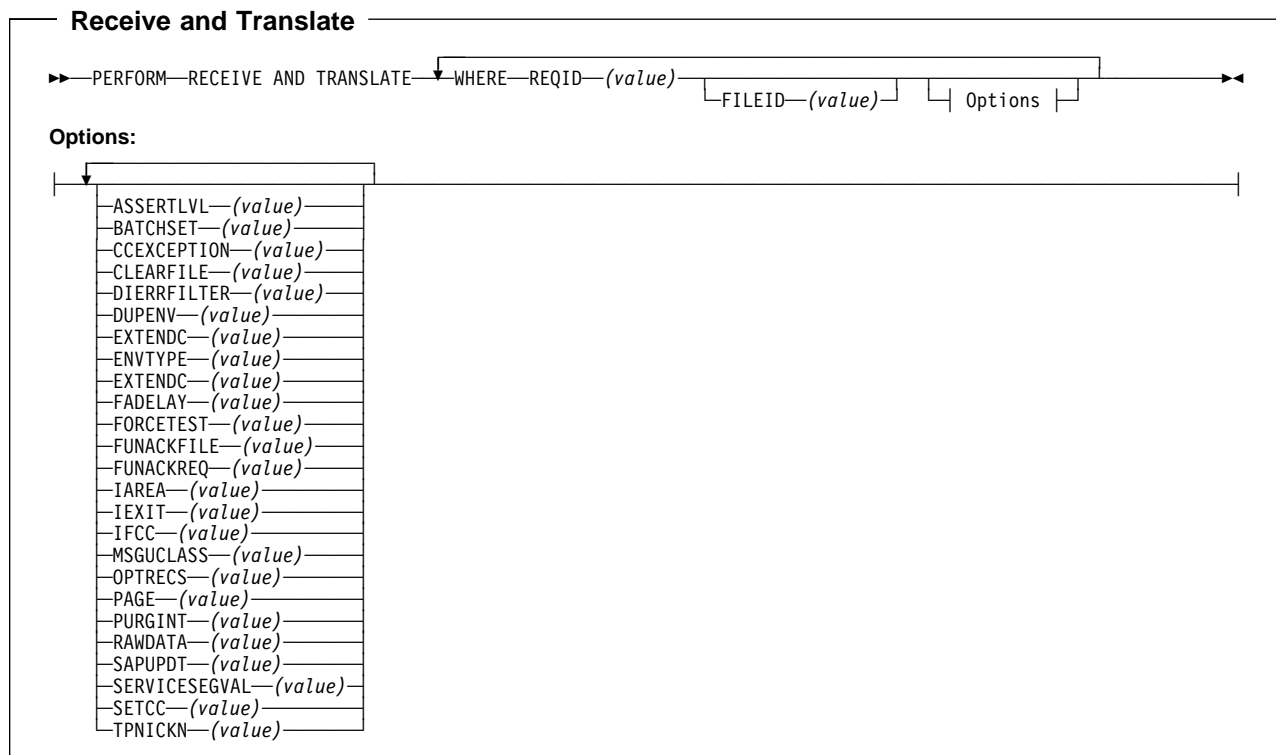
PERFORM DEENVELOPE AND TRANSLATE
WHERE REQID(DEPTA7F) RAWDATA(Y) OPTRECS(I)

```

Receiving and Translating EDI Data

The RECEIVE AND TRANSLATE command combines into one step the services of the RECEIVE, DEENVELOPE, and TRANSLATE TO APPLICATION commands. This command performs faster than the separate commands.

Note: In CICS, you can also deliver the data to a program of CICS transaction, as specified in the data format fields **Application file name** and **Application file type**.



RECEIVE AND TRANSLATE Command Examples

Example 1: Receive and translate the EDI documents for requestor DEPTA7F. Along with the transaction data, return all optional records produced during deenveloping and translation. The documents are in X12 format.

```

PERFORM RECEIVE AND TRANSLATE
WHERE REQID(DEPTA7F) OPTRECS(IEGTQ)

```

Example 2: Receive and translate the EDI documents for requestor DEPTA7F. Return the transaction data in raw format. The documents are in EDIFACT format.

```

PERFORM RECEIVE AND TRANSLATE
WHERE REQID(DEPTA7F) RAWDATA(Y) ENVTYPE(E)

```

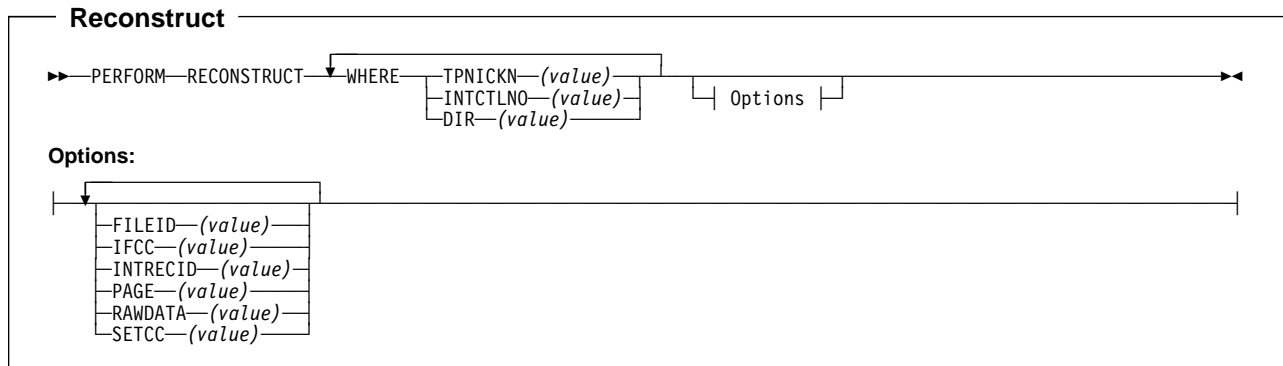
Managing Data

The following set of utility commands serves these functions:

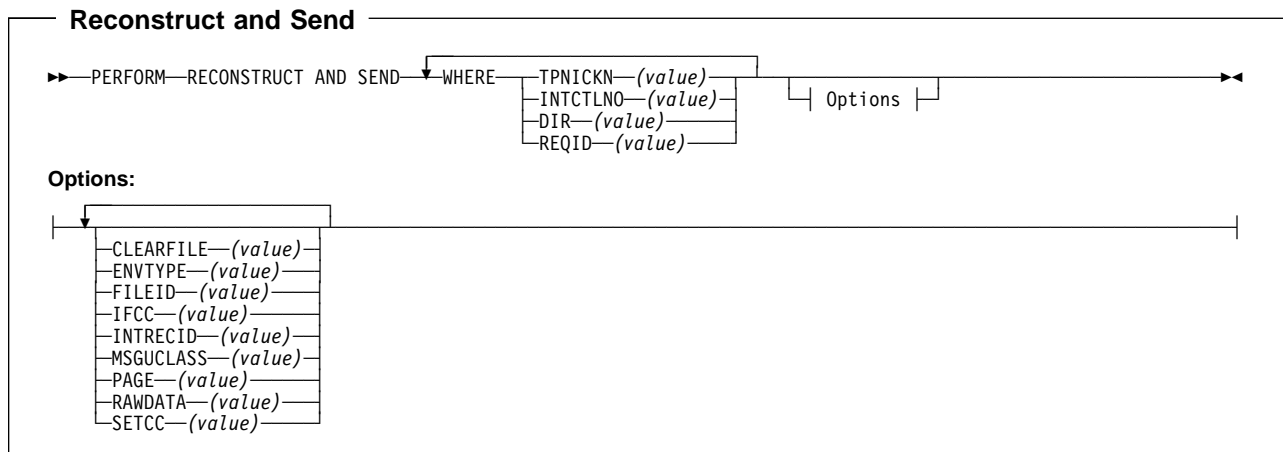
- Rebuilding interchanges from transactions within the Transaction Store.
- Updating status of transactions within the Transaction Store.
- Removing transactions from the Transaction Store.
- Updating management reporting statistics.
- Removing management reporting statistics.

Reconstructing Envelopes

The RECONSTRUCT command takes information that has been saved within the Transaction Store and reconstructs or rebuilds an interchange just as it was sent or received (using the same control numbers). This can be used if your trading partner lost an interchange you sent and you need to send the same interchange again. It could also be used to rebuild interchanges sent to you.



The RECONSTRUCT AND SEND command rebuilds interchanges from data in the Transaction Store and also SENDS those interchanges to the network.



RECONSTRUCT/RECONSTRUCT AND SEND Command Examples

Example 1: You received and processed interchange control number 5 from your MYTP trading partner but have since destroyed the sequential file containing the interchange. You would like to reconstruct this file so you can save it for the auditors.

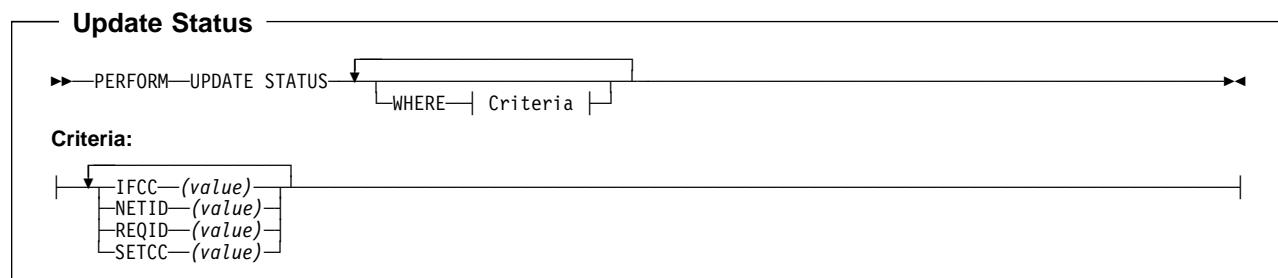
```
PERFORM RECONSTRUCT WHERE TPNICKN(MYTP)
INTRECID(123456789) INTCTLNO(5) DIR(R)
FILEID(AUDITOR)
```

Example 2: Your trading partner (called MYTP) called and indicated that interchange control number 5 is missing and would like for you to resend that interchange. The DUNS number for MYTP is 123456789 and the DUNS number is used as the interchange receiver ID value.

```
PERFORM RECONSTRUCT AND SEND WHERE TPNICKN(MYTP)
INTRECID(123456789) INTCTLNO(5) DIR(S) REQID(MYREQ)
```

Update Network Status

The UPDATE STATUS command retrieves network acknowledgments for all requestor profile members, explicitly supplied members, or explicitly supplied network profile members. It processes all network acknowledgments that resulted from previous SEND commands, pairs them with the transactions originally sent, and updates transaction status.



| The WHERE clause is optional for this command. If one is not supplied, all network profile members are processed.

You request network acknowledgments in the **Net acknowledgment** field of the trading partner or requestor profile.

UPDATE STATUS Command Examples

Example 1: Receive and process all network acknowledgments for all defined requestor ID profile members.

```
PERFORM UPDATE STATUS
```

Example 2: Receive and process network acknowledgments for all defined requestor ID profile members with the associated network profile of IINB41.

```
PERFORM UPDATE STATUS  
WHERE NETID(IINB41)
```

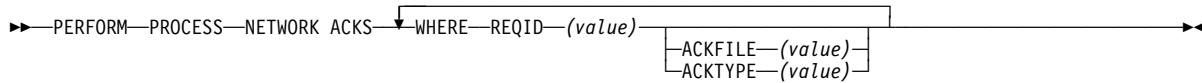
Example 3: Receive and process network acknowledgments for two different requestor ID profile members, ROBOX and KANBOX.

```
PERFORM UPDATE STATUS  
WHERE REQID(ROBOX)  
WHERE REQID(KANBOX)
```

Processing Received Network Acknowledgments

The UPDATE STATUS command issues network receive commands to gather acknowledgments and then updates transaction status based on the acknowledgments. The PROCESS NETWORK ACKS command is different in that it processes acknowledgments that have already been received and are located in a file. This command is used internally by DataInterchange for CICS when network acknowledgments are processed on a continuous receive basis. This command might be useful if network acknowledgments are received outside DataInterchange control but still need to be applied to the Transaction Store.

Process Network Acks



PROCESS NETWORK ACKS Command Example

Example: Process a file allocated to *ddname* PROCACKS containing network acknowledgments associated with requestor ID IINB41REQ.

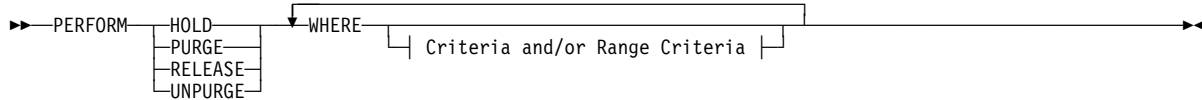
```

PERFORM PROCESS NETWORK ACKS
WHERE REQID(IINB41REQ) ACKFILE(PROCACKS)
  
```

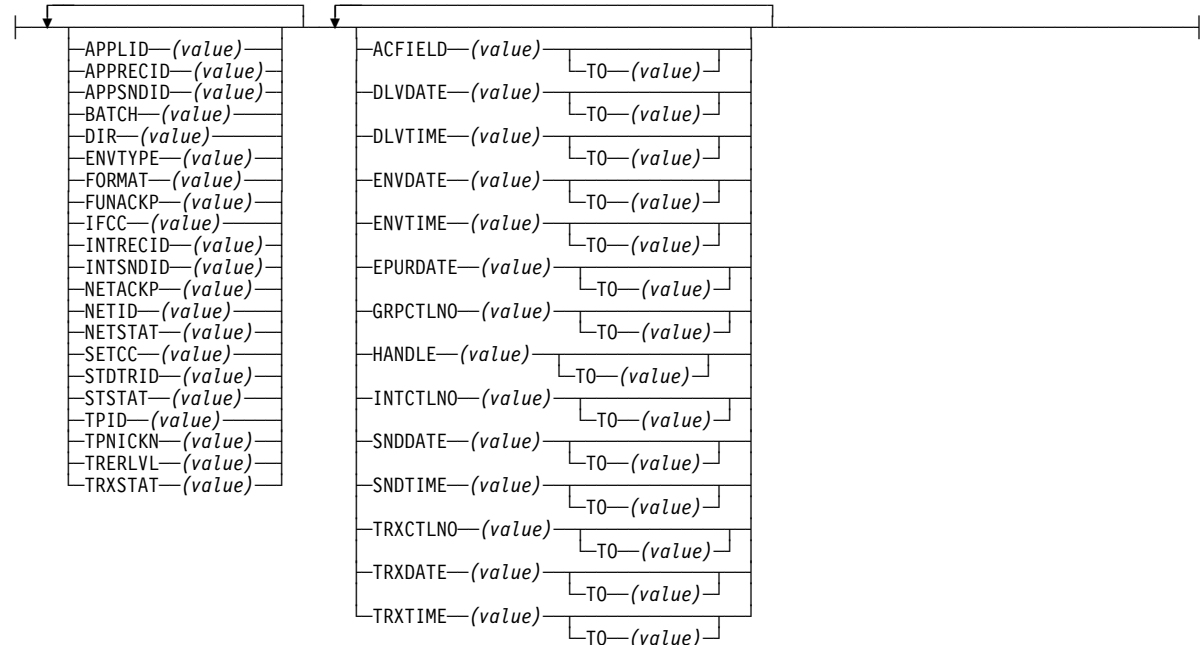
Hold, Purge, Release, and Unpurge

The HOLD, PURGE, RELEASE, and UNPURGE commands have a common syntax, which is illustrated below.

Hold, Purge, Release, and Unpurge



Criteria and/or Range Criteria:



Hold Transactions

The HOLD command puts a transaction in held status. While in held status, no actions are permitted that would change the status of the transaction. Nor is it purged automatically when its store time expires. If the transaction is one of a related group, all transactions in the group are placed in held status. The RELEASE command restores held transactions to their former status (or to store-time-expired status if their store time expired during the hold period).

Example: Place in hold status all EDI documents destined for trading partner PISCES that were translated on December 14, 1997.

```
PERFORM HOLD
WHERE TPNICKN(PISCES) TRXDATE(97/12/14) DIR(S)
```

Mark Transaction for Purge

| The PURGE command marks a transaction for purging from the Transaction Store. Transactions are marked for purging when old outbound transactions are still pending functional reconciliation. The transaction is deleted from the store when you execute the REMOVE TRANSACTIONS command. For more information, see “Purging Documents From the Transaction Store” on page 1-32. If the transaction is one of a related group, all transactions in the group are marked for purging. Reconciled transactions are automatically marked PURGE-DATE EXPIRED after the purge interval has expired.

| The default expiration date is 30 days from translation. You can change the default by specifying a PURGINT value on PERFORM statements for which transactions are added to the transaction store (for example, PERFORM TRANSLATION AND ENVELOPE). Refer to Page 1-100 in “Keyword, Option, Criteria, and Range Criteria Descriptions” for applicable PERFORM commands.

| **Example:** Delete eligible transactions that are more than 30 days old.

```
| PERFORM REMOVE TRANSACTIONS
| WHERE HANDLE(*-9999) TO(*-30)
```

| Only transactions in PURGE-USER REQUESTED or PURGE-DATE EXPIRED are eligible.

| **Example:** Mark for purging all EDI documents that have been delivered to trading partner PISCES and accepted.

```
| PERFORM PURGE
| WHERE TPNICKN(PISCES) TRXSTAT(61)
```

Release Transactions

| The RELEASE command restores a transaction in held status to its former status (or to purge-pending status if its store time expired during the hold period). If the transaction is one of a related group, all transactions in the group are released. DataInterchange never automatically sets a transaction's store status to HELD; this is done only through the HOLD operation.

| **Example:** Release all EDI documents destined for trading partner PISCES that are in HELD status.

```
| PERFORM RELEASE
| WHERE TPNICKN(PISCES) DIR(S) STSTAT(1)
```

Restore Transaction Status

| The UNPURGE command restores a transaction to the status it had before it was marked for purging. Until you execute the REMOVE TRANSACTIONS command, you can restore transactions marked for purging by the PURGE command. After the REMOVE TRANSACTIONS command runs, you cannot restore them. If the transaction is one of a related group, all transactions in the group are restored to their former status. If a transaction's store time has expired, it can still be eligible for purging. UNPURGE does not change the purge status of transactions whose store time has expired.

| **Example:** Restore all EDI documents with application control numbers PO112233 through PO112244 that are marked for purging by user request.

```
| PERFORM UNPURGE  
| WHERE ACFIELD(P0112233) TO(P0112244) STSTAT(4)
```

| **Purging Documents From the Transaction Store**

| The REMOVE TRANSACTIONS command deletes from the Transaction Store documents with a store status of either of the following:

- | • Purge-store time expired
- | • Purge-user request

| You can protect documents you want to keep with the HOLD or UNPURGE command. UNPURGE, however, protects only the documents whose status is purge-user request. It does not protect those whose status is purge-store time expired. Once purged, the documents are not recoverable.

| The default expiration date is 30 days from translation. You can change the default by specifying a PURGINT value on PERFORM statements for which transactions are added to the transaction store (for example, PERFORM TRANSLATION AND ENVELOPE). Refer to “Keyword, Option, Criteria, and Range Criteria Descriptions” on page 1-69 for applicable PERFORM commands.

Note: Running PERFORM REMOVE TRANSACTIONS with selection criteria specified in the WHERE clause may remove entries from the EDIVTSTH table that do not match the selection criteria and report those handle numbers as being removed in the audit trail. These handles that are removed not matching the criteria have no direction assigned to them and are over 10 days old. These orphan handles are created if the first transaction of a translation process does not complete normally. These handles cannot be accessed in any way and only show up when removed. They are validly removed any time the REMOVE program is executed.

| **Example:** Delete eligible transactions that are more than 30 days old.

```
| PERFORM REMOVE TRANSACTIONS  
| WHERE HANDLE(*-9999) TO(*-30)
```

| **Note:** Refer to “DataInterchange Optimization” on page G-1 for more information.

| Only transactions in PURGE-USER REQUESTED or PURGE-DATE EXPIRED are eligible.

PERFORM—REMOVE TRANSACTIONS

WHERE

Criteria and/or Range Criteria

Options

Criteria and/or Range Criteria:

APPLID—(value)

APPRECID—(value)

APPSNDID—(value)

BATCH—(value)

DIR—(value)

ENVTYPE—(value)

FORMAT—(value)

FUNACKP—(value)

INTRECID—(value)

INTSNDID—(value)

NETACKP—(value)

NETID—(value)

NETSTAT—(value)

STDTRID—(value)

STSTAT—(value)

TPID—(value)

TPNICKN—(value)

TRERLVL—(value)

TRXSTAT—(value)

ACFIELD—(value)

DLVDATE—(value)

DLVTIME—(value)

ENVDATE—(value)

ENVTIME—(value)

EPURDATE—(value)

GRPCTLNO—(value)

HANDLE—(value)

INTCTLNO—(value)

SNDDATE—(value)

SNDDTIME—(value)

TRXCTLNO—(value)

TRXDATE—(value)

TRXTIME—(value)

Options:

IFCC—(value)

MAXRUNTIME—(value)

NUMDELS—(value)

SETCC—(value)

STANDALONE—(value)

The WHERE clause is optional for this command. If one is not supplied, all eligible transactions are removed from the Transaction Store.

After purging documents from the Transaction Store, you can use the Archive action to remove event logs associated with these documents. For more information, see the *DataInterchange Administrator's Guide*.

REMOVE does not delete ACTIVE or HELD transactions.

- To make ACTIVE transactions eligible for removal, they must first be marked for purge.
- To make HELD transactions eligible for removal, they must first be released.

Removing and Archiving Event Log Entries

Using the DataInterchange Utility, you can remove event log entries and optionally archive them. Typically, this is a multi-step process (see Table 1-3 on page 1-34). Event log entries are removed real-time in a DB2 environment; however, reclaiming and reorganizing the table space requires an additional step.

In the VSAM environment, event logs are entry sequenced data sets (ESDS). Therefore, entries cannot be removed real-time; a three-step batch process must be followed.

Table 1-3. Various Ways to Remove and Archive Event Log Entries

Function	Step One	Step Two	Step Three
Remove DB2 log entries with no archive and no DB2 reorg	Run DataInterchange Utility: PERFORM REMOVE LOG ENTRIES WHERE APPLID (value)...		
Remove DB2 log entries with no archive and with DB2 reorg	Run DataInterchange Utility: PERFORM REMOVE LOG ENTRIES WHERE APPLID (value)...	Run DBS Utility: REORG TABLESPACE EDIELOG	
Remove DB2 log entries with archive and with no DB2 reorg	Run DataInterchange Utility: PERFORM UNLOAD LOG ENTRIES WHERE APPLID (value) ARCHIVEFILE (value) HOLDFILE (value)...		
Remove DB2 log entries with archive and with DB2 reorg	Run DataInterchange Utility: PERFORM UNLOAD LOG ENTRIES WHERE APPLID (value) ARCHIVEFILE (value) HOLDFILE (value)...	Run DB2 Utility: REORG TABLESPACE EDIELOG	
Remove VSAM log entries with archive	Run DataInterchange Utility: PERFORM UNLOAD LOG ENTRIES WHERE APPLID (value) LOGFILE (value) ARCHIVEFILE (value) HOLDFILE (value)...	Run IDCAMS to delete and define the LOGFILE dataset	Run DataInterchange Utility: PERFORM LOAD LOG ENTRIES WHERE APPLID (value) LOGFILE (value) HOLDFILE (value)...

Remove Log Entries

The PERFORM REMOVE LOG ENTRIES command is valid only for DB2 event logs. It can be a very quick way to remove entries. If the ARCHIVEFILE keyword is used on the command, it works the same as the UNLOAD LOG ENTRIES command.

If the NUMDELS keyword is used, rows are deleted sequentially with COMMITs done every NUMDELS number of rows. If the NUMDELS keyword is omitted, a single fully-qualified SQL DELETE statement is issued. (This can be very efficient, but can also quickly exhaust DB2 resources. Omit NUMDELS with caution.)

To ensure concurrency, use NUMDELS with a relatively low value. The REMOVE LOG ENTRIES command may be run as step one of a multi-step process (see Table 1-3).

Remove Log Entries

►►—PERFORM—REMOVE LOG ENTRIES—►WHERE—APPLID—(value) | Range Criteria | Options | ►►

Range Criteria:

LOGDATE—(value)	TO—(value)
LOGTIME—(value)	TO—(value)
LOGUSER—(value)	TO—(value)
LOGFORM—(value)	TO—(value)
LOGAEID—(value)	TO—(value)

Options:

ARCHIVEFILE—(value)
ARCHIVETYPE—(value)
HOLDFILE—(value)
HOLDTYPE—(value)
IFCC—(value)
NUMDELS—(value)
SETCC—(value)

Unload Log Entries

The PERFORM UNLOAD LOG ENTRIES command reads all the entries in the event log associated with the APPLID. The entries selected for removal are copied to the ARCHIVEFILE, and all other entries associated with the APPLID are copied to the HOLDFILE.

Event log entries that are associated with active or held transactions in the Transaction Store are not eligible for archive and are not selected for removal.

In DB2, the entries selected for removal are not only copied to the ARCHIVEFILE, but are also immediately deleted. In VSAM, the entries selected for removal are only copied to the ARCHIVEFILE, and are not actually deleted.

To ensure concurrency in DB2, use NUMDELS with a relatively low value. The UNLOAD LOG ENTRIES command may be run as step one of a multi-step process (see Table 1-3 on page 1-34).

Unload Log Entries in a DB2 Environment

```

PERFORM UNLOAD LOG ENTRIES WHERE APPLID—(value)—ARCHIVEFILE—(value)—HOLDFILE—(value)

```

Range Criteria Options

Range Criteria:

```

LOGDATE—(value) TO—(value)
LOGTIME—(value) TO—(value)
LOGUSER—(value) TO—(value)
LOGFORM—(value) TO—(value)
LOGAEID—(value) TO—(value)

```

Options:

```

ARCHIVETYPE—(value)
HOLDTYPE—(value)
IFCC—(value)
NUMDELS—(value)
SETCC—(value)

```

Unload Log Entries in a VSAM Environment

```

PERFORM UNLOAD LOG ENTRIES
WHERE APPLID—(value)—LOGFILE—(value)—ARCHIVEFILE—(value)—HOLDFILE—(value)

```

Range Criteria Options

Range Criteria:

```

LOGDATE—(value) TO—(value)
LOGTIME—(value) TO—(value)
LOGUSER—(value) TO—(value)
LOGFORM—(value) TO—(value)
LOGAEID—(value) TO—(value)

```

Options:

```

ARCHIVETYPE—(value)
HOLDTYPE—(value)
IFCC—(value)
SETCC—(value)

```

Load Log Entries

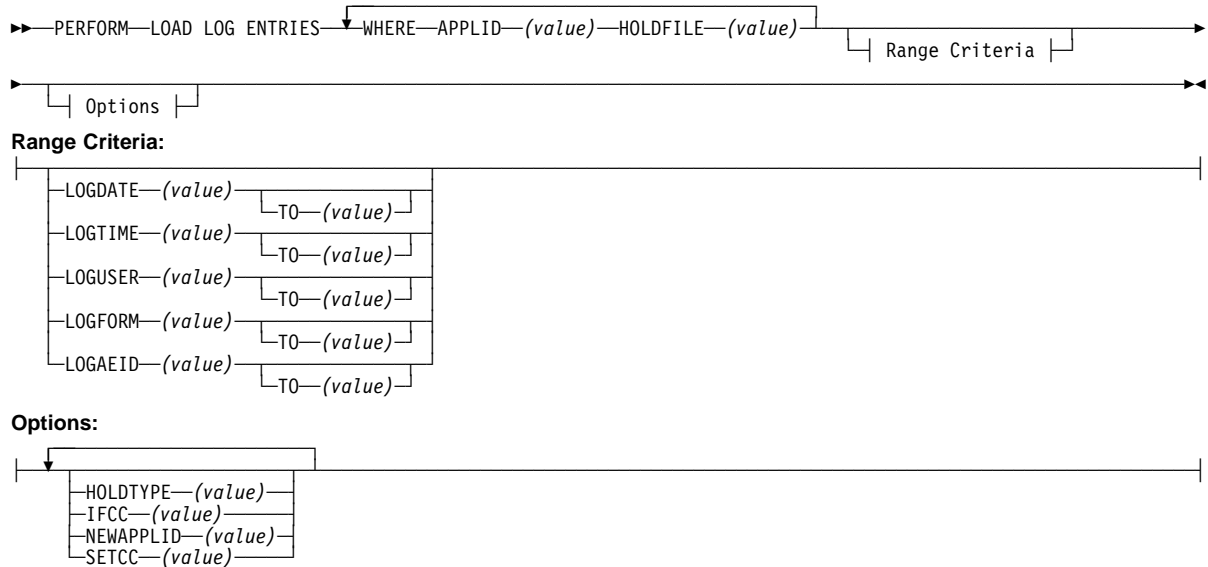
The PERFORM LOAD LOG ENTRIES command copies the selected HOLDFILE records back into the event log table (in DB2) or back into the LOGFILE (in VSAM).

In DB2, the LOAD LOG ENTRIES command can be run to restore deleted records back into the event log. To do this, specify the ARCHIVEFILE value (from the UNLOAD LOG ENTRIES command) with the HOLDFILE keyword.

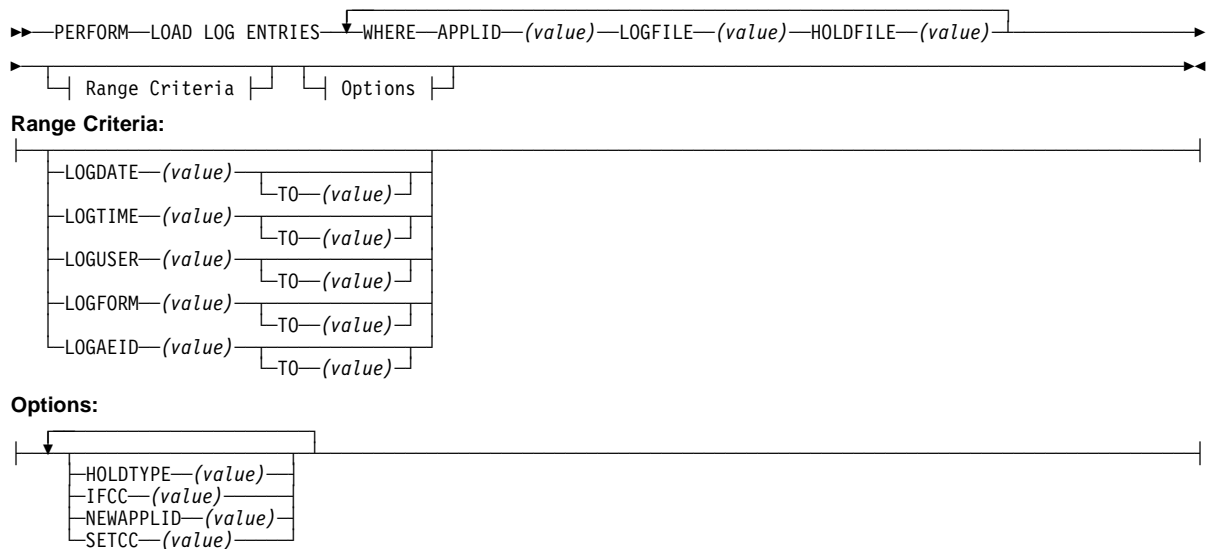
In VSAM, the LOAD LOG ENTRIES command must be run as the final step of a three-step process (see Table 1-3 on page 1-34). Step one copies the non-archived event log entries to the HOLDFILE. Step two deletes and defines the event log. Step three reloads the empty event log with the records from the HOLDFILE.

Note: The VSAM event logs are entry sequenced data sets (ESDS). Therefore, caution must be observed when loading them; running the same LOAD LOG ENTRIES command multiple times loads the same records multiple times.

Load Log Entries in a DB2 Environment



Load Log Entries in a VSAM Environment



Updating Management Reporting Statistics

The UPDATE STATISTICS command allows you to update the management reporting statistics tables.

Update Statistics

►►—PERFORM—UPDATE STATISTICS—
└─WHERE NUMUPDTS—┘

UPDATE STATISTICS Command Example

Example: Update the statistics tables before you run a management reporting data extract. In addition, suppose you use the DB2 version of DataInterchange. Because of application requirements, the system is running multiple simultaneous translation jobs and it is heavily loaded. In addition, the Database Administrator (DBA) has set the DB2 timeout value rather low. As a result, you are encountering DB2 timeouts. To reduce the amount of time that the Update Statistics job holds table locks, set the NUMUPDTS value to a low number like 25. This causes the Update Statistics job to commit work and release all locks after every 25 DB2 updates.

```
PERFORM UPDATE STATISTICS  
WHERE NUMUPDTS(25)
```

Removing Management Reporting Statistics

The REMOVE STATISTICS command allows you to delete outdated management reporting statistics.

Remove Statistics

►►—PERFORM—REMOVE STATISTICS—
└─WHERE— Options ┘

Options:

└─IFCC—(value)—
└─PRIORTO—(value)—
└─NUMDELS—(value)—
└─SETCC—(value)—

REMOVE STATISTICS Command Example

Remove any statistics over six months old. In addition, suppose you are a user of the DB2 version of DataInterchange and have exceeded the maximum number of page locks on a previous Remove Statistics job. To get rid of statistics over six months old, use the PRIORTO keyword with a date of six months ago. To reduce the accumulation of page locks by the Remove Statistics process, you could set the NUMDELS parameter to a low number, for example 25. This causes the Remove Statistics process to commit work and release page locks after every 25 deletes.

```
PERFORM REMOVE STATISTICS  
WHERE PRIORTO(*-180) NUMDELS(25)
```

Note: The RESET STATISTICS command allows you to reset the Management Reporting cumulative statistics counts to zero.

Reporting and Data Extraction

The following set of utility commands serves these functions:

- Generate formatted reports containing data from the Transaction Store.
- Format application data into conventional business documents.
- Extract data from the Transaction Store to be further processed outside DataInterchange.
- Extract data from the management reporting statistics tables to be further processed outside DataInterchange.

Printing Transaction Store Reports

Each command described in this section uses your selection criteria to extract information from the Transaction Store. It then writes a formatted report to a file with the *ddname* RPTFILE. The same type of information can be obtained without report formatting by using the ENVELOPE and TRANSACTION DATA EXTRACT commands. For more information, see "Reporting Using Management Reporting and Transaction Store" on page 1-48. You can print the information using the system facilities. A cover page for each report lists the selection criteria you entered.

You can view most of the information in these reports using the Transaction Store Facility.

Print

PERFORM-PRINT

ACKNOWLEDGMENT IMAGE

ACTIVITY SUMMARY

EVENT LOG

STATUS SUMMARY

STATUS SUMMARY2

TRANSACTION DETAILS

TRANSACTION IMAGE

WHERE

Criteria and/or Range Criteria

Options

Criteria and/or Range Criteria:

APPLID—(value)

APPRECID—(value)

APPSNDID—(value)

BATCH—(value)

DIR—(value)

ENVTYPE—(value)

FORMAT—(value)

FUNACKP—(value)

INTRECID—(value)

INTSNDID—(value)

NETACKP—(value)

NETID—(value)

NETSTAT—(value)

STDTRID—(value)

STSTAT—(value)

TPID—(value)

TPNICKN—(value)

TRERLVL—(value)

TRXSTAT—(value)

ACFIELD—(value)

DLVDATE—(value)

DLVTIME—(value)

ENVDATE—(value)

ENVTIME—(value)

EPURDATE—(value)

GRPCTLNO—(value)

HANDLE—(value)

INTCTLNO—(value)

SNDDATE—(value)

SNDDTIME—(value)

TRXCTLNO—(value)

TRXDATE—(value)

TRXTIME—(value)

Options:

IFCC—(value)

MERGED—(value)

SEGMENTED—(value)

SETCC—(value)

PRINT ACTIVITY SUMMARY Command Example

The PRINT ACTIVITY SUMMARY command creates a summary of activity for inbound and outbound transactions that match your selection criteria. It writes the summary to RPTFILE.

Example: Print the Transaction Activity Summary for all EDI documents entering the Transaction Store between November 29 and November 30, 1997.

```
PERFORM PRINT ACTIVITY SUMMARY
WHERE TRXDATE(97/11/29) TO(97/11/30)
```

Note: This example is only valid when the date mask in the language profile (LANGPROF) is &Y/&M/&D.

This is an example of the activity summary report.

TF11	Activity Summary Report	Date: 97/12/14 Time: 12:12:12
	Outbound Transactions	Count
	Selected transactions	1940
	Translation	1940
	Acceptably translated . . .	1930
	Unacceptably translated . .	10
	Enveloping	1000
	Enveloped	900
	Enveloping errors	100
	Send requests	900
	Sent	800
	Pending functional ack . .	10
	Pending network ack . . .	10
	Not sent	100
	Error on request to send:	73
	Network error	27
	Detached	0
	Inbound Transactions	Count
	Selected transactions	1000
	Acceptably translated . . .	800
	Unacceptably translated . .	10
	Not yet translated	190
	Detached	0

PRINT ACKNOWLEDGMENT IMAGE Command Example

The PRINT ACKNOWLEDGMENT IMAGE command writes an image of functional acknowledgments to RPTFILE for the transactions you selected. The image does not include the envelope segments.

Example: Print functional acknowledgment images for all EDI documents with the batch ID 121497.

```
PERFORM PRINT ACKNOWLEDGMENT IMAGE
WHERE BATCH(121497)
```

This is an example of the functional acknowledgment image report.

TF13	Functional Acknowledgment Image	Date: 97/12/14 Time: 12:12:12
Trading partner nickname: PISCES		
Transaction handle . . . : 19971214101533000001		
Transaction status . . . : Transaction accepted		
AK1*IN*40088!AK2*810*000048118!AK5*A!AK9*A*1*1*1!		

PRINT TRANSACTION DETAILS Command Example

The PRINT TRANSACTION DETAILS command extracts detailed information about the transactions you select and writes the information to RPTFILE.

Example: Print transaction details for all EDI documents that were created using the Interactive Entry Facility and put in the Transaction Store on December 14, 1997.

```
PERFORM PRINT TRANSACTION DETAILS
WHERE APPL(EDIMP) TRXDATE(97/12/14)
```

This is an example of the transaction details report. For more information on the fields, see the *DataInterchange Administrator's Guide*.

TF73	Transaction Details	Date: 97/12/14 Time: 12:12:12
Transaction handle : 19971214101533000001		
Trading partner nickname : PISCES		
Internal trading partner ID : 12345678901234567890123456789012345		
Direction : SEND		
Data format ID : POSEND		
Application control number : 12345678901234567890123456789012345		
Interchange control number : 12345678901234		
Group control number : 12345678901234		
Transaction control number : 12345678901234		
Transaction status : Translated		
Added to store : 97/12/14-10:29:48		
Store status : Active		
Delivered to application :		
Translation error level : 1		
Translation : Acceptable		
Network acknowledgment requested . . . : D		
Network status :		
Functional acknowledgment :		
Functional ack date :		
Application ID : EDIMP		
Batch ID : BATCH123		
Earliest purge date : 97/12/14		
Earliest envelope date : 97/12/14		
Enveloped/deenveloped :		
Sent :		
Envelope profile member : X12V2R2		
Standard ID : X12V2R2		
Standard version : V2		
Standard level : R2		
Standard transaction ID : 850		
Network ID : IN		
Usage indicator : P		
Total segment count : 22		
Transaction size (bytes) : 742		

PRINT TRANSACTION IMAGE Command Example

The PRINT TRANSACTION IMAGE command writes an image of the transactions you select to RPTFILE. The images do not include envelope or security segments.

Example: Print transaction images for all EDI documents with receive translation errors.

```
PERFORM PRINT TRANSACTION IMAGE
WHERE TRXSTAT(73)
```

This is an example of the transaction image report.

TF75	Transaction Image	Date: 97/12/14 Time: 12:12:12
Trading partner nickname: PISCES		
Transaction handle . . : 19971214101533000001		
Transaction status . . : Receive trans error		
SEG1*DATA:DATAN**dANTA:dANTAN:-99:-99!SEG2*123456:-1.00:-100*12.3456:-1.00:900920:123000:-10000!SEG3*123.456:99:891128:083 0:990*1234.56:-1:900920:1259:-1*1234.56:-1.0000:900920:125900:-100!SEG4*15:1.00:891128:1259:1000*15:-1.0:891128:1259:99*C1 C2:D1D2D3D4D5:1.891128:1259:1000!SEG5*891128*1259!TOTALS*210.000000*96.000000*328.000000*196.000000*32.000000*8.000000!		

PRINT STATUS SUMMARY Command Example

The PRINT STATUS SUMMARY command extracts status information about the transactions you select and writes the information to RPTFILE. This report shows transaction status (such as translated, enveloped), network status (such as delivered, purged), and store status (active, held, marked for purging). An R after the transaction handle indicates the transaction is a related transaction and is part of a bundle.

The status summary is not available for online viewing.

Example: Print the status summary for all EDI documents that were translated on December 14, 1997.

```
PERFORM PRINT STATUS SUMMARY
WHERE TRXDATE(97/12/14)
```

This is an example of the outbound and inbound transaction report. For more information on these fields, see the *DataInterchange Administrator's Guide*.

TF80		Status Summary Report for Outbound Transactions			Date: 97/12/14 Time: 12:12:12
Transaction Handle Date Enveloped	Trading Partner Nickname Interchange Cntrl No	Data Format ID Network Status	Transaction Status Group Control No	Store Status Func Ack Status	
19931110093012000000 97/12/14	Partner111111111 12121212121212	Format111111111 Accepted by network	Transaction accepted 11111111111111	Purge Requested Received	
19931212104533000001	Partner111111111	Format333333333	Send translate error	Active	
19931110103311000000 97/12/14	Partner222222222 22222222222222	Format222222222 Accepted by network	Transaction accepted 11111111111111	Active Received	
19931110103311000004 R 97/12/14	Partner222222222 22222222222222	Format333333333 Accepted by network	Transaction accepted 22222222222222	Active Received	
19931110103311000003 97/12/14 97/12/14	Partner333333333 33333333333333 33333333333333	Format111111111 Recall request error Not sent - net error	Transaction accepted 11111111111111 22222222222222	Purge Requested Received	
TF80		Status Summary Report for Inbound Transactions			Date: 97/12/14 Time: 12:12:12
Transaction Handle Date Translated	Trading Partner Nickname Data Format ID	Standard Trans ID Translation Status	Transaction Status	Store Status	
19931121134427000000 97/12/14	Partner222222222 Format1212121212	12121212 Acceptable	Receive translated	Active	
19931010101010000001 97/12/14 97/12/14	Partner222222222 Format3333333333 Format3333333333	12345678 Acceptable Unacceptable	Receive translated	Active	

PRINT STATUS SUMMARY2 Command Example

The PRINT STATUS SUMMARY2 command extracts the same information as the PRINT STATUS SUMMARY command and adds a line containing the application control number and the internal trading partner ID value for the transaction. An R after the transaction handle indicates the transaction is a related transaction and is part of a bundle.

Example: Print transaction images for all EDI documents with receive translation errors.

```
PERFORM PRINT STATUS SUMMARY2
WHERE TRXSTAT(73)
```

This is an example of the selection criteria for the status summary report.

1	Selection Criteria for Status Summary Report				Date: 97/12/14 Time: 14:25:44
SELECTION CRITERIA 1					
Transaction handle : 19930323000000000000 TO 19930323240000000000					
1TF80	Status Summary Report for Outbound Transactions				Date: 97/12/14 Time: 14:25:44
Transaction Handle	Trading Partner Nickname	Data Format ID	Transaction Status	Store Status	
Application Control Number	Internal Trading Partner ID				
Date Enveloped	Interchange Cntrl No	Network Status	Group Control No	Func Ack Status	
19930323092157900000	BOBS	BOBS8004833	ENVELOPED	ACTIVE	
		1111122222			
97/12/14	00000000000418	ENVELOPED	00000000000399	PENDING	
19971214111656200000	BOBS	BOBSPORDER	ENVELOPED	ACTIVE	
BOBS 12345678	BOBS				
97/12/14	00000000000419	ENVELOPED	00000000000400	NOT REQUESTED	

PRINT EVENT LOG Command Example

The PRINT EVENT LOG command writes an image of event log entries to RPTFILE for transactions that match your selection criteria.

Example: Print log entries for all EDI documents that encountered errors during send processing.

```
PERFORM PRINT EVENT LOG
WHERE TRXSTAT(42)
WHERE TRXSTAT(43)
```

[illegible]

The PRINT CUSTOM LAYOUT command prints customized documents. For more information, see the *DataInterchange Administrator's Guide*.

▶▶—PERFORM—PRINT CUSTOM LAYOUT—WHERE—APPFILE—(*value*)————▶▶

Example: Print the customized document in the file whose ddname is A7FIN.

```
PERFORM PRINT CUSTOM LAYOUT
WHERE APPFILE(A7FIN)
```

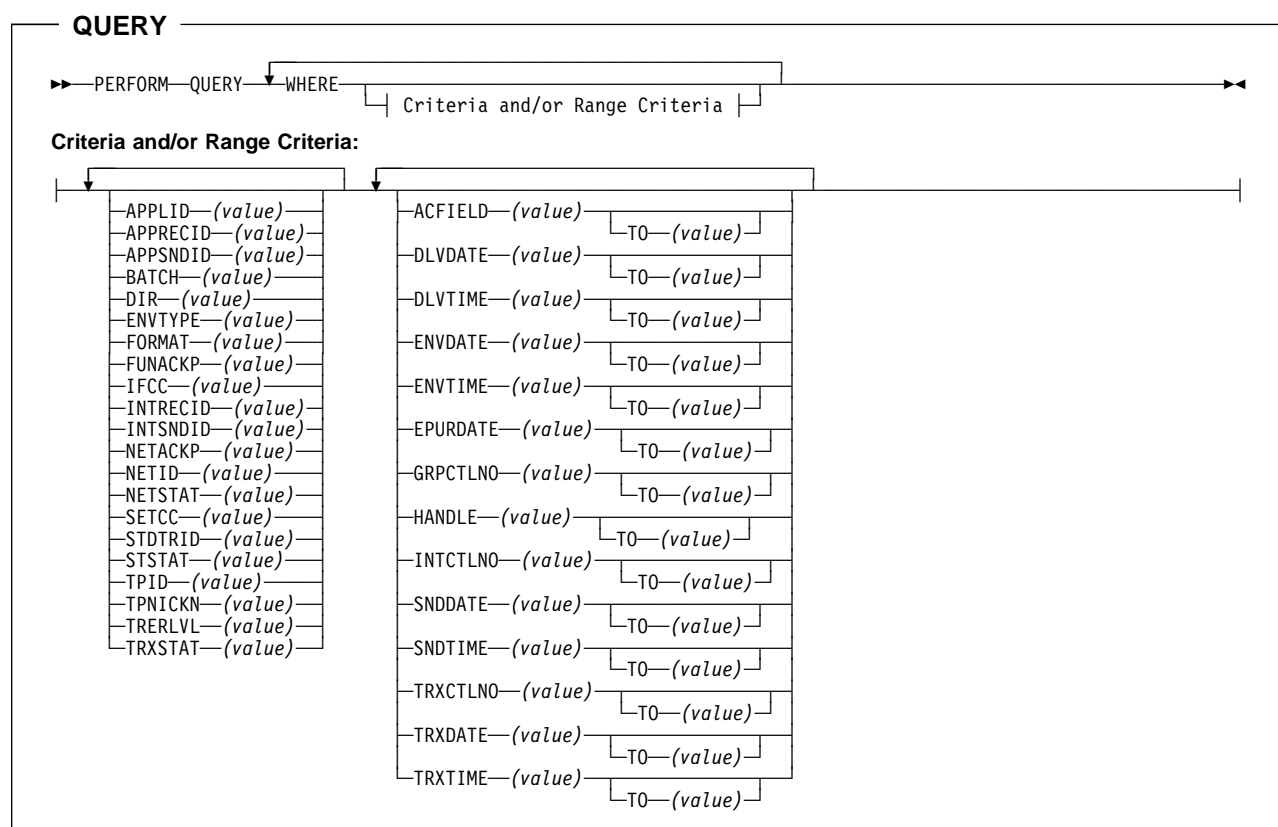
Query Transaction

The QUERY command obtains a list of transactions that meet the selection criteria you specify in the keywords. In the returned list, transactions are identified by their Transaction Store ID, called a transaction handle. The DataInterchange Utility writes the list of handles to a file whose ddname is EDIQUERY for MVS users. Each item in the list is 30 bytes long and has the following format:

Bytes 1-10: Handle in packed decimal format
Bytes 11-30: Handle in EBCDIC character format

You might want to use the QUERY command to provide input for a user-written reporting program.

Note: In CICS, you can pass the file name and type as parameters to the DataInterchange Utility.



QUERY Command Example

Return a list of transactions with batch ID 121497 whose translation error level is 2 (data element and segment errors):

```
PERFORM QUERY
WHERE BATCH(121497) TRERLVL(2)
```

Reporting Using Management Reporting and Transaction Store

DataInterchange provides two mechanisms for producing reports:

- Management Reporting Data Extracts
- Transaction Store Data Extracts

Both collect, update, and extract DataInterchange's trading partner data and transaction information. Both are invoked with PERFORM commands to extract the data, and both require user-written programs to sort, format, and print data. The difference between the Management Reporting and the Transaction Store Data Extracts is the nature of information each provides.

Management Reporting Data Extracts pull information from statistics tables, while Transaction Store Data Extracts pull information from the Transaction Store.

DataInterchange collects information in four management reporting categories and two Transaction Store reporting categories. They are:

- Management Reporting Categories
 1. Trading Partner Profile - provides general trading partner information alphabetically, by trading partner. Information includes company name, address, point of contact, telephone numbers, account ID, and user ID.
 2. Trading Partner Capability - relates trading partner names used. For example, this extract allows you to determine which maps are installed in test and production, which trading partners are using which maps, mapping direction, standards used, transaction ID, total number of transactions processed, and number of transactions that had errors.
 3. Transaction Activity - provides information relating transaction volumes to trading partner names. For example, this report can calculate the total number of transactions sent to a trading partner (by date and by map ID) or the total transactions errors by trading partner (and by date or map ID). This information is often used to gauge a trading partner's activity or to reconcile documents sent or received.
 4. Network Activity - provides information such as network ID, name and account number, user ID, direction, charge code, and total number of bytes sent or received. This information is typically used to determine network charges, by trading partner or application.
- Transaction Store Reporting Categories
 1. Transaction Data Extract - extracts detailed technical information from the Transaction Store. It provides data such as trading partner nickname, direction (S/R), transaction image, send acknowledgment data and image, receive acknowledgment data and image, and transaction handle. This information is commonly used for creating daily reports, such as overdue functional acknowledgments.
 2. Envelope Data Extract - extracts detailed technical information from the Transaction Store. It provides the same information as the transaction data extract except it only reports on transactions that have been enveloped, and it sorts the data differently. This information is commonly used for creating daily reports.

Steps for Creating Management Reporting Reports

The steps for using Management Reporting are:

1. Set the Management Reporting Active? flag to Y in the APPDEFs profile member. The default is Y.
2. Change any trading partner profiles you want to report to include selection information in the profile fields, such as a department name in a comment field. For more information on the management reporting examples, see “TRADING PARTNER PROFILE DATA EXTRACT Command Examples” on page 1-50.
3. Perform your usual daily activity of translations and communications. During these activities, DataInterchange collects information about these activities in a set of pending tables.
4. Execute the update statistics command to move the pending information to statistics tables. This step must be performed prior to data extraction.

| PERFORM UPDATE STATISTICS

For more information, see “Updating Management Reporting Statistics” on page 1-38.

5. Execute the data extract command:

| PERFORM DATA EXTRACT

| This set of PERFORM commands collects the requested data and places it in a QSAM file named EDIQUERY. Multiple WHERE clauses and wild cards (* and ?) are allowed within each PERFORM DATA EXTRACT command.

Wild Card	Description
(any character)	Matches any character
?	Matches any single character
*	Matches any sequence of zero or more characters

For more information, see the report examples that follow this topic heading.

6. Process the data extract file into reports by:
 - Verifying that you successfully created the QSAM data extract file (extract file is EDIQUERY). A sequential set of independent records with a record length of 1024. For more information, see “Management Reporting” on page 2-76.
 - Sorting the data according to your requirements through a sort utility.
 - Formatting the data using a report writing utility or a user written program.

7. Remove previously collected daily statistics by executing command:

| PERFORM REMOVE STATISTICS

Because statistics are not dependent on the Transaction Store, the perform remove transactions command will not affect them. For more information, see “Removing Management Reporting Statistics” on page 1-38.

Note: The USERPGM parameter can be used to pass the report records to a program prior to writing them to the EDIQUERY QSAM file. When using this parameter, your program should return a code indicating whether the record is to be written to the EDIQUERY file or discarded. If a program is not supplied, DataInterchange writes the record to the EDIQUERY file.

Below are some examples of reports generated by the management reporting and data extract facilities.

Management Reporting Trading Partner Profile Data Extract

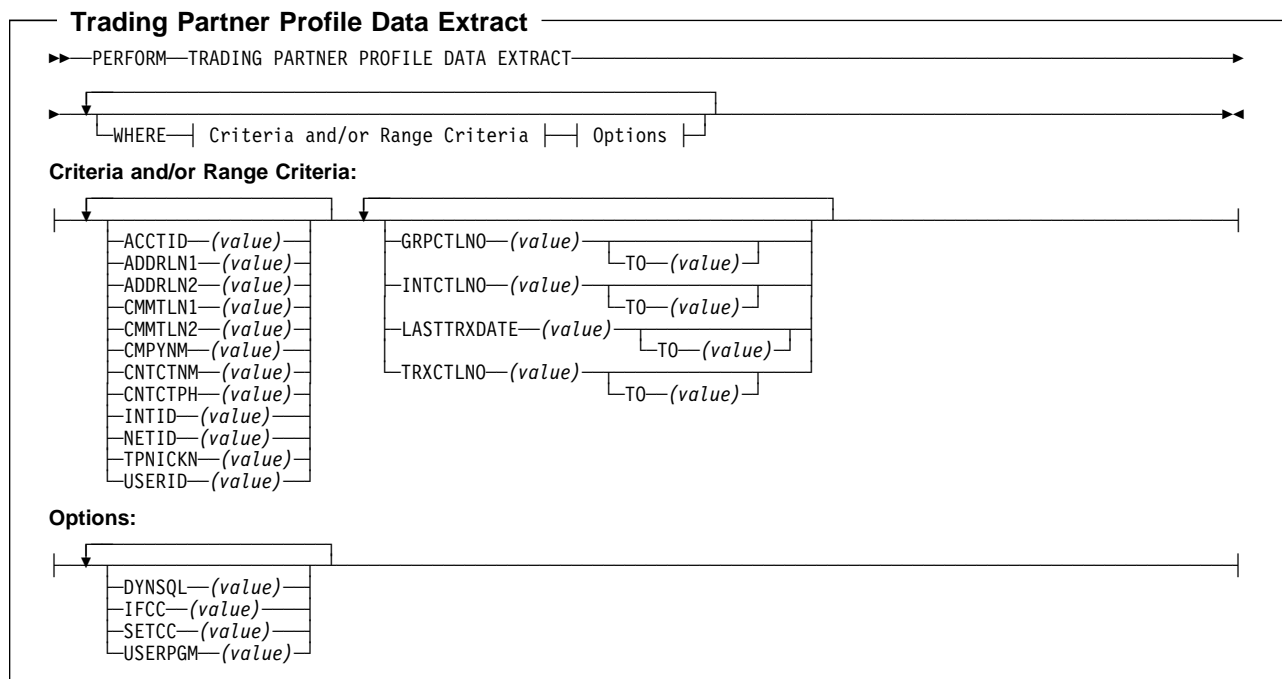
Trading Partner Profile Reports are created by formatting data collected from the PERFORM TRADING PARTNER PROFILE DATA EXTRACT command. Remember, you must run the PERFORM UPDATE STATISTICS command first.

The trading partner data extract command provides detailed information about your trading partners, such as company name, address, contacts, telephone number, nickname, network name, interchange ID, account ID, and user ID. Additionally, the last data transmission date, by trading partner, is provided.

Reports that can be generated from the trading partner data extract command include:

- Inactive trading partners (no activity for X period of time)
- Trading partners by division
- Simple trading partner lists
- Comprehensive trading partner lists

The format of trading partner data extract command is as follows:



TRADING PARTNER PROFILE DATA EXTRACT Command Examples

Example 1: Your EDI Administrator has requested that you provide a hard copy report every Monday morning that shows the trading partners who have sent something within the last six months to the Metal Finishing and Metal Shipping Divisions. This report is used to interface with the various departments and to contact the trading partner in case of trouble. The report contains:

- Company name and location
- Supplier/customer Dun and Bradstreet (DUNS) number
- EDI contact's name and phone number
- Date of the last transaction sent to them
- Control numbers of the last transaction sent to them

```
PERFORM TRADING PARTNER PROFILE DATA EXTRACT
WHERE CMMTLN2(Metal Finishing) LASTTRXDATE(*-180) TO(*)
WHERE CMMTLN2(Metal Shipping) LASTTRXDATE(*-180) TO(*)
```

Or, if these are the only two divisions beginning with Metal, use this command:

```
PERFORM TRADING PARTNER PROFILE DATA EXTRACT
WHERE CMMTLN2(Metal*) LASTTRXDATE(*-180) TO(*)
```

Run the data extract through a report writer or custom program to generate the following report:

Trading Partner Information Report Data									
Company Name	Location	Int ID	Net	Accnt	UserID	Contact	Contact Phone	Last Trx Date	Ctrl No.
A0 Corp	Morris, MN	333219	NETA	MMID00	55532214	Sam Heusen	813-555-5668	08/01/95	00000171
IBM	Tampa, FL	200762	NETA	ADV001	99665593	Deb Garrie	762-772-2328	08/15/97	00990171
Alloy Inc.	Chicago, IL	543189	NETA	ALUM00	18654771	Lou Green	619-432-7784	08/15/96	00044573
DeKant Systems	Queens, NY	337690	IIN	DECANT	USER01	Bob Kling	252-555-9006	11/06/97	00670170
	London, UK	856690	IIN	DECAHH	USER02	Joe Miller	413-545-5555	03/17/98	00006392
	London, UK	009210	IIN	DECA00	USER03	Heather Liu	413-548-2256	03/27/98	00066392
IBM	Boulder, CO	221907	IIN	IBM2	GATEWAY	Hans Gooden	768-434-0968	02/08/98	00006836
	Paris, FR	665767	IIN	IBMIM	01NYC	Mary Lockler	919-409-8585	02/22/98	00601054
	Raleigh, NC	976554	IIN	IBMAP	SANJOSE	Jane Hart	878-319-2876	02/06/98	00070378
MMM Motors	Edison, NJ	332323	NETA	MMM01	MMADMIN	Jon Murphy	828-877-9058	03/18/98	05567836
	Miami, FL	003292	IIN	MMM02	MMTECH	Pauline Gold	813-907-8421	01/22/98	00650054

Example 2: Generate a data extract of all the trading partners for your ABC and DEF divisions with whom you have not exchanged any transactions within the last six months. Print only information about trading partners on the IBM Global Network. Next, put the division in the *comment line 2* field of the trading partner profile.

```
PERFORM TRADING PARTNER PROFILE DATA EXTRACT
WHERE CMMTLN2(ABC) LASTTRXDATE(*-999) TO(*-180) NETID(IN*)
WHERE CMMTLN2(DEF) LASTTRXDATE(*-999) TO(*-180) NETID(IN*)
```

Management Reporting Trading Partner Capability Data Extract

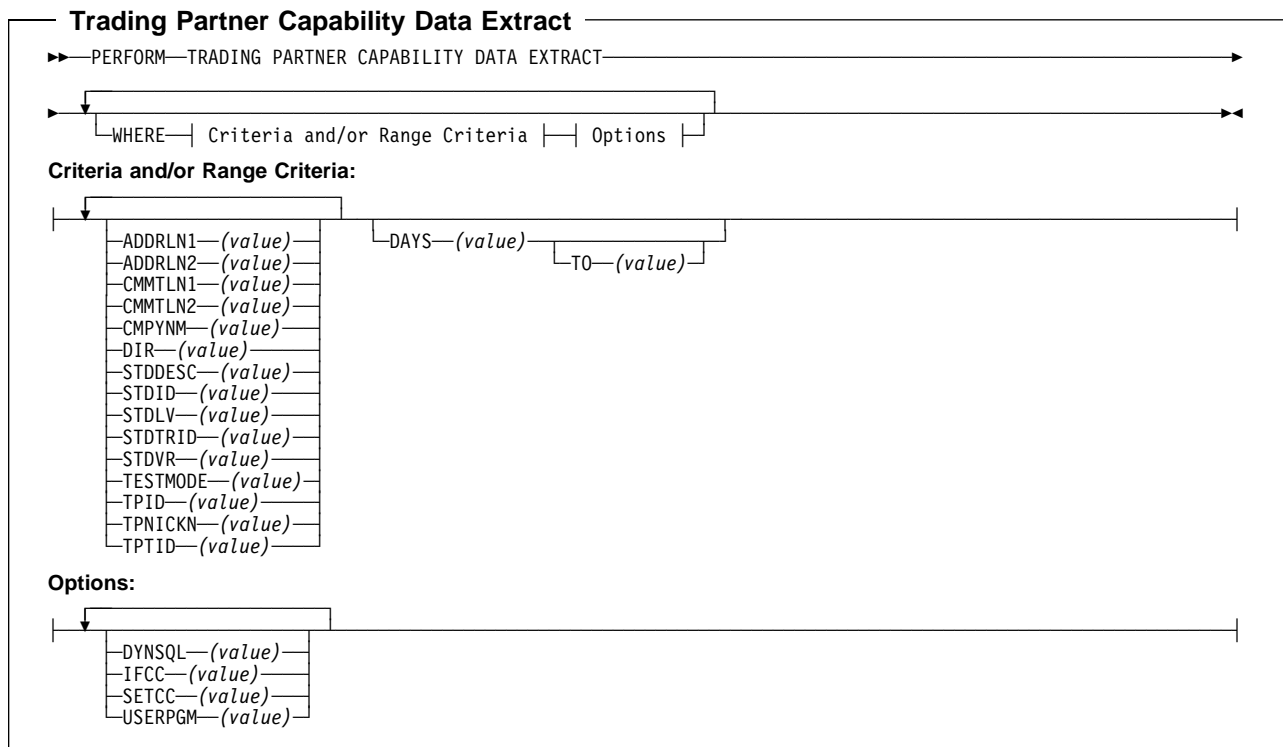
Trading Partner Capability Reports are created by formatting data collected from the PERFORM TRADING PARTNER CAPABILITY EXTRACT command. This command provides data about the transaction capabilities of trading partners including:

- Which transactions have maps, by trading partner
- Total number of transactions translated against these maps
- Total number of transactions with unacceptable error levels that have been translated against these maps

The trading partner capability extract command can be used to create reports such as:

- Pre-Migration status
- Trading Partner Implementation Status
- Trading Partner Testing Status
- Cumulative Transaction Activity by Trading Partner
- Where Used Maps by Trading Partner and vice versa

| The format of PERFORM TRADING PARTNER CAPABILITY EXTRACT command follows:



| TRADING PARTNER CAPABILITY EXTRACT Command Examples

Example 1: Company X is implementing EDI with all their suppliers to their Atlanta assembly plant. They have set a target date of 12/31 to have 100% EDI on several transactions. Suppliers start out in test mode for each new transaction and after 100 test transactions they are eligible to move into production. The EDI steering committee meets monthly to review progress and needs a report that shows information for each supplier, such as:

- All the transactions (messages) they send and receive
- When they went into test on each transaction
- Total number of transactions they have sent or received in test mode
- When they went into production for each transaction
- The total number of transactions

```

PERFORM TRADING PARTNER CAPABILITY DATA EXTRACT
WHERE CMMTLN1(Atlanta) CMMTLN2(Supplier)
  
```

Run the data extract through a report writer or custom program to generate the following report:

Trading Partner Capability Report Data

Company NameID	Location	Int ID	Trx Std.	ID	Dir	Map ID	Started Testing	Test Trx's	Started Productn	Product Trx's
IBM Global Services	Tampa, FL	233119	850	X12V2R1	S	NONPRODX12V2R1P0	07/03/94	101	08/15/94	262
Only1 Co.	Lima, OH	399129	850	X12V2R3	S	PRODSUPX12V2R3P0	08/15/94	71	*NODATE*	0
Pulse Services	Armonk, NY	997223	850	X12V2R3	S	PRODSUPX12V2R3P0	*NODATE*	0	*NODATE*	0
Handlebars Etc.	Raleigh, NC	877519	850	X12V2R3	S	PRODSUPX12V2R3P0	05/07/95	115	07/02/95	554
			855	X12V2R3	R	PRODSUPX12V2R3PA	05/22/95	109	07/26/95	445
			856	X12V2R3	R	PRODSUPX12V2R3SN	06/13/94	187	07/22/94	1878
			861	X12V2R3	S	PRODSUPX12V2R3RA	08/08/94	242	*NODATE*	0
IBM	Dallas, TX	660599	850	X12V2R3	S	PRODSUPX12V2R3P0	01/08/95	126	03/14/95	927
			855	X12V2R3	R	PRODSUPX12V2R3PA	08/22/94	054	*NODATE*	0
			856	X12V2R3	R	PRODSUPX12V2R3SN	04/06/95	378	*NODATE*	0
Pen-n-Paper	Carson, CA	523567	850	X12V2R3	S	PRODSUPX12V2R3P0	02/02/95	113	03/02/95	856
			855	X12V2R3	R	PRODSUPX12V2R3PA	03/26/95	104	03/12/95	535
			856	X12V2R3	R	PRODSUPX12V2R3SN	03/13/95	127	11/23/95	278
			861	X12V2R3	S	PRODSUPX12V2R3RA	04/08/95	102	06/08/95	322

Example 2: Extract information associated with migrating one of your transactions, the purchase order, from X12V2R1 to X12V3R1, and provide a list of all the trading partners and usages you must migrate to the new standard.

```
PERFORM TRADING PARTNER CAPABILITY DATA EXTRACT
WHERE STDID(X12V2R1) STDTRID(850)
```

Management Reporting Transaction Activity Data Extract

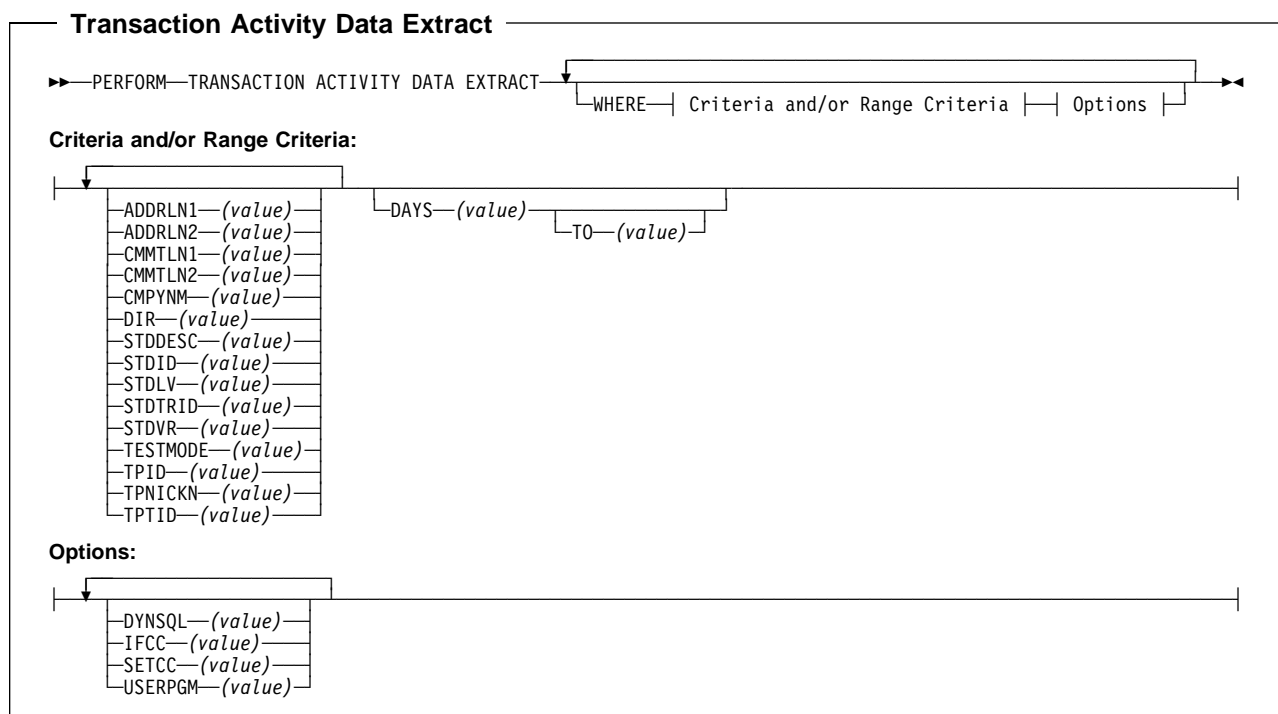
Reports concerning transaction activity are generated by formatting data collected from the PERFORM TRANSACTION ACTIVITY DATA EXTRACT command. The transaction activity extract command provides data about the daily transaction activity of trading partners including:

- The total number of transactions sent or received on any day or range of days, by map ID
- The total number of transactions with errors that were sent or received on any day or range of days, by map ID

Possible uses for this information are:

- Outbound Document Audit Summary (not detailed)
- Inbound Document Audit Summary
- Transaction Activity by Transaction
- Transaction Activity by Trading Partner
- Outbound/Inbound Errors
- Activity Summary by Trading Partner or by Transaction

The format of PERFORM TRANSACTION ACTIVITY DATA EXTRACT command follows:



TRANSACTION ACTIVITY DATA EXTRACT Command Example

Extract a list of all your trading partners and the number of invoices they sent you through EDI in the first six months of 1996. This report can be used to spot-check the accounts receivable system to verify that the invoice receipt process is in control. You receive both EDIFACT and X12 invoices. The Transaction Activity report provides the necessary data listed by trading partner.

```
PERFORM TRANSACTION ACTIVITY DATA EXTRACT
WHERE STDTRID(INVOIC) DAYS(96/01/01) TO(96/06/30) DIR(R)
WHERE STDTRID(810)    DAYS(96/01/01) TO(96/06/30) DIR(R)
```


An example of a transaction activity data extract report, after it has been sorted and formatted follows:

EDI Invoice Activity Summary for 1H96										
Company Name	Location	Int ID	Trx ID	Std. ID	Dir	Trx MTD	Errors MTD	Trx YTD	Errors YTD	
-----	-----	-----	-----	-----	---	---	---	----	----	---
IBM Global Services	Tampa, FL	933489	810	X12V2R1	R	14	1	136	9	
CULATER	Dallas, TX	322217	810	X12V2R3	R	49	5	450	8	
			810	X12V2R3	R	10	5	90	14	
			810	X12V2R3	R	250	23	2200	28	
			INVOIC	EDIL921	R	24	0	224	0	
Definite Inc.	Paris, FR	005901	810	X12V2R3	R	456	0	4104	0	
			810	X12V2R3	R	45	0	405	0	
			INVOIC	X12V2R3	R	789	0	7101	0	
Jay's Turbo	Perry, NC	555489	810	X12V2R3	R	102	2	918	7	
			810	X12V2R3	R	83	3	747	12	
			810	X12V2R3	R	89	0	789	0	
			810	X12V2R3	R	105	0	945	0	
Cool Pool Service	Carmel, CA	748723	810	X12V2R3	R	234	0	1245	2	
			810	X12V2R3	R	216	0	1245	4	

Management Reporting Network Activity Data Extract

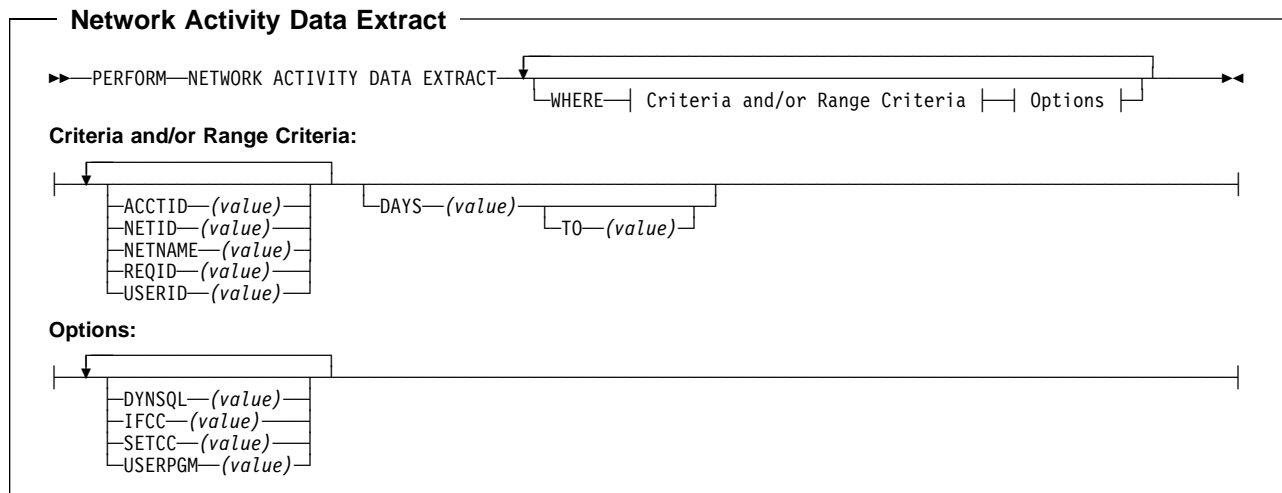
Reports about the network activity of each of your requestors (in the requestor profile) are collected through the PERFORM NETWORK ACTIVITY DATA EXTRACT command. The information provided includes the total number of:

- Envelopes sent by each requestor for any given day or range of days
- Characters sent by each requestor for any given day or range of days

A common use of the Network Activity data is to create reports that:

- Reconcile network charges
- List network usage by user ID
- List heaviest network users
- List inactive requestors

The format of PERFORM NETWORK ACTIVITY DATA EXTRACT command follows:



NETWORK ACTIVITY DATA EXTRACT Command Examples

Example 1: Your network charges jumped significantly last month and you want to understand which of your applications has increased its network usage. To do this, you need a report that shows the:

- Account ID
- User ID
- Number of envelopes exchanged with them, both send and receive
- Number of characters exchanged with them, both send and receive

```
PERFORM NETWORK ACTIVITY DATA EXTRACT
WHERE DAYS(96/08/01) TO(96/08/31)
```

The data that resulted from this command was written to file EDIQUERY, sorted by a local sort utility, and used as input to a user-written program that created the following report:

Network Traffic Activity Report						Date: 96/09/20 Time: 12:31:02	
Month	Network Name	Net ID	Account ID	User ID	S/R	Interchange Envelopes	Total Characters
08/96	IBM Global Network	IINB1	XTEV	PURCHASE	S	7776	9998376
					R	4356	2333157
				ACCTPAYB	S	1526	8867837
					R	662	333457
						-----	-----
						14320	21532827
			XTEW	RECEIVNG	S	856	82251
					R	2856	434457
				TRAFFIC	R	770	66615
						-----	-----
						4482	583323
			IINCICS XTEX	PURCHASE	S	4476	2648376
					R	756	453457
				PRODPO	S	6625	7838902
					R	559	467457
						-----	-----
						12416	11408192

Example 2: Extract information to reconcile your EDI traffic through a particular account (CMPY) user ID (USER21) on the IBM Global Network for the month of May during 1996. Include a list of the requestors that use that account/userid on the IBM Global Network and the number of messages sent and received by each. (Note that this is only the traffic through DataInterchange on the account and user ID.)

```
PERFORM NETWORK ACTIVITY DATA EXTRACT
WHERE ACCTID(CMPY) USERID(USER21) DAYS(96/05/01) TO(96/05/31)
```

Transaction Store Envelope or Transaction Data Extract Overview

DataInterchange's Transaction Store Data Extracts provide commands for extracting detailed envelope and transaction data that can later be formatted into a report. The PERFORM ENVELOPE DATA EXTRACT and PERFORM TRANSACTION DATA EXTRACT commands retrieve information that can be used to:

- Report on the number of purchase orders sent in a given period of time
- Report on the total number of bytes sent in a given period of time (this is useful when it is necessary to charge back costs to other departments based on their EDI usage)
- Create a customized functional acknowledgment tracking report by application or by department; for example, Exception Report on Purchase Orders sent more than 2 days ago that have not been acknowledged
- Create an exception report flagging missing control numbers for inbound envelopes
- Create statistical reports identifying the most frequently used transactions, standards, versions sizes, or delimiters

The envelope and transaction data extract commands can also be used for functions other than reporting, such as:

- Archiving Transaction Store data
- Loading status data directly into the application by application key; for example, the status for each invoice sent could be loaded into the billing system by invoice number

Envelope data extracts include only enveloped transactions, in envelope key order (TP nickname, direction, interchange control number, receiver ID). The transaction data extract includes all transactions and the file is in transaction handle order. For more information on record layouts, see Table 2-62 on page 2-84.

Creating Transaction Store Reports

The steps for using creating Transaction Store reports are:

1. To collect information in the Transaction Store, set the 'Trx. store active?' in the APPDEFS profile to other than N.
2. To collect transaction images in the Transaction Store, set the 'Trx. image wanted?' in the APPDEFS profile to other than N.
3. To collect FA images in the transactions store, set the 'FA image wanted?' to other than N.
4. Perform the usual translations and communications.
5. Perform the Data Extract using the following command:

```
PERFORM DATA EXTRACT
```

Examples of the transaction and envelope extract commands follow:

6. Create user-written program to format the data extract file output (EDIQUERY) into reports.
7. Perform Remove Transactions periodically using the following command:

```
PERFORM REMOVE TRANSACTIONS
```

For more information on this command, see "Purging Documents From the Transaction Store" on page 1-32.

The format of PERFORM ENVELOPE DATA EXTRACT command or PERFORM TRANSACTION DATA EXTRACT command follows:

Envelope or Transaction Data Extract

PERFORM

ENVELOPE

TRANSACTION

DATA EXTRACT SELECTING
Options

WHERE

Criteria and/or Range Criteria

Options:

APPLICATION—(value)

CONCATENATE—(value)

GROUP—(value)

IFCC—(value)

IMAGE—(value)

INTERCHANGE—(value)

RECEIVEACKDATA—(value)

RECEIVEACKIMAGE—(value)

SENDACKDATA—(value)

SENDACKIMAGE—(value)

SETCC—(value)

TRANSACTION—(value)

USERPGM—(value)

Criteria and/or Range Criteria:

APPLID—(value)

APPRECID—(value)

APPSNDID—(value)

BATCH—(value)

DIR—(value)

ENVTYPE—(value)

FORMAT—(value)

FUNACKP—(value)

INTRECID—(value)

INTSNDID—(value)

NETACKP—(value)

NETID—(value)

NETSTAT—(value)

STDTRID—(value)

STSTAT—(value)

TPID—(value)

TPNICKN—(value)

TRERLVL—(value)

TRXSTAT—(value)

ACFIELD—(value)

DLVDATE—(value)

DLVTIME—(value)

ENVDATE—(value)

ENVTIME—(value)

EPURDATE—(value)

GRPCTLNO—(value)

HANDLE—(value)

INTCTLNO—(value)

SNDDATE—(value)

SNDDTIME—(value)

TRXCTLNO—(value)

TRXDATE—(value)

TRXTIME—(value)

Note: The Envelope or Transaction Data Extract commands allow multiple WHERE clauses. All keywords within a WHERE clause are implicitly ANDed, versus WHERE clauses that are implicitly ORed.

ENVELOPE DATA EXTRACT Command Examples

Example 1: The EDI users in Company XYZ want to view the status of their EDI documents for the last month. Specifically, the Purchasing department would like to look up purchase orders, by purchase order number, to determine the daily EDI status. The report must contain:

- Document type (P.O., Invoice, and so on)
- Control numbers
- Date and Time
- Trading partner Nickname
- Reference number (P.O.#, Invoice #, and so on)
- Transaction status

The I/S shop has decided to use a PERFORM ENVELOPE DATA EXTRACT command to collect the necessary information from DataInterchange. They have written a COBOL program, which they run daily, that will format the records into a customized report. Users will read this customized report, searching on their reference number to easily find the status they are interested in.

```
PERFORM ENVELOPE DATA EXTRACT SELECTING
CONCATENATE(Y) INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y)
WHERE HANDLE(19950201) TO(19950328)
```

The following is a customized report:

XYZ COMPANY							
TYPE	CONTROL NUMBER	SET NUMBER	DATE	TIME	TRADING NICKNAME	PO NUMBER or INVOICE NUMBER	STATUS
PO	00000000000375	00000000003386	03/02/95	06:02:93	BOBS	P0100332201	ENVELOPED
PO	00000000000383	00000000003394	03/01/95	11:12:53	BOBS4790	P0100222210	ENVELOPED
PO	00000000000388	00000000003399	03/02/95	11:33:06	BOBS4790	P0100232115	ENVELOPED
PO	00000000000389	00000000003400	03/01/95	06:52:23	BOBS4790		ENVELOPED
PO	00000000000377	00000000003388	03/01/95	11:41:23	BOBS	P0100123130	ENVELOPED
PO	00000000000381	00000000003392	03/01/95	06:22:56	BOBS4790	P0121312322	ENVELOPED
IN	00000000000145	00000000000383	03/02/95	12:53:42	DLGLEVEL	IN94410	RECEIVED
IN	00000000000144	00000000000382	03/02/95	09:12:15	DLGLEVEL	IN95443	RECEIVED
PO	00000010000130	00000000004853	03/02/95	09:43:02	DLGOCCUR	IN76445	ENVELOPED
PO	00000010000130	00000000004854	03/01/95	07:25:16	DLGOCCUR	IN76835	ENVELOPED
PO	00000010000130	00000000004855	03/01/95	11:51:06	DLGOCCUR	IN76337	ENVELOPED
PO	00000010000130	00000000004856	03/02/95	06:34:29	DLGOCCUR	IN76838	ENVELOPED

Example 2: Retrieve the interchange, group, and transaction information for all transactions in a particular envelope with trading partner name PISCES and interchange control number 000008888. Place the results in the EDIQUERY file.

```
PERFORM ENVELOPE DATA EXTRACT SELECTING
INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y)
WHERE TPNICKN(PISCES) INTCTLNO(000008888) DIR(S)
```

Example 3: Retrieve the interchange, group, and transaction information for all envelopes with an application sender ID of SERVO. Include application and image information.

```
PERFORM ENVELOPE DATA EXTRACT SELECTING
INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y) APPLICATION(Y) IMAGE(Y)
WHERE APPSNDID(SERVO) DIR(S)
```

TRANSACTION DATA EXTRACT Command Examples

Example 1: Retrieve the interchange, group, and transaction information for all transactions with trading partner name PISCES and interchange control number 000008888. Place the results in the EDIQUERY file.

```
PERFORM TRANSACTION DATA EXTRACT SELECTING
INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y)
WHERE TPNICKN(PISCES) INTCTLNO(000008888) DIR(S)
```

Example 2: Retrieve the interchange, group, and transaction information for all transactions with an application sender ID of SERVO. Include application and image information.

```
PERFORM TRANSACTION DATA EXTRACT SELECTING  
INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y) APPLICATION(Y) IMAGE(Y)  
WHERE APPSNDID(SERVO) DIR(S)
```

Customizing

The following set of utility commands serves these functions:

- Migrating trading partner transactions maps without having to rework the entire map.
- Exporting and importing administrative data.

Migrating Trading Partner Transactions

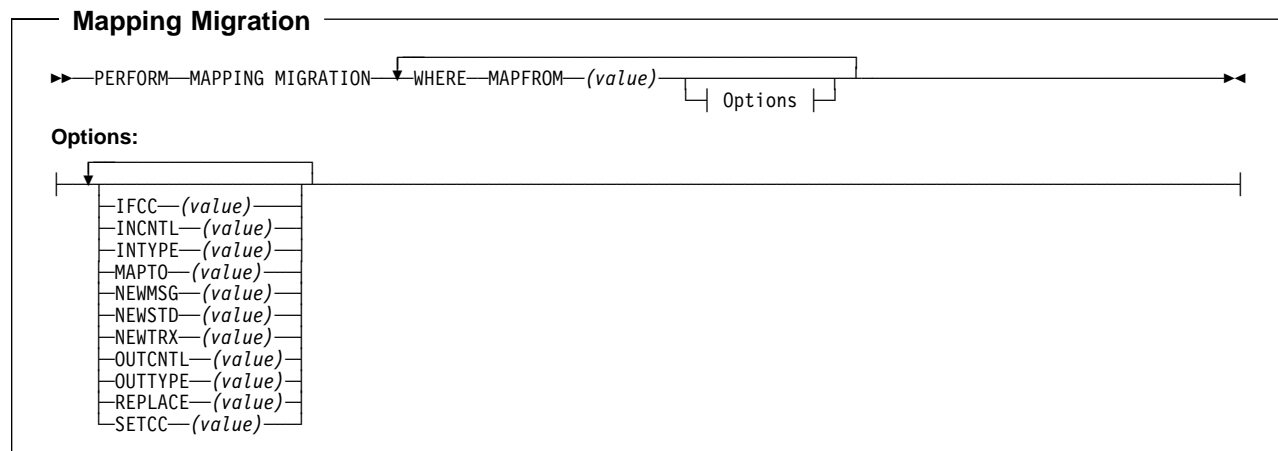
The MAPPING MIGRATION command allows the EDI administrator to migrate a trading partner transaction map to another version of an EDI standard.

Note: The MAPPING MIGRATION command will not migrate the element mapping if any of the elements in the segment were added, deleted, or changed.

The DataInterchange Utility control statements identify:

- The source trading partner transaction
- Optionally, the target trading partner transaction
- Optionally, a replace flag
- Optionally, the new standard to use
- Optionally, the new transaction or message within the standard to use
- Optionally, an input control file to assist in the migration
- Optionally, an output control file
- Optionally, the type of file in CICS (TS queue is the default)

Note: You can use this command to generate a list of segment-level differences in two standard transactions. To view the differences, use the OUTCNTL keyword to specify the data definition name (ddname) of the output file.



MAPPING MIGRATION Command Examples

Example 1: Migrate a mapping from an older version of a standard to a newer version of the same standard. The following command migrates a purchase order mapping from EDIFACT 90.2 to EDIFACT 91.2 and writes mapping migration control records to the file allocated to the data definition name (*ddname*) of OUTCNTL. See “Mapping Migration Output Control File (OUTCNTL)” on page 2-60 for the layout of the OUTCNTL file.

```
PERFORM MAPPING MIGRATION WHERE MAPFROM(ABCORDERRECEIVE)
MAPTO(ABCORDERREC912) NEWSTD(EDI912) OUTCNTL(OUTCNTL)
```

Example 2: Migrate a mapping from one transaction set to another that is very similar. The following command migrates a purchase order mapping to a purchase order change mapping, reading mapping migration control records from the mapping migration input control file (INCNTL) and writing them to the mapping migration output control file (OUTCNTL).

```
PERFORM MAPPING MIGRATION WHERE MAPFROM(ABC850SEND)
MAPTO(ABC860SEND) NEWTRX(860) INCNTL(INCNTL) OUTCNTL(OUTCNTL)
```

Example 3: Refresh a mapping to reflect segments added or deleted in the underlying standard.

Note: You should back up your mapping by exporting it or copying it to another name before refreshing it, just in case the service is unable to match all the segments in the old mapping to segments in the new standard.

```
PERFORM MAPPING MIGRATION WHERE MAPFROM(ABCORDERSEND) REPLACE(Y)
```

Exporting and Importing

The Export/Import commands, like the Export/Import online menus, allow you to exchange setup information (for example, profile members) with your trading partners. For more information about using the Export/Import utility, see the *DataInterchange Administrator's Guide*.

You tell DataInterchange specifically which setup information you want to exchange using a control file. For more information on the Export/Import control file, see “Export/Import Control File (CTLFILE)” on page 2-12.

Exporting Setup Information

The EXPORT command uses the control file to extract setup information from DataInterchange and write records to one or more files (see “Export/Import Control File (CTLFILE)” on page 2-12 for *ddnames*). You can specify the requested file record format with the EIFORMAT keyword on the Perform statement. If the Export/Import file is empty, the requested format is used. Otherwise, the existing file format is used. The file or files can transfer to a trading partner with a system utility.

Export	
▶▶	PERFORM—EXPORT—WHERE—CTLFILE—(value)—————▶▶
Options:	
↓	_____
CTLTYPE—(value)——	
EIFORMAT—(value)——	
IFCC—(value)——	
NOMSG—(value)——	
SETCC—(value)——	

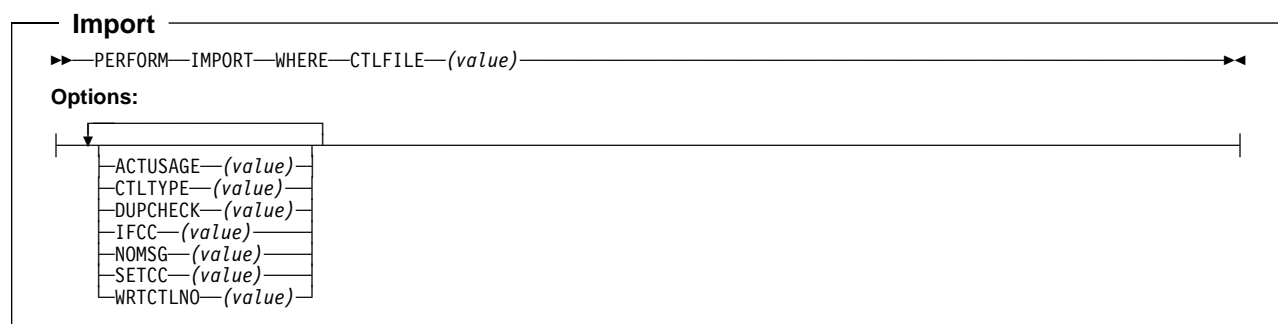
EXPORT Command Example

Example: Export information in fixed format using a control file named EXOUT.

```
PERFORM EXPORT
WHERE CTLFILE(EXOUT) EIFORMAT(FIXED)
```

Importing Setup Information

The IMPORT command uses the control file to read a file or group of files for importing setup information (see “Export/Import Control File (CTLFILE)” on page 2-12 for *ddnames*).



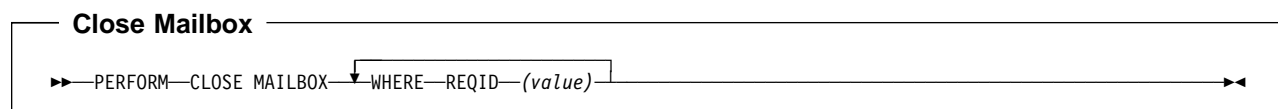
IMPORT Command Example

Example: Import all records specified by the batch control file that is in a TS queue named INCTL.

```
PERFORM IMPORT
WHERE CTLFILE(INCTL) CTLTYPE(TS)
```

Managing Mailboxes

DataInterchange for CICS does not wait for send requests to complete when using Expedite/CICS and Information Exchange. Once a send, receive, continuous receive, network status update, or any other type of access to the Information Exchange mailbox is made, the mailbox remains open. If you want to close the mailbox, an explicit command must be issued to end the session. This should be unnecessary unless the same mailbox will be used in the MVS environment. The command is only available in the CICS environment and is not required for general processing.



CLOSE MAILBOX Command Example

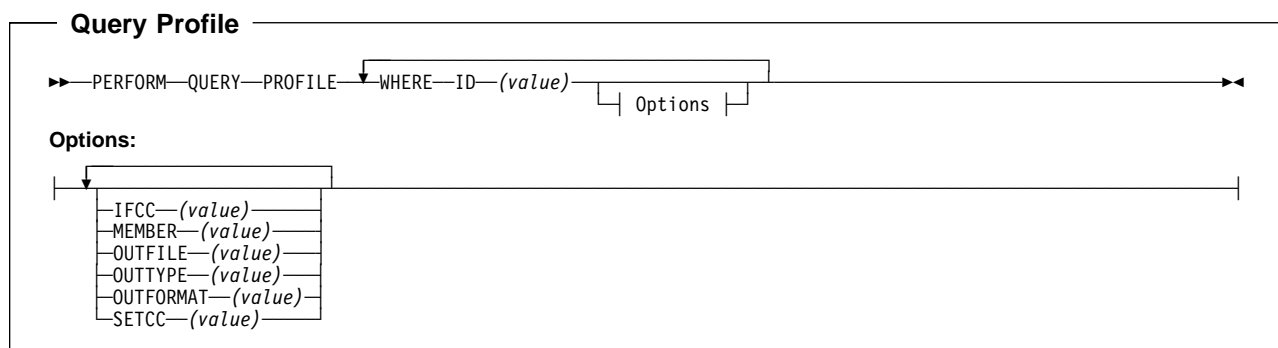
Example: Close the mailbox for requestor DEPTA7F.

```
PERFORM CLOSE MAILBOX
WHERE REQID(DEPTA7F)
```

Profile Maintenance

Querying Profiles

It is possible to query profiles. The results are written in tagged, fixed, or native format to the file specified in the command. The default format is fixed. EDIQUERY is the default output file name, and TS is the default output file type in CICS. If no member is specified in the command, the entire profile is queried.



QUERY PROFILE Command Examples

Example 1: Query all trading partner profile members and write results to MYFILE in fixed format:

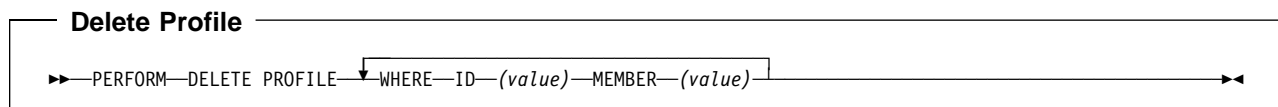
```
PERFORM QUERY PROFILE WHERE ID(TPPROF) OUTFILE(MYFILE)
```

Example 2: Query requestor profile member REQMEM and write results to MYFILE in native format:

```
PERFORM QUERY PROFILE WHERE ID(REQPROF) MEMBER(REQMEM)  
OUTFILE(MYFILE) OUTFORMAT(N)
```

Deleting Profile Members

Profile members can be deleted. However, it is not possible to delete trading partner profile members where an associated usage or an associated IEF component still exists.



DELETE PROFILE Command Examples

Example 1: Delete trading partner profile member TPMEM.

```
PERFORM DELETE PROFILE WHERE ID(TPPROF) MEMBER(TPMEM)
```

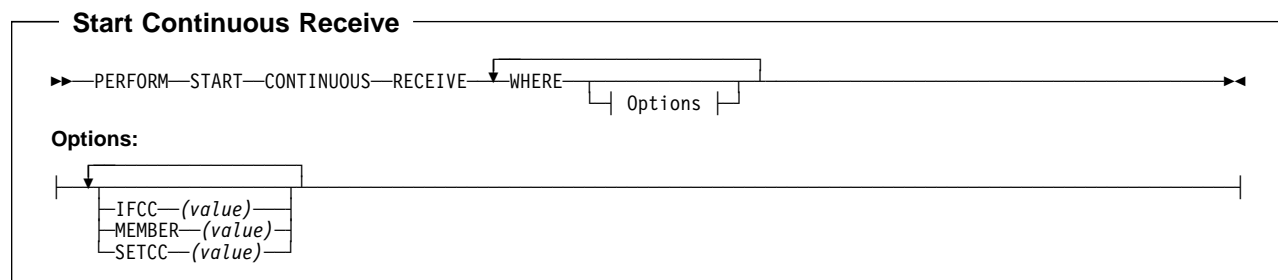
Example 2: Delete requestor profile member REQMEM.

```
PERFORM DELETE PROFILE WHERE ID(REQPROF) MEMBER(REQMEM)
```

Continuous Receive Functions

Starting Continuous Receives

This command is available only in the CICS environment. Like the CICS transaction EDIR, this command can be used to start a single continuous receive or start all eligible continuous receives. A continuous receive is eligible to be started if: a valid requestor ID is specified in the continuous receive profile member, the continuous receive profile member is marked active, and the network ID in the associated requestor profile member is IINCICS.



START CONTINUOUS RECEIVE Command Examples

Example 1: Start all continuous receives.

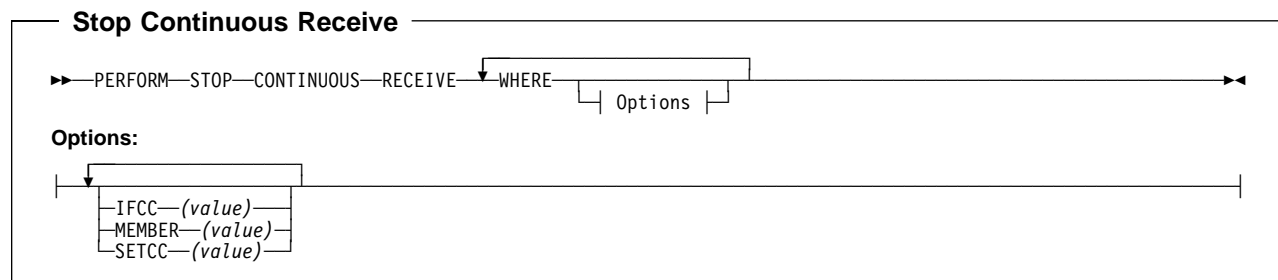
```
PERFORM START CONTINUOUS RECEIVE
```

Example 2: Start continuous receive profile member CRMEM.

```
PERFORM START CONTINUOUS RECEIVE WHERE MEMBER(CRMEM)
```

Stopping Continuous Receives

This command is available only in the CICS environment. Like the CICS transaction EDIS, this command can be used to stop a single continuous receive or stop all eligible continuous receives. A continuous receive is eligible to be stopped if: a valid requestor ID is specified in the continuous receive profile member, the continuous receive profile member is marked active, and the network ID in the associated requestor profile member is IINCICS.



STOP CONTINUOUS RECEIVE Command Examples

Example 1: Stop all continuous receives.

```
PERFORM STOP CONTINUOUS RECEIVE
```

Example 2: Stop continuous receive profile member CRMEM.

```
PERFORM STOP CONTINUOUS RECEIVE WHERE MEMBER(CRMEM)
```

Reporting Continuous Receive Statuses

When a continuous receive is started through DataInterchange for CICS, DataInterchange keeps a control record about the continuous receive. Expedite/CICS keeps a separate control record. It is possible for DataInterchange and Expedite/CICS to get out-of-sync in managing continuous receives, but this would be an exception to the normal processing. A continuous receive status report reports the status of either a single continuous receive member or all continuous receives known to DataInterchange and Expedite/CICS. The output goes to the report file. The report file name and type can be specified in the Utility Control Information block. The default is RPTFILE (type TS in CICS).

Table 1-4. Continuous Receive Status Report Statuses

Report Status	Description
STARTED	This continuous receive is considered to be running by both DataInterchange and Expedite/CICS, and a continuous receive profile member exists.
DI STARTED	This continuous receive is considered to be running by DataInterchange, but not by Expedite/CICS. A continuous receive profile member exists.
EXP STARTED	This continuous receive is considered to be running by Expedite/CICS, but not by DataInterchange. A continuous receive profile member exists.
STARTED N/P	This continuous receive is considered to be running by both DataInterchange and Expedite/CICS. However, a continuous receive profile member does not exist.
DI STARTED N/P	This continuous receive is considered to be running by DataInterchange, but not by Expedite/CICS. A continuous receive profile member does not exist.
EXP STARTED N/P	This continuous receive is considered to be running by Expedite/CICS, but not by DataInterchange. A continuous receive profile member does not exist.
NOT STARTED	This continuous receive is not considered to be running by either DataInterchange or Expedite/CICS. However, a continuous receive profile member does exist and it is eligible to be started.
NOT STARTED N/A	This continuous receive is not considered to be running by either DataInterchange or Expedite/CICS. A continuous receive profile member does exist, but it is not eligible to be started.

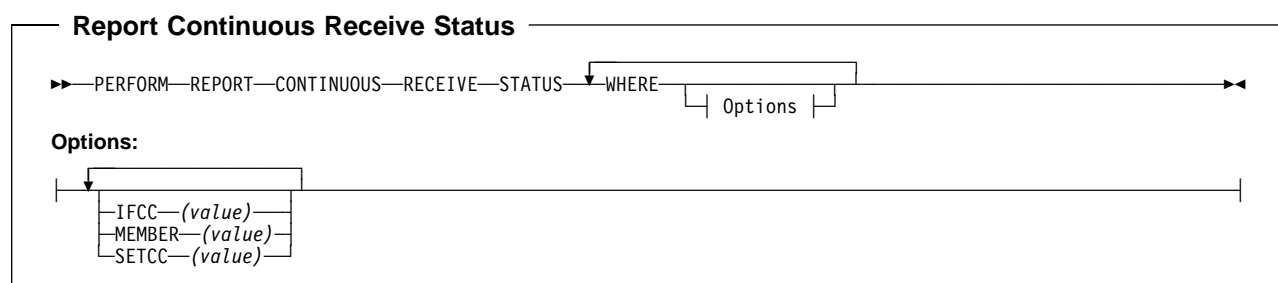
Note: The continuous receive report statuses do not necessarily mean good versus bad states of the continuous receive. For example, EXP STARTED N/P could very well be okay and perfectly describe the state of that particular continuous receive. On the other hand, a status of EXP STARTED might be a problem and indicate an out-of-sync situation between DataInterchange and Expedite/CICS. If an error is encountered while generating the report, the processing program immediately terminates and does not report on subsequent continuous receives. For more information about out-of-sync situations, see “Continuous Receive Session Cleanup” on page 5-39.

Table 1-5 (Page 1 of 2). Continuous Receive Status Report Record Format

Name	Type	Offset	Length	Description
CRSTATUS	CHAR	0	16	Report status
CDPROFID	CHAR	16	16	DI Continuous receive profile member
CDREQACT	CHAR	32	08	DI Requestor's account
CDREQUSE	CHAR	40	08	DI Requestor's user ID

Table 1-5 (Page 2 of 2). Continuous Receive Status Report Record Format

Name	Type	Offset	Length	Description
CDUNIQUE	CHAR	48	08	DI continuous receive unique ID
CDMSGUCL	CHAR	56	08	DI message user class
CDTPACCT	CHAR	64	08	DI trading partner's account
CDTPUSER	CHAR	72	08	DI trading partner's user ID
CEREQACT	CHAR	80	08	Exp/CICS account
CEREQUSE	CHAR	88	08	Exp/CICS user ID
CEUNIQUE	CHAR	96	08	Exp/CICS continuous receive unique ID
CEMSGUCL	CHAR	104	08	Exp/CICS message user class
CETPACCT	CHAR	112	08	Exp/CICS trading partner's account
CETPUSER	CHAR	120	08	Exp/CICS trading partner's user ID
RESERVED	CHAR	128	04	Blanks



REPORT CONTINUOUS RECEIVE STATUS Command Examples

Example 1: Report all continuous receives.

```
PERFORM REPORT CONTINUOUS RECEIVE STATUS
```

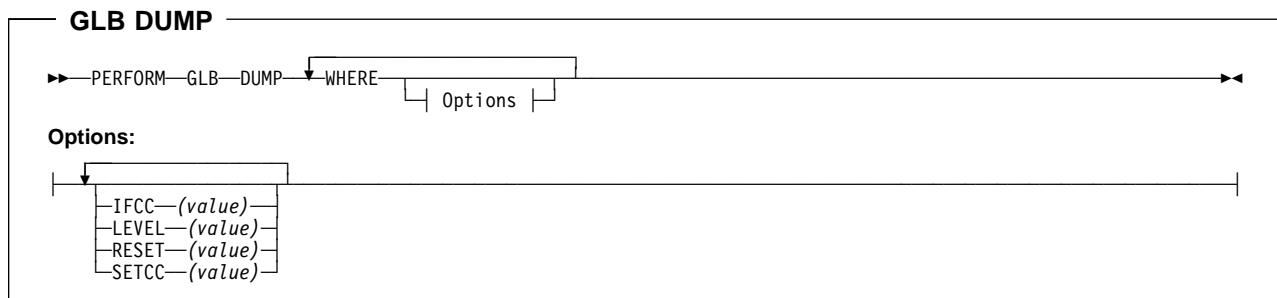
Example 2: Report the status of continuous receive profile member CMEM.

```
PERFORM REPORT CONTINUOUS RECEIVE STATUS WHERE MEMBER(CMEM)
```

Persistent Environment Debugging

Obtaining a Persistent Environment Dump

The Persistent Environment is a DataInterchange for CICS optional feature available with CICS/ESA. This command is available only in the CICS environment and is not required for normal processing. Instead, this command can be used to gather information required by DataInterchange support personnel when debugging problems related to the Persistent Environment. The dump is written to the file associated with *ddname* EDIGDMP1 in the CICS startup JCL.



GLB DUMP Command Examples

Example 1: Open EDIGDMP1 for output and generate a complete Persistent Environment dump.

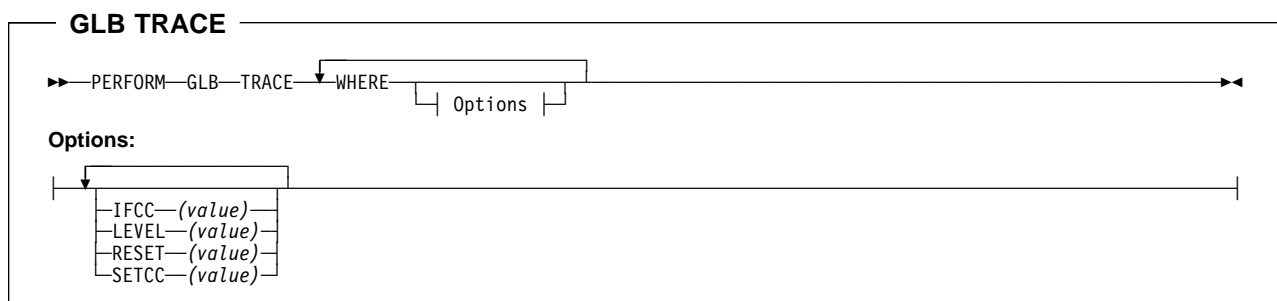
```
PERFORM GLB DUMP WHERE LEVEL(1) RESET(Y)
```

Example 2: Open EDIGDMP1 for append and generate a Persistent Environment dump of just the data area.

```
PERFORM GLB DUMP WHERE LEVEL(12)
```

Obtaining a Persistent Environment Trace

The Persistent Environment is a DataInterchange for CICS optional feature available with CICS/ESA. This command is available only in the CICS environment and is not required for normal processing. Instead, this command can be used to gather information required by DataInterchange support personnel when debugging problems related to the Persistent Environment. The trace is written to the file associated with *ddname* EDIGTRC1 in the CICS startup JCL.



GLB TRACE Command Examples

Example 1: Open EDIGTRC1 for output and start a Persistent Environment trace for as much detail as possible:

```
PERFORM GLB TRACE WHERE LEVEL(3) RESET(Y)
```

Example 2: End the Persistent Environment trace:

```
PERFORM GLB TRACE WHERE LEVEL(0)
```

Keyword, Option, Criteria, and Range Criteria Descriptions

This section describes the keywords, options, criteria, and range criteria used with the DataInterchange Utility commands. Keywords, options, criteria, and range criteria description follow:

ACCTID The Network Account ID of the trading partner as entered in the Account Number field of the Trading Partner Profile. This keyword is used with the following commands:

NETWORK ACTIVITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT

ACFIELD Specifies either the concatenation of data in the fields specified during transaction mapping as application control fields or the AC field in the data format. For more information about the AC field, see the *DataInterchange Administrator's Guide*. The concatenated field is used if any application control field names are specified in transaction mapping. This field is case sensitive. The maximum length is 35. This keyword is used with the following commands:

ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE

ACKFILE Specifies the ddname of a file containing network acknowledgments. If this keyword is not supplied, the value of the "Net output file" in the associated network profile member will be used.

Note: In CICS, DataInterchange does not uppercase the ACKFILE value.

This keyword is used with the following command:

PROCESS NETWORK ACKS

ACKTYPE Acknowledgments file type used only for CICS (except MQ, which is supported in both MVS and CICS). Valid values are:

Value	Description
MQ	DataInterchange MQSeries Queue profile member name
TD	Transient data queue
TM	Temporary storage queue (main storage)

TS Temporary storage queue (auxiliary storage)
 VS VSAM entry sequenced data set

This keyword is used with the following command:

PROCESS NETWORK ACKS

If the keyword is not provided for MVS, this field is ignored; instead, the default for the corresponding file keyword is the ddname of a sequential file. For CICS, the default is TS.

ACTUSAGE Indicates how the activation of imported usages should be handled. Valid values are:

Value	Description
NOREPL	Specifies that an active imported usage should not replace an existing active usage in the database. If an active usage that matches the key for the imported usage already exists, the imported usage is made inactive. This is the default value.
REPL	Specifies that an active imported usage should replace an existing active usage in the database. In this case, the existing usage is made inactive and the imported usage becomes the active usage.
FORCE	Forces all imported usages to be active regardless of their current status or the existence of another active usage in the database. If an active usage that matches the key for the imported usage exists in the database, it is made inactive.

This keyword is used with the following commands:

IMPORT

ADDRLN1 The first line of the address as entered in the *Address line 1* field of the trading partner profile. This field is case sensitive. This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
 TRADING PARTNER PROFILE DATA EXTRACT
 TRANSACTION ACTIVITY DATA EXTRACT

ADDRLN2 The second line of the address as entered in the *Address line 2* field of the trading partner profile. This field is case sensitive. This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
 TRADING PARTNER PROFILE DATA EXTRACT
 TRANSACTION ACTIVITY DATA EXTRACT

APPFILE When used with EDI processing commands, this keyword specifies the ddname of the file containing the application data. The data can be C and D records or raw data records. For the format of these records, see "Application File" on page 2-3. The maximum length is 8.

When used with the PRINT CUSTOM LAYOUT command, this keyword points to a fixed-block, 80-byte QSAM file containing a list of ddnames. These ddnames must contain the application data in C and D record format. The report data is written to the RPTFILE ddname in the job.

This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
 TRANSLATE AND SENT
 TRANSLATE TO STANDARD
 DEENVELOPE AND TRANSLATE
 RECEIVE AND TRANSLATE

TRANSLATE TO APPLICATION
PRINT CUSTOM LAYOUT

Note: In CICS, DataInterchange does not uppercase the APPFILE value.

APPLICATION Indicates if an application data record is to be written to the EDIQUERY file. Valid values are:

Value	Description
Y	Keeps application data record
N	Discards application data record (default)

Received transactions can be translated multiple times and have an application record for each attempt. Send transactions only have one application record. For the format of these records, see "Application Data Extract Record Layout" on page 2-86. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

APPLID Application ID. When used as a selection criteria, APPLID specifies the application ID that was used to create the records being examined (these may be transactions in the Transaction Store or entries in the event log). Provided application IDs are:

ID	Application
EDIMP	DataInterchange Facility
EDIFFS	DataInterchange Utility
(user defined)	Your application APPLID

An application ID can be switched by using DataInterchange's application program interface function codes. The maximum APPLID field length is 8. This keyword is used with the following commands:

ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
LOAD LOG ENTRIES
NETWORK ACTIVITY DATA EXTRACT
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE LOG ENTRIES
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
UNLOAD LOG ENTRIES
UNPURGE

APPRECID Application receiver ID. Identifies the specific department or business area in the receiver's company. This is the ID the translator first attempts to use to locate the trading partner usage for receive transactions. The maximum length is 35. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE

APPSEC Application file security. This determines if the application files specified through the APPFILE keyword are read only or read and write. Valid values are:

Value	Description
U	Indicates the application files are read/write and can be opened for extend and then closed (default). This process makes sure each file has been properly initialized and DataInterchange does not process data that is not valid.
R	Indicates the application files should be opened in read only mode. If this option is chosen, you must make sure the application files are properly initialized.

This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD

APPSNDID Identifies the specific department or business area in the sender's company. The maximum length is 35. This keyword is used with the following commands:

HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE

REENVELOPE AND SEND
 RELEASE
 REMOVE TRANSACTIONS
 RETRANSLATE TO APPLICATION
 TRANSLATE TO APPLICATION
 UNPURGE

APPTYPE Application file type used only for CICS (except MQ, which is supported in both MVS and CICS). Valid values are:

Value	Description
MQ	DataInterchange MQSeries Queue profile member name
TD	Transient data queue
TM	Temporary storage queue (main storage)
TS	Temporary storage queue (auxiliary storage)
VS	VSAM entry sequenced data set
	Inbound processing only
PG	Response program
TX	Response transaction

This keyword is used with the following commands:

DEENVELOPE AND TRANSLATE
 RECEIVE AND TRANSLATE
 TRANSLATE AND ENVELOPE
 TRANSLATE AND SEND
 TRANSLATE TO STANDARD
 TRANSLATE TO APPLICATION
 PRINT CUSTOM LAYOUT

If the keyword is not provided for MVS, this field is ignored; instead, the default for the corresponding file keyword is the ddname of a sequential file. For CICS, the default is TS.

ARCHIVEFILE Event log archive file name. Event log entries selected for removal are written to this file. The maximum length is 8. This keyword is used with the following commands:

REMOVE LOG ENTRIES
 UNLOAD LOG ENTRIES

ARCHIVETYPE

Event log archive file type used only for CICS (except MQ, which is supported in both MVS and CICS). Valid values are:

Value	Description
MQ	DataInterchange MQSeries Queue profile member name
TD	Transient data queue
TM	Temporary storage queue (main storage)
TS	Temporary storage queue (auxiliary storage)
VS	VSAM entry sequenced data set

This keyword is used with the following commands:

REMOVE LOG ENTRIES
 UNLOAD LOG ENTRIES

If the keyword is not provided for MVS, this field is ignored; instead, the default for the corresponding file keyword is the ddname of a sequential file. For CICS, the default is TS.

ASSERTLVL Determines the level of assertions that are active for this translation session. Assertions are established during the mapping process using the &ASSERT n special literals where the n can be a value from 0 to 9, which establishes 10 assertion levels. ASSERTLVL indicates which of these levels should be active because only &ASSERTions with an n value greater than or equal to the ASSERTLVL value will be evaluated. Thus, with ASSERTLVL(5) specified, only special literals &ASSERT5, &ASSERT6, &ASSERT7, &ASSERT8, and &ASSERT9 will be evaluated. Level 9 assertions (&ASSERT9) are always active.

This keyword is used with the following commands:

- DEENVELOPE AND TRANSLATE
- RECEIVE AND TRANSLATE
- RETRANSLATE TO APPLICATION
- TRANSLATE AND ENVELOPE
- TRANSLATE AND SEND
- TRANSLATE TO APPLICATION
- TRANSLATE TO STANDARD

BATCH Specifies the batch ID assigned to a transaction when it was translated. Use this keyword to select transactions with the batch ID you specify. The maximum length is 8. This keyword is used with the following commands:

- ENVELOPE
- ENVELOPE AND SEND
- ENVELOPE DATA EXTRACT
- HOLD
- PRINT ACKNOWLEDGMENT IMAGE
- PRINT ACTIVITY SUMMARY
- PRINT EVENT LOG
- PRINT STATUS SUMMARY
- PRINT STATUS SUMMARY2
- PRINT TRANSACTION DETAILS
- PRINT TRANSACTION IMAGE
- PURGE
- QUERY
- RECEIVE AND DEENVELOPE
- RECEIVE AND TRANSLATE
- REENVELOPE
- REENVELOPE AND SEND
- RELEASE
- REMOVE TRANSACTIONS
- RETRANSLATE TO APPLICATION
- TRANSACTION DATA EXTRACT
- UNPURGE

BATCHSET Specifies a user-defined ID to be assigned to the transaction when it is translated. Use this keyword to relate transactions you want to retrieve as a group. The transactions remain independent. An action performed on one transaction does not affect other transactions with the same batch ID. The maximum length is 8.

The default batch ID is a date and time stamp in the form *ddhhmmss*. This keyword is used with the following commands:

- DEENVELOPE AND TRANSLATE
- RECEIVE AND TRANSLATE
- RETRANSLATE TO APPLICATION

TRANSLATE AND ENVELOPE
 TRANSLATE AND SEND
 TRANSLATE TO APPLICATION
 TRANSLATE TO STANDARD

CCEXCEPTION

Indicates whether job step condition code 0003 or 0000 is returned if application data is written to the exception file. Valid values are:

Value	Description
Y	Generates a minimum job step condition code of 0003 if application data is written to the exception file.
N	Does not report application data written to the exception file as an error. If translation completes successfully and the application data was written to the exception file, return 0000 as the job step condition code (default).
X	Works just like a value of Y, except instead of 3, a job step condition code of 903 is generated to distinguish this from other errors that result in a 0003 condition code.

This keyword is used with the following commands:

DEENVELOPE AND TRANSLATE
 RECEIVE AND TRANSLATE
 RETRANSLATE TO APPLICATION
 TRANSLATE TO APPLICATION

CLEARFILE

Indicates if the transaction data queue file should be cleared after the send is complete or before the receive is issued. Valid values are:

Value	Description
Y	Clears the file after the send is complete or before the receive is issued
N	Does not clear the file after the send is complete or before the receive is issued (default)

This keyword is used with the following commands:

ENVELOPE AND SEND
 SEND
 RECEIVE
 RECEIVE AND DEENVELOPE
 RECEIVE AND TRANSLATE
 REENVELOPE AND SEND
 TRANSLATE AND SEND

Note: This keyword is ignored for batch receive processes. The file allocation disposition in the batch JCL will determine if the transaction data queue file is cleared.

CLIENT

Extracts or removes SAP status records using the SAP client ID. The default is ALL.

This keyword is used with the following commands:

SAP STATUS EXTRACT
 SAP STATUS REMOVE

CMMTLN1

The first comment line as entered in the *Comment line 1* field of the trading partner profile. This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT

TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

CMMTLN2 The second comment line as entered in the *Comment line 2* field of the trading partner profile. This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

CMPLYNM The company name entered in the *Company name* field of the trading partner profile. This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

CNTCTNM The contact name as entered in the *Contact name* field of the trading partner profile. This keyword is used with the following commands:

TRADING PARTNER PROFILE DATA EXTRACT

CNTCTPH The contact phone number as entered in the *Contact phone number* field of the trading partner profile. This keyword is used with the following commands:

TRADING PARTNER PROFILE DATA EXTRACT

CONCATENATE

Determines if requested data extract information is written as separate records or concatenated and written as a single record. Valid values are:

Value	Description
-------	-------------

Y	Specifies categories are concatenated and written as a single record
---	--

N	Specifies information is written as separate records (default)
---	--

If concatenation is requested, the following hierarchy is used:

1. Interchange
2. Group
3. Transaction
4. Application

Each application entry for a transaction is written with duplicate interchange, group, and transaction information.

Note: Concatenation does not apply to IMAGE records. Transaction and acknowledgment images are always written as separate records. Concatenation always applies for detailed acknowledgment data.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

CTLFILE Specifies the ddname of an import or export control file that describes what data is to be imported or exported. For more information, see Table 2-2 on page 2-13. This keyword is used with the following commands:

EXPORT
IMPORT

CTLTYPE An import or export control file type used only for CICS (except MQ, which is supported in both MVS and CICS). Valid values are:

Value	Description
MQ	DataInterchange MQSeries Queue profile member name
TD	Transient data queue
TM	Temporary storage queue (main storage)
TS	Temporary storage queue (auxiliary storage)
VS	VSAM entry sequenced data set

This keyword is used with the following commands:

EXPORT
IMPORT

If the keyword is not provided for MVS, this field is ignored; instead, the default for the corresponding file keyword is the ddname of a sequential file. For CICS, the default is TS.

DAYS A single date, or the start date of a date range. If it is the start date of a date range, it must be followed by the TO() keyword. This keyword is used with the following commands:

NETWORK ACTIVITY DATA EXTRACT
SAP STATUS EXTRACT
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

DIERRFILTER The initial set of errors that should be filtered for this translation session. For more information on error filtering, see “Error Filtering” on page 1-4, the field definition for ERRFILTER on page 3-30, or the *DataInterchange Administrator’s Guide*. This keyword is used with the following commands:

TRANSLATE TO STANDARD
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND
DEENVELOPE
TRANSLATE TO APPLICATION
RETRANSLATE TO APPLICATION
RECEIVE AND TRANSLATE
RECEIVE AND DEENVELOPE
DEENVELOPE AND TRANSLATE

DIR Indicates the direction to select an S (send) or an R (receive). This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY

RECONSTRUCT
RECONSTRUCT AND SEND
RELEASE
REMOVE TRANSACTIONS
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
TRANSACTION DATA EXTRACT
UNPURGE

DLVDATE Specifies the date, or a range of dates, when the transactions you want to work with were delivered to the application. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
UNPURGE

DLVTIME Specifies the time, or a period of time, when the transactions you want to work with were delivered to the application. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
UNPURGE

DUPCHECK Indicates if the duplicate account/user ID and duplicate interchange ID/qualifier checks are to be performed when importing TPPROF members. This keyword is optional and if not specified, the default is Y.

Note: The import program checks the incoming TPPROF members to ensure that no duplicate accounts/user IDs or duplicate interchange IDs/qualifiers are imported. This is a time-consuming task and if you are confident that there are no duplicates in the import file, this keyword can be specified with a value of N to improve the performance of the

import. If duplicates are imported into the TPPROF and are referenced during translation, unpredictable results can occur.

Valid values are:

Value	Description
N	Bypass the duplicate checks on TPPROF import
Other	Any value other than N performs the duplicate checks

This keyword is used with the following command:

IMPORT

DUPENV Specifies if duplicate envelopes should be processed. Valid values are:

Value	Description
Y	Processes duplicate envelopes (default)
N	Does not process duplicate envelopes

A condition code of 0005 is returned if a value of N is specified and you attempt to process a duplicate envelope. This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

DYNSQL Specifies that dynamic SQL should be used for extracting the management report data. Dynamic SQL provides significant performance improvements if you maintain a lot of statistics. It allows DB2 to create an optimized plan specifically for your query. To use dynamic SQL, you must have authorization by your DBA to the appropriate views. Valid values are:

Value	Description
Y	Use dynamic SQL
N	Use static SQL (default)

This keyword is used with the following commands:

NETWORK ACTIVITY DATA EXTRACT
TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

EENVDATE The earliest date that transactions created during this run can be enveloped. The Date mask field in the language profile determines the format of the date code. This keyword is used with the following command:

TRANSLATE TO STANDARD

EIFORMAT Specifies the requested export file record format. This keyword is optional and if not specified, the default is TAGGED. Valid values are:

Value	Description
TAGGED	Tagged record format
FIXED	Fixed record format

This keyword is used with the following command:

EXPORT

ENVDATE Specifies the date on which the transaction was enveloped. The maximum length is 10. This keyword is used with the following commands:

ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
TRANSACTION DATA EXTRACT
TRANSLATE TO STANDARD
UNPURGE

ENVPRBREAK Specifies if a new interchange envelope or a new group envelope is started when the standard envelope profile member name changes. Usually, the envelope profile provides the group envelope information. Use this keyword if your envelope profile provides interchange envelope information. Valid values are:

Value	Description
Y	Starts a new interchange envelope
N	Starts a new group envelope (default)

This keyword is used with the following commands:

ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND

ENVTIME Specifies the time at which the transaction was enveloped. The maximum length is 8. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE

REENVELOPE AND SEND
 RELEASE
 TRANSACTION DATA EXTRACT
 REMOVE TRANSACTIONS
 UNPURGE

ENVTYPE Specifies the type of envelope used for the transactions to select. The maximum length is 1. Valid values are:

Value	Envelope
X	ISA/IEA
E	UNB/UNZ
U	BG/EG
I	ICS
T	STX/END
0	Envelopes with no interchange header and trailer

For SEND and RECEIVE, the type of receive issued. Valid values are:

Value	Description
X	X12 (default)
E	EDIFACT
I	ICS or nonstandard file
T	UNTDI
U	UCS

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
 HOLD
 PRINT ACKNOWLEDGMENT IMAGE
 PRINT ACTIVITY SUMMARY
 PRINT EVENT LOG
 PRINT STATUS SUMMARY
 PRINT STATUS SUMMARY2
 PRINT TRANSACTION DETAILS
 PRINT TRANSACTION IMAGE
 PURGE
 QUERY
 RECEIVE
 RECEIVE AND DEENVELOPE
 RECEIVE AND TRANSLATE
 REENVELOPE
 REENVELOPE AND SEND
 RELEASE
 REMOVE TRANSACTIONS
 RETRANSLATE TO APPLICATION
 SEND
 TRANSACTION DATA EXTRACT
 TRANSLATE TO APPLICATION
 UNPURGE

EPURDATE Specifies the date on which the transaction is purged from the Transaction Store. The translator sets the default purge date when it adds the transaction to the Transaction Store. You can override the default length of time the transaction can stay in the Transaction Store before being purged by using the PURGINT keyword when translating

the transaction. The maximum number of characters in this field is 10. This keyword is used with the following commands:

ENVELOPE
 ENVELOPE AND SEND
 ENVELOPE DATA EXTRACT
 HOLD
 PRINT ACKNOWLEDGMENT IMAGE
 PRINT ACTIVITY SUMMARY
 PRINT EVENT LOG
 PRINT STATUS SUMMARY
 PRINT STATUS SUMMARY2
 PRINT TRANSACTION DETAILS
 PRINT TRANSACTION IMAGE
 PURGE
 QUERY
 REENVELOPE
 REENVELOPE AND SEND
 RELEASE
 REMOVE TRANSACTIONS
 RETRANSLATE TO APPLICATION
 TRANSACTION DATA EXTRACT
 TRANSLATE TO APPLICATION
 UNPURGE

EXTENDC

Used when translating to application data with C and D records. Indicates whether or not to use the extended C record format.

Value	Description
Y	Use the extended C record format
N	Do not use the extended C record format. This is the default.

This keyword is used with the following commands:

DEENVELOPE AND TRANSLATE
 RECEIVE AND TRANSLATE
 RETRANSLATE TO APPLICATION
 TRANSLATE TO APPLICATION

FADELAY

Indicates whether functional acknowledgments are to be immediately queued for sending or only put in the Transaction Store for enveloping later:

Value	Description
Y	Puts functional acknowledgments in the Transaction Store but does not envelope them
N	Puts functional acknowledgments in the Transaction Store and also envelopes them to one of the following (default): <ul style="list-style-type: none"> The file specified by the FUNACKFILE keyword, if present The transaction data queue name from the network profile QDATA, QDATAU, or QDATAE (depending on envelope type) by default

This keyword is used with the following commands:

DEENVELOPE
 DEENVELOPE AND TRANSLATE

RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

FILEID

Specifies the ddname of a file for the following purposes:

- Data is written to file during an ENVELOPE operation
- Data is read from file during a DEENVELOPE operation
- Data is sent from file during a SEND operation
- Data is written to file during a RECEIVE operation

For outbound processing, if this keyword is not supplied, the *Transaction data queue* field in the network profile is used as the ddname for the envelope file.

For inbound processing, if this keyword is not supplied, the *Receive file name* field in the requestor profile is used as the ddname for the envelope file.

The maximum length of the file name is 8 characters.

CAUTION:

On ENVELOPE and DEENVELOPE commands, all transactions are placed in this file. You should also specify the NETID keyword to make sure all transactions you select are for the same network.

Note: In CICS, DataInterchange does not uppercase the FILEID value. FILEID is the name of a temporary storage queue. You should include this keyword to make sure that different applications running in the same CICS region do not envelope transactions to the same TS queue.

This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
RECONSTRUCT
RECONSTRUCT AND SEND
REENVELOPE
REENVELOPE AND SEND
SEND
SENDFILE
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND

FIXEDFILEID

Specifies the ddname of a file to be used for output during Fixed-to-Fixed translation processing. Data is written to the file during an ENVELOPE operation for Fixed-to-Fixed translation.

For outbound processing, if this keyword is not supplied, the ddname, based on the standard ID, is the same as the Application file name within the target ADF definition. There is a file suffix field in the trading partner so that there can be a unique envelope file for each trading partner.

The maximum length of the file name is 8 characters.

CAUTION:

On **ENVELOPE** commands, all transactions are placed in this file. You should also specify the **NETID** keyword to make sure all transactions you select are for the same network.

Note: In CICS, DataInterchange does not uppercase the **FIXEDFILEID** value. **FIXEDFILEID** is the name of a temporary storage queue. You should include this keyword to make sure that different applications running in the same CICS region do not envelope transactions to the same TS queue. This keyword is used with the following commands for Fixed-to-Fixed translation process.

ENVELOPE
 ENVELOPE AND SEND
 REENVELOPE
 REENVELOPE AND SEND
 TRANSLATE AND ENVELOPE
 TRANSLATE AND SEND

FORCETEST Indicates whether the deenvelope and/or translate to application process is to be forced to select only a test usage regardless of the value of the test indicator in the envelope. This keyword is most useful when receiving test envelopes, but the envelopes do not have a test indicator (such as the UCS BG). In this case, the **FORCETEST(Y)** can be used to force the translator to consider the envelopes to be tested and only look for test usages. Valid values are:

Value	Description
Y	Force the process to test mode and select only a test usage if it is defined. If a test usage is not found, an error is generated and the transaction is rejected. If FORCETEST(Y) was used on the Deenvelope process, then it must also be used on the Translate to Application or Retranslate to Application to select those transactions stored by the Deenvelope.
N	Use the test indicator from the envelope to determine the usage to select (default). An envelope without a test indicator is always considered a production envelope.

This keyword is used with the following commands:

DEENVELOPE
 DEENVELOPE AND TRANSLATE
 RECEIVE AND DEENVELOPE
 RECEIVE AND TRANSLATE
 TRANSLATE TO APPLICATION
 RETRANSLATE TO APPLICATION

FORMAT Specifies the ID of the application data format associated with the transaction or transactions you want to select. The maximum length is 16. This keyword is used with the following commands:

ENVELOPE
 ENVELOPE AND SEND
 ENVELOPE DATA EXTRACT
 HOLD
 PRINT ACKNOWLEDGMENT IMAGE
 PRINT ACTIVITY SUMMARY
 PRINT EVENT LOG
 PRINT STATUS SUMMARY

PRINT STATUS SUMMARY2
 PRINT TRANSACTION DETAILS
 PRINT TRANSACTION IMAGE
 PURGE
 QUERY
 REENVELOPE
 REENVELOPE AND SEND
 RELEASE
 REMOVE TRANSACTIONS
 RETRANSLATE TO APPLICATION
 TRANSACTION DATA EXTRACT
 TRANSLATE TO APPLICATION
 UNPURGE

FUNACKFILE Specifies the ddname of the file that you want to use for returning functional acknowledgments for the deenveloped transactions. Use this keyword if you do not want to use the transaction data queue specified in the network profile. The maximum length is 8.

Note: In CICS, DataInterchange does not uppercase the FUNACKFILE value. FUNACKFILE is the name of a temporary storage queue.

This keyword is used with the following commands:

DEENVELOPE
 DEENVELOPE AND TRANSLATE
 RECEIVE AND DEENVELOPE
 RECEIVE AND TRANSLATE

FUNACKP Specifies if a functional acknowledgment is pending. Valid values are:

Value	Description
-------	-------------

Y	Selects transactions for which a functional acknowledgment was requested and not received
N	Selects transactions for which a functional acknowledgment was not requested or has already been received

The maximum length is 1. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
 HOLD
 PRINT ACKNOWLEDGMENT IMAGE
 PRINT ACTIVITY SUMMARY
 PRINT EVENT LOG
 PRINT STATUS SUMMARY
 PRINT STATUS SUMMARY2
 PRINT TRANSACTION DETAILS
 PRINT TRANSACTION IMAGE
 PURGE
 QUERY
 REENVELOPE
 REENVELOPE AND SEND
 RELEASE
 REMOVE TRANSACTIONS
 TRANSACTION DATA EXTRACT
 UNPURGE

FUNACKREQ Specifies that the functional acknowledgment envelope file is required and DataInterchange will produce an error if unable to open. If this specification is desired use the keyword with a value of 'Y'. The default value is 'N'. This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

GROUP Indicates if the group data record value is written to the EDIQUERY file. Valid values are:

Value	Description
Y	Keeps group data record
N	Discards group data record (default)

For the format of these records, see “Group Data Extract Record Layout” on page 2-83. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

GRPCTLNO Specifies the group control number assigned by the sender to identify the functional group to the sender. The maximum length is 14. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE

HANDLE Specifies an ID the system assigns to a transaction when it is placed in the Transaction Store. It is a concatenation of the date, time, and a sequence number to ensure uniqueness, as follows:

YYYYMMDDHHMMSSnnnnnn

You can use this keyword to envelope a specific EDI document or all the documents whose time stamp falls within a given range. The system left-justifies and pads your entries. It pads the FROM value with 0s and the TO value with 9s. You can use an asterisk (*) for the current date. This keyword is used with the following commands:

ENVELOPE

ENVELOPE AND SEND
 ENVELOPE DATA EXTRACT
 HOLD
 PRINT ACKNOWLEDGMENT IMAGE
 PRINT ACTIVITY SUMMARY
 PRINT EVENT LOG
 PRINT STATUS SUMMARY
 PRINT STATUS SUMMARY2
 PRINT TRANSACTION DETAILS
 PRINT TRANSACTION IMAGE
 PURGE
 QUERY
 REENVELOPE
 REENVELOPE AND SEND
 RELEASE
 REMOVE TRANSACTIONS
 RETRANSLATE TO APPLICATION
 TRANSACTION DATA EXTRACT
 TRANSLATE TO APPLICATION
 UNPURGE

HOLDFILE Event log hold file name. The UNLOAD LOG ENTRIES command copies the non-archived event log entries into this file. The LOAD LOG ENTRIES command uses the records in this file to load into the event log. The maximum length is 8. This keyword is used with the following commands:

LOAD LOG ENTRIES
 REMOVE LOG ENTRIES
 UNLOAD LOG ENTRIES

HOLDTYPE Event log hold file type used only for CICS (except MQ, which is supported in both MVS and CICS). Valid values are:

Value	Description
MQ	DataInterchange MQSeries Queue profile member name
TD	Transient data queue
TM	Temporary storage queue (main storage)
TS	Temporary storage queue (auxiliary storage)
VS	VSAM entry sequenced data set

This keyword is used with the following commands:

LOAD LOG ENTRIES
 REMOVE LOG ENTRIES
 UNLOAD LOG ENTRIES

The default in MVS is that the file name is the ddname of the sequential file. The default in CICS is TS.

IACCESS Indicates how the interchange should be presented to the IEXIT program. A value of **M** indicates the interchange should be given to the exit in virtual storage. This option only applies when the ITYPE is UE (user exit). A value of **F** indicates the interchange should be given to the exit in a file. With an IACCESS value of **F**, the interchange is first written to the transaction data queue file and then the IEXIT program is started. This keyword is used with the following commands:

ENVELOPE
 ENVELOPE AND SEND

REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND

IAREA Specifies up to 16 bytes of information, the address of which is provided to the IEXIT program. This only applies to MVS programs. Within CICS, the address provided to the IEXIT program is the address of the utility control block. This keyword is used with the following commands:

ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

ID The name of a DataInterchange profile; for example, REQPROF, TPPROF, or NETPROF. The value must be a valid DataInterchange profile ID and has a maximum length of 8. This keyword is used with the following commands:

QUERY PROFILE
DELETE PROFILE

IEXIT The name of the program to receive control as each interchange is processed. See ITYPE on page 1-91 for your options as to the type of program you can provide. This keyword is used with the following commands:

ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

IMAGE Indicates if the image data record value is written to the EDIQUERY file. Valid values are:

Value	Description
Y	Keeps image data record
N	Discards image data record (default)

Images are always written as separate records. For the format of these records, see "Transaction/Acknowledgment Image Data Extract Record Layout" on page 2-87. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

IFCC	Specifies the condition codes that you wish to override. You can specify up to 10 utility condition codes, separated by commas. These are checked by the DataInterchange Utility and overridden with values from the SETCC keyword on a one-on-one basis. This keyword can be used with any utility PERFORM command. For more information, see “Overriding the Utility Condition Codes” on page 1-6.
INCNTL	<p>The ddname of an optional file that you can use to assist the service in matching the old segments and new segments. You should run the migration once with an OUTCNTL file and no INCNTL file. If changes to the mapping are needed, edit the OUTCNTL file and submit only the changed records as the INCNTL file for the next run of mapping migration. You can use the same file and data definition name (ddname) for INCNTL and OUTCNTL.</p> <p>Note: In CICS, INCNTL is the name of a temporary storage queue, transient data queue, or VSAM entry sequenced data set. You must also provide the INTYPE keyword. This keyword is used with the following command:</p> <p style="padding-left: 40px;">MAPPING MIGRATION</p>
INMEMTRANS	<p>Specifies the number of transactions in memory. This is a value from 1 to 65535 that indicates the number of transactions that should be maintained in storage before the database updates are attempted. This field is valid only if envelope level recovery (RECOVERY(E)) is in effect. Keeping transactions in storage delays the time the database lock is obtained and reduces the time that the database lock is held. The more transactions kept in storage, the higher concurrency rate DataInterchange can achieve. The amount of storage used for each INMEMTRANS is approximately 2K. The default is 100. This keyword is used with the following commands:</p> <p style="padding-left: 40px;"> DEENVELOPE DEENVELOPE AND TRANSLATE ENVELOPE ENVELOPE AND SEND RECEIVE AND DEENVELOPE REENVELOPE REENVELOPE AND SEND TRANSLATE AND ENVELOPE TRANSLATE AND SEND </p>
INTCTLNO	<p>Specifies the interchange control number assigned by the sender to identify the interchange data to the sender. The maximum length is 14. This keyword is used with the following commands:</p> <p style="padding-left: 40px;"> ENVELOPE DATA EXTRACT HOLD PRINT ACKNOWLEDGMENT IMAGE PRINT ACTIVITY SUMMARY PRINT EVENT LOG PRINT STATUS SUMMARY PRINT STATUS SUMMARY2 PRINT TRANSACTION DETAILS PRINT TRANSACTION IMAGE PURGE QUERY RECONSTRUCT RECONSTRUCT AND SEND REENVELOPE REENVELOPE AND SEND </p>

RELEASE
 REMOVE TRANSACTIONS
 RETRANSLATE TO APPLICATION
 TRADING PARTNER PROFILE DATA EXTRACT
 TRANSACTION DATA EXTRACT
 TRANSLATE TO APPLICATION
 UNPURGE

INTERCHANGE

Indicates if the interchange data record value is written to the EDIQUERY file. Valid values are:

Value	Description
Y	Keeps interchange data record
N	Discards interchange data record (default)

For the format of these records, see “Interchange Data Extract Record Layout” on page 2-81. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
 TRANSACTION DATA EXTRACT

INTID

The Interchange Sender/Receiver ID of the trading partner as entered in the Interchange ID field of the Trading Partner Profile. This keyword is used with the following command:

TRADING PARTNER PROFILE DATA EXTRACT

INTRECID

Specifies the interchange receiver ID. The receiver assigns this value to identify the receiver to the sender. The maximum length is 35. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
 HOLD
 PRINT ACKNOWLEDGMENT IMAGE
 PRINT ACTIVITY SUMMARY
 PRINT EVENT LOG
 PRINT STATUS SUMMARY
 PRINT STATUS SUMMARY2
 PRINT TRANSACTION DETAILS
 PRINT TRANSACTION IMAGE
 PURGE
 QUERY
 RECONSTRUCT
 RECONSTRUCT AND SEND
 REENVELOPE
 REENVELOPE AND SEND
 RELEASE
 REMOVE TRANSACTIONS
 RETRANSLATE TO APPLICATION
 TRANSACTION DATA EXTRACT
 TRANSLATE TO APPLICATION
 UNPURGE

INTSNDID

Specifies the interchange sender ID. The sender assigns this value to identify the sender to the receiver. The maximum length is 35. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
 HOLD

PRINT ACKNOWLEDGMENT IMAGE
 PRINT ACTIVITY SUMMARY
 PRINT EVENT LOG
 PRINT STATUS SUMMARY
 PRINT STATUS SUMMARY2
 PRINT TRANSACTION DETAILS
 PRINT TRANSACTION IMAGE
 PURGE
 QUERY
 REENVELOPE
 REENVELOPE AND SEND
 RELEASE
 REMOVE TRANSACTIONS
 RETRANSLATE TO APPLICATION
 TRANSACTION DATA EXTRACT
 TRANSLATE TO APPLICATION
 UNPURGE
 INTYPE

INTYPE The file type of an optional file defined by the INCNTL keyword. Used only for CICS (except MQ, which is supported in both MVS and CICS). Valid values are:

Value	Description
MQ	DataInterchange MQSeries Queue profile member name
TD	Transient data queue
TM	Temporary storage queue (main storage)
TS	Temporary storage queue (auxiliary storage)
VS	VSAM entry sequenced data set

This keyword is used with the following command:

MAPPING MIGRATION

If the keyword is not provided for MVS, this field is ignored; instead, the default for the corresponding file keyword is the ddname of a sequential file. For CICS, the default is TS.

ITPBREAK Determines if a new interchange envelope starts when the internal trading partner ID changes. Valid values are:

Value	Description
Y	Starts new interchange envelope (default)
N	Does not necessarily start a new interchange envelope

This keyword is used with the following commands:

ENVELOPE
 ENVELOPE AND SEND
 REENVELOPE
 REENVELOPE AND SEND
 TRANSLATE AND ENVELOPE
 TRANSLATE AND SEND

ITYPE The type of program specified in IEXIT. The type can be **PG**, indicating that IEXIT is a program that should be linked to (EXEC CICS LINK in CICS), or the type can be **UE**, indicating that IEXIT is a DataInterchange user exit program defined in the ADAMCTL profile. This keyword is used with the following commands:

ENVELOPE

ENVELOPE AND SEND
 REENVELOPE
 REENVELOPE AND SEND
 TRANSLATE AND ENVELOPE
 TRANSLATE AND SEND
 DEENVELOPE
 DEENVELOPE AND TRANSLATE
 RECEIVE AND DEENVELOPE
 RECEIVE AND TRANSLATE

LASTTRXDATE

The date of the last transaction sent to or received from the trading partner. This keyword is used with the following command:

TRADING PARTNER PROFILE DATA EXTRACT

LEVEL

Specifies what information is to be dumped or how much detail is to be traced. The value must be numeric. This keyword is used with the following commands:

GLB DUMP - valid values are:

Value	Description
1	Dump everything (the default)
2	Dump working storage
3	Dump entire dataspace
10	Dump common area
11	Dump record index area
12	Dump data area

GLB TRACE - valid values are:

Value	Description
0	End trace (the default)
1	Start trace functions only
2	Start trace functions and subroutines
3	Start trace functions, subroutines, and subroutine keypoints

LOGAEID

Event log associated entry ID. This keyword can be used to specify a range of associated entry IDs for event log entry selection. To do this, use this keyword along with a corresponding TO keyword. The value specified with the LOGAEID keyword is the beginning value, and the value specified with the TO keyword is the ending value. If a LOGAEID value is specified and there is no corresponding TO value, only entries matching the LOGAEID value will be eligible for selection. If no LOGAEID value is specified, all associated entry IDs are eligible for selection. The maximum length is 40. This keyword is used with the following commands:

LOAD LOG ENTRIES
 REMOVE LOG ENTRIES
 UNLOAD LOG ENTRIES

LOGDATE

Event log date. This keyword can be used to specify a range of dates for event log entry selection. To do this, use this keyword along with a corresponding TO keyword. The value specified with the LOGDATE keyword is the beginning value, and the value specified with the TO keyword is the ending value. If a LOGDATE value is specified and there is no corresponding TO value, only entries matching the LOGDATE value will be eligible for selection. If no LOGDATE value is specified, all dates are eligible for selection. The length is 8. This keyword is used with the following commands:

LOAD LOG ENTRIES

	REMOVE LOG ENTRIES UNLOAD LOG ENTRIES
LOGFILE	Event log file name. This keyword is used in VSAM only to identify the ddname of the event log to be processed. The maximum length is 8. This keyword is used with the following commands:
	LOAD LOG ENTRIES UNLOAD LOG ENTRIES
LOGFORM	Event log format ID. This keyword can be used to specify a range of format IDs for event log entry selection. To do this, use this keyword along with a corresponding TO keyword. The value specified with the LOGFORM keyword is the beginning value, and the value specified with the TO keyword is the ending value. If a LOGFORM value is specified and there is no corresponding TO value, only entries matching the LOGFORM value will be eligible for selection. If no LOGFORM value is specified, all format IDs are eligible for selection. The maximum length is 16. This keyword is used with the following commands:
	LOAD LOG ENTRIES REMOVE LOG ENTRIES UNLOAD LOG ENTRIES
LOGTIME	Event log time. This keyword can be used to specify a range of times for event log entry selection. To do this, use this keyword along with a corresponding TO keyword. The value specified with the LOGTIME keyword is the beginning value, and the value specified with the TO keyword is the ending value. If a LOGTIME value is specified and there is no corresponding TO value, only entries matching the LOGTIME value will be eligible for selection. If no LOGTIME value is specified, all times are eligible for selection. The length is 6. This keyword is used with the following commands:
	LOAD LOG ENTRIES REMOVE LOG ENTRIES UNLOAD LOG ENTRIES
LOGUSER	Event log user ID. This keyword can be used to specify a range of user IDs for event log entry selection. To do this, use this keyword along with a corresponding TO keyword. The value specified with the LOGUSER keyword is the beginning value, and the value specified with the TO keyword is the ending value. If a LOGUSER value is specified and there is no corresponding TO value, only entries matching the LOGUSER value will be eligible for selection. If no LOGUSER value is specified, all user IDs are eligible for selection. The maximum length is 8. This keyword is used with the following commands:
	LOAD LOG ENTRIES REMOVE LOG ENTRIES UNLOAD LOG ENTRIES
MAPFROM	The name of the trading partner transaction (TPT) to copy. Must be an existing TPT ID for which the user has authority. This keyword is used with the following command:
	MAPPING MIGRATION
MAPTO	The name of the trading partner transaction (TPT) to create. Must be a valid TPT ID for which the user has authority. If omitted, refresh the MAPFROM TPT. This keyword is used with the following command:
	MAPPING MIGRATION

MAXRUNTIME Specifies the maximum time in minutes the Remove Transaction process is allowed to run. When the specified time is exceeded, the Remove Transaction process is stopped even if it has not completed. This keyword and value are useful when the Remove Transaction process is to run standalone, but can only run in this mode for a limited time.

MAXRUNTIME makes sure the Remove Transaction process does not exceed a given time frame in which other DataInterchange process are not allowed to run. The default value is 0 indicating there is no maximum run time. This keyword is used with the following command:

REMOVE TRANSACTIONS

MEMBER The name of a DataInterchange profile member. The maximum length is dependent upon the type of profile, and cannot exceed 35. This keyword is used with the following commands:

QUERY PROFILE
DELETE PROFILE
START CONTINUOUS RECEIVE
STOP CONTINUOUS RECEIVE
REPORT CONTINUOUS RECEIVE STATUS

MERGED A value of 'Y' indicates that the transaction image is to be printed with each segment beginning a new line and merged with the functional acknowledgment image. This keyword is used with the following command:

PRINT TRANSACTION IMAGE

MRREQID A requestor ID that associates management reporting statistics with DEENVELOPE processing instead of RECEIVE processing. This keyword is used if you receive interchanges without using DataInterchange and you want to keep management reporting receive statistics on those interchanges. Do not use this keyword if management reporting statistics were created during the receive of the interchange. Creating management reporting statistics during receive processing is the standard method. Use MRREQID keyword only in exceptional situations. This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE

MSGUCLASS An override message user class for a send or receive type command. Message user class is a code that you and your trading partner agree to use for identifying the class of information to be sent or received. If this keyword is used, the value overrides the value specified in the Requestor profile member supplied by the REQID keyword. The maximum length is 8.

This keyword is used in the following commands:

ENVELOPE AND SEND
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE AND SEND
RECONSTRUCT AND SEND
SEND
SENDFILE
TRANSLATE AND SEND

NETACKP Specifies if a network acknowledgment is pending. Valid values are:

Value	Description
-------	-------------

- | | |
|---|---|
| Y | Selects transactions for which a network acknowledgment was requested but not received. |
| N | Selects transactions for which a network acknowledgment is not pending or was not requested. The maximum length is 1. |

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
TRANSACTION DATA EXTRACT
UNPURGE

NETID Specifies the name used to identify a network in the network profile. The maximum length is 8. This keyword is used with the following commands:

ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
NETWORK ACTIVITY DATA EXTRACT
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
UPDATE STATUS

NETNAME The name of the network as entered in the *Network name* field of the Network Profile. This keyword is used with the following command:

NETWORK ACTIVITY DATA EXTRACT

NETSTAT The network status of a transaction for which a send has been requested. The maximum length of this value is 2. Valid values are:

Value	Description
30	Enveloped
31	Envelope error
41	Sent with errors
42	Send request error
43	Not sent net error
46	Send started
48	Send requested
49	Sent to network
50	Accepted by network
51	Delivered by network
52	Purged by network
53	Recall requested
54	Recall request error
55	Recalled

For complete status information, see the *DataInterchange Administrator's Guide*. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
TRANSACTION DATA EXTRACT
UNPURGE

NEWAPPLID New application ID. With this keyword, it is possible to change the application ID when loading event log entries into an event log. The maximum length is 8. This keyword is used with the following command:

LOAD LOG ENTRIES

NEWMSG The name of the new transaction or message to which to migrate the trading partner transaction. Must be an existing transaction or message to which the user has authority. If not given, the service uses the same message or transaction from the MAPFROM. This keyword has the same meaning as NEWTRX. Use only one of the two (NEWTRX

or NEWMSG), as they both have identical meanings. This keyword is used with the following command:

MAPPING MIGRATION

NEWSTD The standard to which to migrate the trading partner transaction. Must be an existing standard ID to which the user has authority. If not given, the service will use the same standard ID from the MAPFROM. This keyword is used with the following command:

MAPPING MIGRATION

NEWTRX The name of the new transaction or message to which to migrate the trading partner transaction. Must be an existing transaction or message to which the user has authority. If not given, the service will use the same message or transaction from the MAPFROM. This keyword has the same meaning as NEWMSG. Use only one of the two (NEWTRX or NEWMSG), as they both have identical meanings. This keyword is used with the following command:

MAPPING MIGRATION

NOMSG Specifies whether or not extraneous messages should appear in the print file. This keyword is optional and if not specified, the default is N. This keyword is only applicable with the EXPORT and the IMPORT commands and is usually only used when large numbers of records are being exported or imported. For example, it may be used with export-all (exporting after building a control file using program EDIXPEA) or import-all (using a control file with category 9). A value of Y suppresses the following messages from appearing in the print file:

EI0048
EI0049
EI0062

and extraneous messages:

FF0512
FF0514

A value of Y will also suppress certain error conditions. For example, if in your command file you specify that you want to export CONTRECV members, and a CONTRECV profile does not exist on your system, this error would be ignored. Also, if you specify certain associated objects that do not exist, this error would also be ignored. This keyword is used with the following commands:

EXPORT
IMPORT

NUMDELS Specifies the number of database deletes before DataInterchange issues a database commit. DB2 includes a site dependent value (NUMLKUS) that controls the maximum number of page locks one Remove Transaction process can hold. When the Remove Transaction process is not running standalone, page locks are obtained. If the NUMDELS value is too high, DB2 can stop the Remove Transactions process because NUMLKUS has been exceeded. If this happens, lower the NUMDELS value so that DataInterchange issues more frequent database commits and frees up page locks. The default is 100 and the maximum is 1000. If a value greater than 1000 is entered, the default of 100 will be assumed. This keyword is used with the following commands:

REMOVE LOG ENTRIES
REMOVE STATISTICS
REMOVE TRANSACTIONS
UNLOAD LOG ENTRIES

Note: Using the VSAM version of DataInterchange, NUMDELS is used to specify the actual number of deletes during REMOVE STATISTICS. If more than 100 records are to be deleted during REMOVE STATISTICS, NUMDELS should be used.

NUMUPDTS Specifies the number of database updates before DataInterchange issues a database commit. If you are experiencing timeouts using a command that uses this parameter, you can reduce the amount of time that the command holds locks by forcing it to COMMIT more frequently. The default for NUMUPDTS is 50. If you want more frequent COMMITS, specify a number smaller than 50. If you are trying to increase the performance of a command that uses this parameter, you can specify a value larger than 50. This forces DataInterchange to COMMIT less frequently and should speed up the processing. This keyword is used with the following command:

UPDATE STATISTICS

ONELOGICAPP

Determines if the APPFILE in the current WHERE clause and all APPFILES in proceeding WHERE clauses are considered one logical file. When chosen, envelope breaks are avoided between APPFILE processing. This can be used to avoid APPFILE switching that causes envelope breaks when processing multiple raw data files. This option allows multiple raw data files to be placed in the same envelope without delaying enveloping. If the FILEID keyword is specified in combination with the ONELOGICAPP keyword in multiple WHERE clauses, the value in the first FILEID found is used to envelope the data. This option is also valid for C and D record formats. Valid values are:

Value	Description
--------------	--------------------

Y	Treats multiple application files as one logical file.
N	Each file is processed independently and envelope breaks occur at the end of each application file (default).

This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
TRANSLATE AND SEND

OPTRECS Specifies which optional records are to be created during translation. For TRANSLATE TO STANDARD (delayed enveloping), the only valid value is I (for information records). Q is not valid for TRANSLATE or RETRANSLATE TO APPLICATION. The following values are valid in any combination:

Value	Record
--------------	---------------

I	Information
E	Envelope header
G	Group header
T	Transaction header
Q	Queuing

For the format and location of these records, see "Optional Records" on page 2-72. The maximum length is 5. This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE
RETRANSLATE TO APPLICATION

TRANSLATE AND ENVELOPE
 TRANSLATE AND SEND
 TRANSLATE TO APPLICATION
 TRANSLATE TO STANDARD

OUTCNTL The ddname of the control file to which mapping migration control records are written. These records can be viewed to understand how the migration was performed. Next, the records can be edited and the migration rerun to correct any problems found. You can use the same file and data definition name (ddname) for INCNTL and OUTCNTL.

Note: In CICS, OUTCNTL is the name of a temporary storage queue, transient data queue, or VSAM entry sequenced data set. You must also provide the OUTTYPE keyword. This keyword is used with the following command:

MAPPING MIGRATION

OUTFILE The ddname of an output file (or the name of a temporary storage queue, transient data queue, or VSAM entry sequenced dataset in CICS). The maximum length is 8. This keyword is used with the following commands:

MAPPING MIGRATION
 QUERY PROFILE
 SAP STATUS EXTRACT

OUTFORMAT The format the output is to be written in. The maximum length is 1. Valid values are:

Value	Description
F	Fixed
T	Tagged
N	Native

This keyword is used with the following command:

QUERY PROFILE

OUTTYPE The file type of OUTCNTL or OUTFILE used only in CICS (except MQ, which is supported in both MVS and CICS). Valid values are:

Value	Description
MQ	DataInterchange MQSeries Queue profile member name
TD	Transient data queue
TM	Temporary storage queue (main storage)
TS	Temporary storage queue (auxiliary storage)
VS	VSAM entry sequenced data set

This keyword is used with the following commands:

MAPPING MIGRATION
 QUERY PROFILE
 SAP STATUS EXTRACT

If the keyword is not provided for MVS, this field is ignored; instead, the default for the corresponding file keyword is the ddname of a sequential file. For CICS, the default is TS.

PAGE Enables Pageable Translation. Pageable Translation is designed to better utilize memory during translation. It does this by paging EDI and application data buffers to DASD once the allotted number of internal buffers has been exhausted. The buffer size is 28,632 bytes, and the allotted number of internal buffers before Pageable Translation begins is 1,000. This means that approximately 28 MB of virtual storage can be used to hold EDI and application data before Pageable Translation is triggered.

To use Pageable Translation in MVS, a temporary work file must be defined with ddname EDIVAX. The amount of space allocated to this file is dependent on the maximum amount of data to be translated.

The amount of storage required to translate an envelope without Pageable Translation can be calculated by adding the following components:

- Number of bytes of largest interchange +
- Number of bytes of largest application transaction image +
- 4 MB overhead +
- Number of structures in largest interchange x 120 bytes

Pageable Translation deals with the first two components, and ensures that the amount of virtual storage required for them does not exceed 28 MB. The other components are not addressed by Pageable Translation.

The maximum amount of data that DataInterchange can page with Pageable Translation is approximately one gigabyte (specifically, 32,000 multiplied by 28,632 bytes).

Pageable Translation in CICS uses temporary storage queues with names that begin with EDI. Therefore, the size of DFHTEMP may have to be considered if Pageable Translation is desired in CICS.

A sample of how EDIVAX might be defined follows:

```
//EDIFAX DD DISP=(NEW,DELETE,DELETE),UNIT=SYSDA,SPACE=(CYL,25)
```

Value	Description
-------	-------------

Y	Enables Pageable Translation
---	------------------------------

N	Disables Pageable Translation
---	-------------------------------

This keyword is used with the following commands:

- DEENVELOPE
- DEENVELOPE AND TRANSLATE
- ENVELOPE
- ENVELOPE AND SEND
- RECEIVE AND DEENVELOPE
- RECEIVE AND TRANSLATE
- RECONSTRUCT AND SEND
- REENVELOPE
- REENVELOPE AND SEND
- RETRANSLATE TO APPLICATION
- TRANSLATE AND ENVELOPE
- TRANSLATE AND SEND
- TRANSLATE TO APPLICATION
- TRANSLATE TO STANDARD

PRIORTO

A date before which all statistics will be deleted. This keyword is used with the following commands:

- REMOVE STATISTICS
- SAP STATUS REMOVE

PURGINT

Specifies the number of days a transaction is in the Transaction Store before the transaction is marked for purging. If this keyword and value are not supplied, or if a value of 0 is supplied, the DataInterchange Utility uses 30 days by default. The maximum is 9999. You can use a negative value to indicate that a transaction's store time expired on a date in the past. Using PURGINT(-2), for example, sets the purge date at two days ago. The minimum is -999. If a functional or network acknowledgment is pending when

the store time expires, the transaction retains its current store status until the acknowledgment is no longer pending. This keyword is used with the following commands:

DEENVELOPE
 DEENVELOPE AND TRANSLATE
 RECEIVE AND DEENVELOPE
 RECEIVE AND TRANSLATE
 RETRANSLATE TO APPLICATION
 TRANSLATE AND ENVELOPE
 TRANSLATE AND SEND
 TRANSLATE TO APPLICATION
 TRANSLATE TO STANDARD

RAWDATA Indicates if you want the standard data translated to raw format or to C and D records:

Value	Description
Y	Raw data format
N	C and D record format

For details about these record formats, see “Application File” on page 2-3. This keyword is used with the following commands:

DEENVELOPE AND TRANSLATE
 RECEIVE AND TRANSLATE
 RETRANSLATE TO APPLICATION
 TRANSLATE TO APPLICATION

For Fixed-to-Fixed mappings, this keyword indicates if the output data should be written in the RAWDATA format or in the C and D record format. RAWDATA(Y) would indicate the RAWDATA format is wanted. This keyword has this meaning with the following commands:

TRANSLATE TO STANDARD
 TRANSLATE AND ENVELOPE
 TRANSLATE AND SEND
 ENVELOPE
 ENVELOPE AND SEND
 RECONSTRUCT
 RECONSTRUCT AND SEND
 REENVELOPE
 REENVELOPE AND SEND

See “DataInterchange Utility Records Format” on page 2-61 for a description of C and D records and the RAWDATA format.

See “Envelope File” on page 2-3 for an explanation of the file to which data will be written.

RAWFMTID Specifies the APPFILE data format ID. Use this keyword only if the data in APPFILE is in a raw data format. The maximum length is 16. This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
 TRANSLATE AND SEND
 TRANSLATE TO STANDARD

RAWTEST Valid with raw data only. Use one of the following values to tell the receiver the transaction is for testing only:

Note: This keyword is being replaced by RAWUSAGE. The RAWUSAGE value

overrides RAWTEST if both values are provided. RAWTEST is provided for transitional purposes only, and will be removed in a future release.

Value	Description
-------	-------------

Y	Specifies test transactions. All of the transactions are test transactions, and a test usage should be used if it exists. If a test usage does not exist, the production usage should be used. Same as RAWUSAGE(T).
---	---

N	Specifies production transactions (default). All the transactions are production transactions and only the production usage should be used. Same as RAWUSAGE(P).
---	--

U	Specifies test or production transactions. DataInterchange determines the status of the transaction based on the presence of an active test usage. If an active test usage exists, the transaction is considered a test transaction. If an active test usage does not exist, the transaction is considered a production transaction. Same as RAWUSAGE(U).
---	---

The translator uses this value to set the test indicator in the interchange header (0035 for EDIFACT, I14 for X12). For C and D records (as opposed to raw data), the TESTIND field of the control record serves the same purpose as this keyword.

This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD

RAWUSAGE Valid with raw data only. Use one of the following values to tell the receiver the transaction is for production, testing, or information:

Value	Description
-------	-------------

P	Specifies production transactions (default). All of the transactions are production transactions and only the production usage should be used.
---	--

T	Specifies test transactions. All of the transactions are test transactions, and a test usage should be used if it exists. If a test usage does not exist, the production usage should be used.
---	--

I	Specifies an information transaction. An information usage is used if one exists. If one does not exist, a production usage will be used if it exists. If a production usage does not exist, an error will occur.
---	---

U	Specifies test or production transactions. DataInterchange determines the status of the transaction based on the presence of an active test usage. If an active test usage exists, the transaction is considered a test transaction. If an active test usage does not exist, the transaction is considered a production transaction.
---	--

The translator uses this value to set the test indicator in the interchange header (0035 for EDIFACT, I14 for X12). For C and D records (as opposed to raw data), the TESTIND field of the control record serves the same purpose as this keyword.

This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD

RECEIVEACKDATA

Indicates if detailed acknowledgment data is written to the EDIQUERY file. Valid values are:

Value	Description
Y	Keeps detailed acknowledgment data
N	Discards detailed acknowledgment data (default)

Detailed acknowledgment data consists of the interchange, group, and transaction data for the acknowledgment transaction. This information is concatenated with the group and transaction records to which it applies. For the format of these records, see “Transaction/Acknowledgment Image Data Extract Record Layout” on page 2-87. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

RECEIVEACKIMAGE

Indicates if the receive acknowledgment is written to the EDIQUERY file. Valid values are:

Value	Description
Y	Keeps receive acknowledgment
N	Discards receive acknowledgment (default)

Images are always written as separate records. For the format of these records, see “Transaction/Acknowledgment Image Data Extract Record Layout” on page 2-87. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

RECOVBAD

Indicates whether or not the translation process will try to recover from a bad standard data flag.

Value	Description
Y	Indicates the translation process will try to recover from bad standard data. The translator will check for standard interchange headers when a segment terminator is not found on a particular standard segment. The translator will attempt to reset the delimiters and check the current segment/record for the segment terminator. This is the default.
N	Indicates the translation process will not try to recover.

This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

RECOVERY

Specifies unit of work. Valid values are:

Value	Description
E	Specifies entire envelope should be processed before the database commit is done
T	Specifies database commit is issued after each transaction

The default is environmentally dependent; in MVS the default is T and in CICS the default is E.

Note: The unit of work keyword is ignored in the TSO VSAM environment.

This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND

REPLACE Y or N flag to confirm that the MAPTO trading partner transaction (or MAPFROM if the MAPTO is omitted) should be replaced by the migrated trading partner transaction. The default is N. This keyword is used with the following command:

MAPPING MIGRATION

REQID Specifies the requestor ID as specified in the requestor profile. A WHERE clause can contain only one requestor ID. For multiple requestors, use additional WHERE clauses. A send is issued for each REQID for which data was queued. The maximum length is 16. This keyword is used with the following commands:

CLOSE MAILBOX
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE AND SEND
NETWORK ACTIVITY DATA EXTRACT
PROCESS NETWORK ACKS
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE AND SEND
SEND
SENDFILE
TRANSLATE AND SEND
UPDATE STATUS

REQTP Specifies the trading partner associated with a given requestor for use with direct connection networks such as Point-to-Point. This keyword is used to eliminate the need for a separate requestor for each trading partner. This keyword is used with the following command:

ENVELOPE AND SEND

RESET Specifies whether the output file is to be reset. A value of 'Y' indicates that the file is to be reset. Otherwise, the file is appended to. The maximum length is 1. This keyword is used with the following commands:

GLB DUMP
GLB TRACE

SAPSTAT The SAP status value to extract or used with TO(), for a range of values to extract. The acceptable values are 04–22. The default is all status. This keyword is used with the following commands:

SAP STATUS EXTRACT
SAP STATUS REMOVE

SAPUPDT Specifies SAP status tracking is desired. Valid values are:

Value	Description
Y	SAP status tracking, records are written
N	No SAP status tracking (default), no records written

This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND
SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD

SCRIPT Specifies the value that can be used by communication software to identify a set of instructions to follow when processing requests for service. This set of instructions would be part of the communication software package and not part of DataInterchange. For an example of this, see “Interfacing DataInterchange for CICS with SDM LinkPlus Interactive” on page 6-34. This keyword is optional. The maximum field length is 8.

This keyword is used with the following commands:

ENVELOPE AND SEND
REENVELOPE AND SEND
SEND
SENDFILE
TRANSLATE AND SEND
RECONSTRUCT AND SEND
RESTART SEND

SEGMENTED A value of ‘Y’ indicates that the image is to be printed with each segment beginning a new line. This keyword is used with the following commands:

PRINT ACKNOWLEDGMENT IMAGE
PRINT TRANSACTION IMAGE

SENDACKDATA

Indicates if detailed acknowledgment data is written to the EDIQUERY file. Valid values are:

Value	Description
Y	Keeps detailed acknowledgment data
N	Discards detailed acknowledgment data (default)

Detailed acknowledgment data consist of the interchange, group, and transaction data for the acknowledgment transaction. This information is concatenated with the group and transaction record to which it applies. SENDACKDATA is ignored unless GROUP or TRANSACTION is set to Y. For the format of these records, see “Transaction/Acknowledgment Image Data Extract Record Layout” on page 2-87. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

SENDACKIMAGE

Indicates if the generated acknowledgment record is written to the EDIQUERY file. Valid values are:

Value	Description
Y	Keeps generated acknowledgment record
N	Discards generated acknowledgment record (default)

Images are always written as separate records. For the format of these records, see “Transaction/Acknowledgment Image Data Extract Record Layout” on page 2-87. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

SERVICESEGVAL

A flag that indicates which level of service segment validation should take place. The service segments are the segments used when a transaction is enveloped (ISA, GS, ST, UNB, UNH, UNT, and so on). No validation will occur unless SERVICESEGVAL has one of the values below.

Value	Description
1	Indicates the service segments should be validated for SYNTAX only. This includes checking for mandatory data that is missing, as well as data elements that are too large or too small.
2	Indicates that, in addition to level 1 checking, the values in the service segment data elements should be validated according to their types (only dates and times are validated), and if a validation table has been specified, the value of the data element will be checked.

Validation errors during SEND processing are considered terminating errors. Validation errors during RECEIVE processing result in the INTERCHANGE/GROUP/TRANSACTION with the error being skipped (not processed).

This keyword is used in the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND

SETCC

Specifies the condition codes used by the DataInterchange Utility to override the utility condition codes specified in the IFCC keyword. You can have up to 10 override condition codes separated by commas. If this keyword is omitted, all condition codes specified on the IFCC keyword are overridden to zero (0). If a particular code is omitted in the SETCC keyword, the related condition code in the IFCC keyword is overridden to zero (0). For more information, see “Overriding the Utility Condition Codes” on page 1-6. This keyword can be used with any utility PERFORM command.

SNDDATE

Specifies the date of the previous send request for the transaction. The maximum length is 10. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT

HOLD
 PRINT ACKNOWLEDGMENT IMAGE
 PRINT ACTIVITY SUMMARY
 PRINT EVENT LOG
 PRINT STATUS SUMMARY
 PRINT STATUS SUMMARY2
 PRINT TRANSACTION DETAILS
 PRINT TRANSACTION IMAGE
 PURGE
 QUERY
 REENVELOPE
 REENVELOPE AND SEND
 RELEASE
 REMOVE TRANSACTIONS
 TRANSACTION DATA EXTRACT
 UNPURGE

SNDDTIME Specifies the time of the previous send request for the transaction. The maximum length is 8. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
 HOLD
 PRINT ACKNOWLEDGMENT IMAGE
 PRINT ACTIVITY SUMMARY
 PRINT EVENT LOG
 PRINT STATUS SUMMARY
 PRINT STATUS SUMMARY2
 PRINT TRANSACTION DETAILS
 PRINT TRANSACTION IMAGE
 PURGE
 QUERY
 REENVELOPE
 REENVELOPE AND SEND
 RELEASE
 REMOVE TRANSACTIONS
 TRANSACTION DATA EXTRACT
 UNPURGE

STANDALONE Increases the performance of the Remove Transactions process. Valid values are:

Value	Description
Y	Specifies Remove Transactions process does not contend with any other DataInterchange process. In the DB2 environment, this allows DataInterchange to obtain exclusive high level table locks. This increases the performance of the Remove Transactions process since DataInterchange does not have to obtain lower level page locks. Run the Remove Transaction process in this mode if possible.
N	Specifies Remove Transactions process is contending with other DataInterchange processing. Therefore, high-level table locks should not be obtained (default).

This keyword is used with the following command:

REMOVE TRANSACTIONS

STDDESC	The description of the standard as entered into the standards database. This keyword is used with the following commands: TRADING PARTNER CAPABILITY DATA EXTRACT TRANSACTION ACTIVITY DATA EXTRACT										
STDID	The Standard ID that describes the standard in the standards database. This keyword is used with the following commands: TRADING PARTNER CAPABILITY DATA EXTRACT TRANSACTION ACTIVITY DATA EXTRACT										
STDLV	The level of the standard, for example, R1 for release 1. This keyword is used with the following commands: TRADING PARTNER CAPABILITY DATA EXTRACT TRANSACTION ACTIVITY DATA EXTRACT										
STDTRID	Specifies the standard transaction ID as specified by the standard, such as 850 for an X12 purchase order. The maximum length is 8. This keyword is used with the following commands: ENVELOPE ENVELOPE AND SEND ENVELOPE DATA EXTRACT HOLD PRINT ACKNOWLEDGMENT IMAGE PRINT ACTIVITY SUMMARY PRINT EVENT LOG PRINT STATUS SUMMARY PRINT STATUS SUMMARY2 PRINT TRANSACTION DETAILS PRINT TRANSACTION IMAGE PURGE QUERY REENVELOPE REENVELOPE AND SEND RELEASE REMOVE TRANSACTIONS RETRANSLATE TO APPLICATION TRADING PARTNER CAPABILITY DATA EXTRACT TRANSACTION ACTIVITY DATA EXTRACT TRANSACTION DATA EXTRACT TRANSLATE TO APPLICATION UNPURGE										
STDVR	The version of the standard, for example, V3 for version 3. This keyword is used with the following commands: TRADING PARTNER CAPABILITY DATA EXTRACT TRANSACTION ACTIVITY DATA EXTRACT										
STSTAT	Specifies the status of the transaction in the Transaction Store. Valid values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>0</td><td>Active</td></tr> <tr> <td>1</td><td>Held</td></tr> <tr> <td>3</td><td>Purge-date expired</td></tr> <tr> <td>4</td><td>Purge-user request</td></tr> </table>	Value	Description	0	Active	1	Held	3	Purge-date expired	4	Purge-user request
Value	Description										
0	Active										
1	Held										
3	Purge-date expired										
4	Purge-user request										

For complete status information, see the *DataInterchange Administrator's Guide*. This keyword is used with the following commands:

- ENVELOPE DATA EXTRACT
- HOLD
- PRINT ACKNOWLEDGMENT IMAGE
- PRINT ACTIVITY SUMMARY
- PRINT EVENT LOG
- PRINT STATUS SUMMARY
- PRINT STATUS SUMMARY2
- PRINT TRANSACTION DETAILS
- PRINT TRANSACTION IMAGE
- PURGE
- QUERY
- RELEASE
- REMOVE TRANSACTIONS
- TRANSACTION DATA EXTRACT
- UNPURGE

TESTMODE A flag indicating whether the transactions referred to are test transactions or production transactions. Valid values are Y for test and information transactions and N for production transactions. This keyword is used with the following commands:

- TRADING PARTNER CAPABILITY DATA EXTRACT
- TRANSACTION ACTIVITY DATA EXTRACT

TPID Specifies the internal trading partner ID used by an application to identify a trading partner. TPID is specified in the trading partner send usage. The maximum length is 35. This keyword is used with the following commands:

- ENVELOPE
- ENVELOPE AND SEND
- HOLD
- PRINT ACKNOWLEDGMENT IMAGE
- PRINT ACTIVITY SUMMARY
- PRINT EVENT LOG
- PRINT STATUS SUMMARY
- PRINT STATUS SUMMARY2
- PRINT TRANSACTION DETAILS
- PRINT TRANSACTION IMAGE
- PURGE
- QUERY
- REENVELOPE
- REENVELOPE AND SEND
- RELEASE
- REMOVE TRANSACTIONS
- RETRANSLATE TO APPLICATION
- TRADING PARTNER CAPABILITY DATA EXTRACT
- TRANSACTION ACTIVITY DATA EXTRACT
- TRANSLATE TO APPLICATION
- UNPURGE

Provides the default internal trading partner ID value, if the application data format does not define a field that contains this value or if a field is defined but the value is all blanks. This also becomes the default ID value if the internal trading partner ID is blank in the 'C' record. This keyword has this meaning with the following commands:

TRANSLATE TO STANDARD
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND

TPNICKN

Specifies the trading partner nickname. TPNICKN is a trading partner profile member. The maximum length is 16. TPNICKN has two distinct uses:

- TPNICKN specifies a trading partner nickname to base Transaction Store or management reporting selection upon, or a specific trading partner to receive data from.
- For the SEND and TRANSLATE AND SEND commands, its use is the same as the keyword TPNICKNESEND. This means that the fields in the specified trading partner profile member are used for network override options. These fields override the same fields supplied within the requestor profile. It is important to understand that this keyword (when used with the SEND and TRANSLATE AND SEND commands) does not specify the receiver's mailbox. When sending EDI standard data, the receiver's ID is contained within the data being sent. This field is used for overriding network options (such as network charges).

This keyword is used with the following commands:

ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
RECONSTRUCT
RECONSTRUCT AND SEND
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
SEND
SENDFILE
TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
TRANSACTION DATA EXTRACT
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
UNPURGE

TPNICKNESEND

Specifies a trading partner profile member to use for network override options. The fields in the trading partner profile override the same fields supplied in the requestor profile. It is important to understand that this keyword does not specify the receiver's mailbox. When sending EDI standard data, the receiver's ID is contained within the data being sent. This field is used for overriding network options (such as network charges). This keyword is used with the following commands:

ENVELOPE AND SEND
REENVELOPE AND SEND

TPTID

The ID of the Trading Partner Transaction being referred to. This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

TRANSACTION

Indicates if the transaction data record value is written to the EDIQUERY file. Valid values are:

Value	Description
Y	Keeps transaction data record
N	Discards transaction data record (default)

For the format of these records, see "Transaction/Acknowledgment Image Data Extract Record Layout" on page 2-87. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

TRERLVL

Specifies the maximum translation error level for the transactions you want to select. Use this keyword, for example, to envelope only EDI documents that are error-free. The maximum length is 1. Valid values are:

Value	Description
0	No errors
1	Data element errors
2	Data element and segment errors
3	Severe errors

This keyword is used with the following commands:

ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE

REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
UNPURGE

TRXCTLNO Specifies the transaction set control number assigned by the sender to identify the transaction set to the sender. When combined with the sender ID, it identifies the transaction set to the receiver. The maximum length is 14. This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE

TRXDATE Specifies the date the transaction was added to the Transaction Store. This is also the date on which the transaction was translated to standard format. The maximum length is 10. This keyword is used with the following commands:

ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE

TRXSTAT

Specifies the processing status value.

- For enveloping and sending, the default value is 21. Valid value is:

Value	Description
21	Send translated

- For reenveloping, the default values are 31, 41, 42, and 43. Valid values are:

Value	Description
29	Trx detached — send
30	Enveloped
31	Envelope error
41	Sent with errors
42	Send request error
43	Not sent net error
46	Send started status
48	Send requested
49	Sent to network
50	Accepted by network
51	Delivered by network
52	Purged by network
53	Recall requested
54	Recall request error
55	Recalled
61	Transaction accepted
62	Transaction rejected
63	Transaction accepted with errors

- For TRANSLATE, the default value is 70. For RETRANSLATE TO APPLICATION, the default value is 73. Valid values for both are:

Value	Description
70	Received
72	Receive translated (RETRANSLATE TO APPLICATION only)
73	Receive translate error (RETRANSLATE TO APPLICATION only)

- For UNPURGE TRANSACTIONS and REMOVE TRANSACTIONS, if no value is specified, all values are included in the selection criteria. Valid values are:

Value	Description
20	Send translate error
21	Send translated
29	Trx detached — send
30	Enveloped
31	Envelope error
41	Sent with errors
42	Send request error
43	Not sent net error
46	Send started status
48	Send requested
49	Sent to network
50	Accepted by network
51	Delivered by network
52	Purged by network
53	Recall requested
54	Recall request error
55	Recalled

61	Transaction accepted
62	Transaction rejected
63	Transaction accepted with errors
70	Received
71	Receive syntax error
72	Receive translated
73	Receive trans error
74	Trx detached — recv

This keyword is used with the following commands:

- ENVELOPE
- ENVELOPE AND SEND
- ENVELOPE DATA EXTRACT
- HOLD
- PRINT ACKNOWLEDGMENT IMAGE
- PRINT ACTIVITY SUMMARY
- PRINT EVENT LOG
- PRINT STATUS SUMMARY
- PRINT STATUS SUMMARY2
- PRINT TRANSACTION DETAILS
- PRINT TRANSACTION IMAGE
- PURGE
- QUERY
- REENVELOPE
- REENVELOPE AND SEND
- RELEASE
- REMOVE TRANSACTIONS
- RETRANSLATE TO APPLICATION
- TRANSACTION DATA EXTRACT
- TRANSLATE TO APPLICATION
- UNPURGE

TRXTIME

Specifies the time the transaction was added to the Transaction Store. This is also the time at which the transaction was translated to standard format. The maximum length is 8. This keyword is used with the following commands:

- ENVELOPE
- ENVELOPE AND SEND
- ENVELOPE DATA EXTRACT
- HOLD
- PRINT ACKNOWLEDGMENT IMAGE
- PRINT ACTIVITY SUMMARY
- PRINT EVENT LOG
- PRINT STATUS SUMMARY
- PRINT STATUS SUMMARY2
- PRINT TRANSACTION DETAILS
- PRINT TRANSACTION IMAGE
- PURGE
- QUERY
- REENVELOPE
- REENVELOPE AND SEND
- RELEASE
- REMOVE TRANSACTIONS
- RETRANSLATE TO APPLICATION
- TRANSACTION DATA EXTRACT

	TRANSLATE TO APPLICATION UNPURGE						
USERID	The Network User ID of the trading partner as entered in the User ID field of the Trading Partner Profile. This keyword is used with the following commands: NETWORK ACTIVITY DATA EXTRACT TRADING PARTNER PROFILE DATA EXTRACT						
USERPGM	Specifies a program that DataInterchange links to just before writing a record during DATA EXTRACT processing. Your program should return a code to DataInterchange indicating whether the record is to be written to the EDIQUERY file or discarded. If this program is not supplied, DataInterchange writes the requested records to the EDIQUERY file. For more information, see Chapter 4, "Exit Routines." This keyword is used with the following commands: ENVELOPE DATA EXTRACT NETWORK ACTIVITY DATA EXTRACT TRANSACTION DATA EXTRACT TRADING PARTNER CAPABILITY DATA EXTRACT TRADING PARTNER PROFILE DATA EXTRACT TRANSACTION ACTIVITY DATA EXTRACT						
VERIFY	Specifies the transaction status verification. Determines if the status of a transaction should be verified before the transaction is put into an envelope. Valid values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Status must be correct for an ENVELOPE or REENVELOPE operation</td></tr> <tr> <td>(other)</td><td>Status should not be checked (default)</td></tr> </table> <p>This keyword is used with the following commands:</p> <p>ENVELOPE ENVELOPE AND SEND REENVELOPE REENVELOPE AND SEND</p>	Value	Description	Y	Status must be correct for an ENVELOPE or REENVELOPE operation	(other)	Status should not be checked (default)
Value	Description						
Y	Status must be correct for an ENVELOPE or REENVELOPE operation						
(other)	Status should not be checked (default)						
WRTCTLNO	Specifies writing of trading partner control number values when importing a trading partner profile member. This keyword is optional and if not specified, the default is Y. Valid values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Import control numbers with the trading partner.</td></tr> <tr> <td>N</td><td>Do not import control numbers. If this is a new trading partner, the control number will default to zeros. If this trading partner already exists, the existing control numbers will not be overwritten by the control numbers in the import file.</td></tr> </table> <p>This keyword is used with the following commands:</p> <p>IMPORT</p>	Value	Description	Y	Import control numbers with the trading partner.	N	Do not import control numbers. If this is a new trading partner, the control number will default to zeros. If this trading partner already exists, the existing control numbers will not be overwritten by the control numbers in the import file.
Value	Description						
Y	Import control numbers with the trading partner.						
N	Do not import control numbers. If this is a new trading partner, the control number will default to zeros. If this trading partner already exists, the existing control numbers will not be overwritten by the control numbers in the import file.						

Running the DataInterchange Utility in MVS

DataInterchange provides sample JCL statements for running the DataInterchange Utility in MVS. You can copy and customize these statements as necessary. See member EDIUTILD for DB2 or EDIUTILV for VSAM in the data set EDI.V3R1MO.SEDISAM1 in Appendix D, “Sample Programs” on page D-1. The following is an explanation of the parameters your application can pass in the JCL. All the parameters are keyword type parameters, and they are optional (except PLAN and SYSTEM, which may be required).

APPLID=aaaaaaa Specifies the application ID to run the DataInterchange Utility. This keyword also identifies the log file as specified in the ACTLOGS profile. Replace *aaaaaaa* with the ID of the application that initialized DataInterchange. If you specify this parameter, the activity log profile must contain a matching entry to define which log file is used for recording errors and events pertaining to the application. The two APPLID values shipped with DataInterchange are:

- EDIFFS (default)
 - Associated with the LOGFFS ddname
 - The default APPLID and log when using the utilities
- EDIMP
 - Associated with the LOGEDI ddname
 - The APPLID and log used during online DataInterchange processing

SYSID=bbbbbbb Identifies the installation-defined DataInterchange for MVS system used to run the EDIUTILV utility. This keyword is provided to help control access to various components of DataInterchange. The SYSID is part of the resource name defined using RACF or some other resource control product. Replace *bbbbbbb* with the ID used to protect DataInterchange services (for example, Resource Access Control Facility). The default ID is DIENU.

LANGID=ccc Identifies the language ID used to run the EDIUTILV utility. The value supplied must match an entry in the LANGPROF profile. The language ID is used to establish values such as date formats and decimal notation. Replace *ccc* with the following value for the language version:

Value	Description
ENU	English

DLM=d Specifies the delimiter used in place of left and right parentheses to enclose values in the EDIUTILV utility command language. Replace *d* with the delimiter you want to use in place of the left and right parentheses to enclose values in the DataInterchange Utility command language. You must supply this parameter if a keyword value contains either a left or right parenthesis.

MQSYSIN=eeeeeeee Replace *eeeeeeee* with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to either ddname SYSNAME or ddname EDISYSIN.

MQPRT=ffffff Replace *ffffff* with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname PRTFILE.

MQRPT=gggggggg Replace *gggggggg* with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname RPTFILE.

MQEXCP=hhhhhhh	
	Replace <i>hhhhhhh</i> with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname FFSEXCP.
MQTRAK=iiiiiii	Replace <i>iiiiiii</i> with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname FFSTRAK.
MQQUERY=jjjjjjj	Replace <i>jjjjjjj</i> with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname EDIQUERY.
PLAN=kkkkkkkk	In a DB2 environment, if the utility is invoked by way of IKJEFT01 and there is no EDITSIN data set allocated, then this parameter is required. For more information, see “DataInterchange DB2 Command File (EDITSIN)” on page 2-2. Replace <i>kkkkkkkk</i> with the DB2 plan name.
SYSTEM=///	In a DB2 environment, if the utility is invoked by way of IKJEFT01 and there is no EDITSIN data set allocated, then this parameter is required. For more information, see “DataInterchange DB2 Command File (EDITSIN)” on page 2-2. Replace <i>///</i> with the name of the DB2 subsystem (or group, if Data Sharing).

Examples of Using Optional Parameters

The following are examples of using these optional parameters:

DB2 RUN command:

```
| PARM('SYSID=TEST LANGID=ENU SYSTEM=DB93 PLAN=EDIENU31')
```

VSAM JCL:

```
PARM='SYSID=TEST LANGID=ENU'
```

Chapter 2. File Formats and DataInterchange Utility Records

Transaction Store Input and Output Files	2-1
Command File (EDISYSIN or SYSIN)	2-1
DataInterchange DB2 Command File (EDITSIN)	2-2
Network Commands File	2-2
Application File	2-3
Envelope File	2-3
Exception File (FFSEXCP)	2-4
Tracking File (FFSTRAK)	2-5
Print File (PRTFILE)	2-5
Report File (RPTFILE)	2-6
Query File (EDIQUERY)	2-7
Work File (FFSWORK)	2-8
Pageable Translation Work File (EDIVAX)	2-8
Enveloping Options File for Functional Acknowledgments (FAENV)	2-9
File Format	2-9
Sample Entries	2-10
A Practical Example	2-10
Export/Import Utility Function	2-12
Export/Import Control File (CTLFILE)	2-12
Export/Import Control File Label Descriptions	2-13
The Export/Import Files	2-15
Export/Import Common Control Record (0C1/0C2)	2-16
Export/Import Common End of Group Record (000)	2-17
E/I File Data Area	2-17
Export/Import of Trading Partner Profile	2-18
Export/Import Trading Partner Profile Header Record (7P1)	2-19
Profile Definition (7P1)	2-19
Export/Import Trading Partner Profile Record (7P2)	2-19
Trading Partner Profile Member (TPPROF-P2)	2-19
Trading Partner Profile Member Field Descriptions	2-22
TP Contacts Definition (7P3)	2-32
TP Control Numbers (7P4)	2-33
Comments Definition (7A1)	2-33
Contacts Definition (7Z1)	2-33
Export/Import of Standards Records	2-34
Standard/Envelope Definition (1S1/1E1)	2-35
Standard Transaction Definition (1S2)	2-35
Standard Segment Usage (1S3)	2-36
Standard/Envelope Segment Definition (1S4/1E4)	2-36
Standard/Envelope Data Element Usages (1S5/1E5)	2-36
Standard/Envelope Data Element Definition (1S6/1E6)	2-37
Export/Import of Application Data Formats	2-37
Application Data Format Definitions (2F1)	2-38
Application Data Format Structures (2F2)	2-38
Export/Import of Trading Partner Transactions	2-39
Trading Partner Transaction (M1/U1)	2-40
Trading Partner Mapping Segment (T2)	2-41
Trading Partner Mapping Data Element (T3)	2-42
Trading Partner Mapping Rules (T4)	2-42
Trading Partner Send Usage (T5)	2-43

Trading Partner Receive Usage (T6)	2-44
Export/Import of Table Definitions	2-45
Table Definition (B1)	2-46
Table Definition (B1) Field Descriptions	2-46
Table Entry (B2)	2-48
Table Entry (B2) Field Descriptions	2-48
Additional Profile Layouts	2-48
Requestor Profile Member (REQPROF-P2)	2-49
Security Profile Member (SECUPROF-P2)	2-49
Network Profile Member (NETPROF-P2)	2-50
Network Operations Profile Member (NETOP-P2)	2-50
Activity Log Profile Member (ACTLOGS-P2)	2-51
Application Definition Profile Member (APPDEFS-P2)	2-51
User Program Information Profile Member (ADAMCTL-P2)	2-52
Language Profile Member (LANGPROF-P2)	2-52
EDIFACT Profile Member (E-P2)	2-53
ICS Profile Member (I-P2)	2-54
UNTDI Profile Member (T-P2)	2-55
UCS Profile Member (U-P2)	2-56
X12 Profile Member (X-P2)	2-56
CONTRECV Member (CONTRECV-P2)	2-57
System Profile Member (SYSPROF-P2)	2-58
MQSeries Queue Profile Member (MQSERIES-P2)	2-59
Importing the New Definitions to DataInterchange	2-59
Mapping Migration Input Control File (INCNTL)	2-59
File Format	2-60
Mapping Migration Output Control File (OUTCNTL)	2-60
File Format	2-61
DataInterchange Utility Records Format	2-61
Control Record (C)	2-61
Data (D) Records	2-70
End Transaction and Interchange (Z) Records	2-71
Raw Data Records	2-72
Optional Records	2-72
Information (I) Record	2-74
Interchange Header (E) Record	2-75
Group Header (G) Record	2-75
Transaction Set Header (T) Record	2-76
Queuing Totals (Q) Record	2-76
Management Reporting	2-76
Trading Partner Profile Data Extract	2-76
Trading Partner Capability Data Extract	2-77
Network Activity Data Extract	2-78
Transaction Activity Data Extract	2-79
Transaction Store Data Extract Information Categories	2-80
Transaction Store Data Extract Common Key	2-81
Transaction Store Data Extract Record Formats	2-81
Interchange Data Extract Record Layout	2-81
Group Data Extract Record Layout	2-83
Transaction Data Extract Record Layout	2-83
Application Data Extract Record Layout	2-86
Transaction/Acknowledgment Image Data Extract Record Layout	2-87

Chapter 2. File Formats and DataInterchange Utility Records

This chapter describes the files used by the DataInterchange Utility and the format of data within each file. Appendix E, “Using Sample JCL” on page E-1 also contains information about the DataInterchange Utility files, and Table E-1 on page E-17 shows the required files for each of the PERFORM commands.

Transaction Store Input and Output Files

This section describes the input and output files used by the DataInterchange Utility.

Command File (EDISYSIN or SYSIN)

The command file contains the input DataInterchange Utility commands (PERFORM commands described in Chapter 1, “Using the DataInterchange Utility”) you want executed. The command language syntax is fairly free format and except for values associated with the following keywords, case insensitive:

- ACFIELD
- ADDRLN1
- ADDRLN2
- CMMTLN1
- CMMTLN2
- CMPYNM
- CNCTNM
- CNCTPH
- NETNAME
- STDDDESC

By placing an asterisk (*) in column one of a record, you are indicating the record is a comment. The DataInterchange Utility first checks to see if EDISYSIN is allocated. If it is, this ddname is used. If EDISYSIN does not exist, the ddname SYSIN is used. Prior to Release 4 of DataInterchange, the command file was always allocated to ddname SYSIN. The command file is opened for READ processing only, so it can be allocated as an inline data set in your DataInterchange Utility JCL. The command file specifications are:

Action	Description
Use	Input file containing PERFORM DataInterchange Utility commands.
Specified	Does not apply.
ddname	EDISYSIN or SYSIN. EDISYSIN takes precedence and will be used if allocated. These ddnames can be overridden and an MQSeries Queue be used instead. The optional parameter MQSYSIN= allows the specification of a DataInterchange MQSeries Queue profile member to be used instead of a sequential file.
	In CICS, the command file (EDISYSIN or SYSIN) name and type is specified in the DataInterchange Utility control information. See Table 5-3 on page 5-19 for details.
Suggested format	Record format: Fixed or variable. Record length: 80 bytes, but any length is not a problem.
Remarks	The processing of this file adjusts to the file attributes. Therefore, you can define this file as it best fits with your conventions.

DataInterchange DB2 Command File (EDITSIN)

This optional file is used to contain keywords that determine whether or not DataInterchange for MVS should attach and/or detach DB2. See “DataInterchange for MVS and DB2 Attachment” on page 3-2 for more information. This file applies only in MVS DB2 installations; it does not apply in CICS or VSAM installations.

Action	Description
Use	Input file containing information that tells the DataInterchange Utility whether or not to attach and/or detach DB2.
ddname	EDITSIN
Suggested format	Record format: Fixed or variable. Record length: 80 bytes
Remarks	Typically, this would be an in-stream JCL file that can be defined to best fit your conventions. The Utility can be invoked via EXEC PGM=EDIFFUT or via EXEC PGM=IKJEFT01. For the most part, if EXEC PGM=EDIFFUT is used, EDITSIN is used to contain the DB2 control information. If EXEC PGM=IKJEFT01 is used, SYSTSIN is used to contain the DB2 control information.
Example 1	<p>The following is an example using EXEC PGM=EDIFFUT.</p> <pre>//RUNDI EXEC PGM=EDIFFUT,DYNAMNBR=20,REGION=6144K, // PARM='SYSID=DIENU APPLID=EDIFFS LANGID=ENU' //EDITSIN DD * SYSTEM(DB93) PLAN(EDIENU31) OPEN(Y) CLOSE(Y) CAF(Y)</pre>
Example 2	<p>The following is an example using EXEC PGM=IKJEFT01.</p> <pre>//RUNDI EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=6144K //SYSTSIN DD * DSN SYSTEM(DB93) RUN PROG(EDIFFUT) PARM('SYSID=DIENU APPLID=EDIFFS LANGID=ENU SYSTEM=DB93 PLAN=EDIENU31') PLAN(EDIENU31) END /*</pre>

Network Commands File

The network commands file contains the commands that you want DataInterchange to pass to the network. DataInterchange will read the commands from a member of this partitioned data set (PDS) and write the commands to the Network input file specified in the network profile member. This interface to networks can be used instead of using various NETOP profile member commands; however, not all networks are supported through this interface.

Action	Description
Use	Holds network commands that the user would like DataInterchange to pass to the network.
Specified	Specific members to be used are specified in the Network cmds file field of the trading partner or requestor profiles.
ddname	EDINTCMD
Suggested format	Record format: Fixed or variable. Record length: 80 bytes.

Remarks	The network commands can contain variables that are resolved by DataInterchange before the commands are passed to the network. The expanded records will be truncated if they exceed the file record length.
---------	--

Application File

The application file contains the input records that DataInterchange uses to translate application data into an EDI standard format during the send process, and the output records DataInterchange creates with translation from a standard to an application format during the receive process. For translating to an application format, this file must be able to handle the largest data record you expect to receive and the largest information record the DataInterchange Utility might return. If you request raw data records, the DataInterchange Utility writes the information and other optional records to the exception file (FFSEXCP). The DataInterchange Utility opens the file for OUTPUT when processing the first received transaction. It then opens the file for EXTEND. Use the JCL DISP options to control whether the file is cleared or appended to during the first use.

For TRANSLATE TO STANDARD operations, the RAWFMTID keyword on the PERFORM command indicates if the file contains raw data records. If RAWFMTID is omitted, the file is expected to contain C and D records. For TRANSLATE TO APPLICATION operations, the application data format and the RAWDATA keyword on the PERFORM command indicates if the file is written in C and D record format or raw data record format. If the RAWDATA(Y) keyword is used and the application data format does not have a record ID position specified, C and D records will be written. The application file specifications are:

Action	Description
Use	Input file for translating to standard; output file for translating to application.
Specified	Sending: APPFILE keyword. Receiving: Data format; transaction usage override panel
ddname	User defined.
Suggested format	Record format: Fixed or variable. Record length (whichever is greater): <ul style="list-style-type: none"> • If raw data is used, maximum structure size • If C and D records are used, maximum structure size plus 17 bytes • If C and D records are used, 1024 bytes • If I records are used, 483 bytes
Remarks	Records are truncated if record length is not large enough.

Envelope File

For outbound documents, the envelope file is the file specified as the Trans data queue in the network profile member or an override file specified in a command that requests enveloping. For inbound documents, this is the Receive file name specified in the requestor profile member or an override file specified in a command that requests receiving. The envelope file specifications are:

Action	Description
Use	Sending: Holds complete envelopes when enveloping takes place. Receiving: Contains envelopes to be deenveloped and optionally translated.

Specified	<p>Sending: Transaction data queue in network profile or overridden by FILEID keyword in command.</p> <p>Receiving: Receive file in requestor profile or overridden by FILEID keyword in command.</p>
ddname	User defined. Default value of QDATA, QDATAE, or QDATAU during enveloping.
Suggested format	<p>Record format: Fixed or variable.</p> <p>Record length: 80 bytes or greater.</p>
Remarks	<p>For enveloping on point-to-point networks, the enveloping file is dynamically allocated with the DS name that the current user ID and trading partner nickname construct.</p> <p>The envelope file for Fixed-to-Fixed translations is based on the standard ID used in the mapping. For Fixed-to-Fixed translations that have a target application data format, the standard ID is the same as the Application file name within the application data format. The File suffix field in the trading partner profile (TPPROF) member is used as a suffix to the standard ID so that there can be a unique envelope file per trading partner. Data written to these files can have either a C and D record format or a raw data format based on the RAWDATA keyword used in the PERFORM command. Please see "DataInterchange Utility Records Format" on page 2-61 for a description of C and D records and the RAWDATA format.</p> <p>Note: The envelope file formed by the above concatenation can be overridden with the FIXEDFILEID keyword on the PERFORM command.</p>

Exception File (FFSEXCP)

For translating to application format, the DataInterchange Utility writes translated transactions to the exception file if it cannot open the file intended to receive them, or if a file name is not provided. When translating to standard format, the DataInterchange Utility writes transactions that were not translated successfully to this file. It also writes the optional records to this file if the tracking file (FFSTRAK) does not exist.

This file must be large enough to contain the largest data record you are sending or receiving, and the largest information record the translator might return. The DataInterchange Utility opens the file for OUTPUT when processing the first transaction. It then opens the file for EXTEND. Use the JCL DISP options to control whether the file is cleared or appended to during the first use. The exception file specifications are:

Action	Description
Use	<p>Sending: Holds transactions that were not translated successfully and the unidentified and optional records (see also Tracking file).</p> <p>Receiving: Holds transactions that could not be written to the application file, and optional records if the application file contains raw data.</p> <p>See "Optional Records" on page 2-72 for more information.</p>
Specified	Does not apply.
ddname	<p>FFSEXCP. The ddname can be overridden and an MQSeries Queue used instead. The optional parameter MQEXCP= allows the specification of a DataInterchange MQSeries Queue profile member to be used instead of a sequential file.</p> <p>In CICS, the exception file (FFSEXCP) name and type is specified in the DataInterchange Utility control information. See Table 5-3 on page 5-19 for details.</p>

Suggested format	Record format: Fixed or variable.
	Record length (whichever is greater):
	<ul style="list-style-type: none"> • If raw data is used, maximum structure size • If C and D records are used, maximum structure size plus 17 bytes • If C and D records are used, 1024 bytes • If I records are used, 483 bytes
Remarks	Records are truncated if record length is not large enough.

Tracking File (FFSTRAK)

For translating to standard format, the DataInterchange Utility writes the optional records to the tracking file if it exists. This file must be large enough to contain the largest information record that the DataInterchange Utility might return. The tracking file specifications are:

Action	Description
Use	<p>Sending: Holds optional records. See “Optional Records” on page 2-72 for more information.</p> <p>Receiving: Does not apply.</p>
Specified	Does not apply.
ddname	<p>FFSTRAK (optional). This ddname can be overridden and an MQSeries Queue used instead. The optional parameter MQTRAK= allows the specification of a DataInterchange MQSeries Queue profile member to be used instead of a sequential file.</p> <p>In CICS, the tracking file (FFSTRAK) name and type is specified in the DataInterchange Utility control information. See Table 5-3 on page 5-19 for details.</p>
Suggested format	Record format: Fixed or variable.
	Record length (whichever is greater):
	<ul style="list-style-type: none"> • If C and D records are used, 1024 bytes • If I records are used, 483 bytes
Remarks	Records are truncated if the record length is not large enough. If FFSTRAK is not supplied and your application is using C and D records, DataInterchange writes the optional records to the exception file (FFSEXCP). You cannot mix optional records with raw data in the exception file, so if FFSTRAK is not supplied and your application uses raw data, DataInterchange will not write the optional records.

Print File (PRTFILE)

The print file must allow for a minimum record size of 132 bytes. The DataInterchange Utility opens this file for OUTPUT. Use the JCL DISP options to control whether the file is cleared or appended to. The print file specifications are:

Action	Description
Use	Contains an audit report from the DataInterchange Utility showing the results of processing.
Specified	Does not apply.

ddname	PRTFILE. This ddname can be overridden and an MQSeries Queue used instead. The optional parameter MQPRT= allows the specification of a DataInterchange MQSeries Queue profile member to be used instead of a sequential file.
	In CICS, the print file (PRTFILE) name and type is specified in the DataInterchange Utility control information. See Table 5-3 on page 5-19 for details.
Suggested format	Record format: FBA or VBA. Record length: 132 bytes.
Remarks	Status of processing and any errors DataInterchange encounters are written to this file during DataInterchange Utility processing.

The following is a sample PRTFILE.

```
Audit Trail Report -DataInterchange Utility- Date: 93/10/13 Time: 11:10:22 Page: 0001

Interchange Control Number = 00000000000057

FF0010 Transaction number 1 to 1 translated successfully

FF0013 Transactions with Interchange Control Number 00000000000057 were successfully queued

Message: TR0004 Severity: 04
Code in ID type field not found in validation table. Internal Trading Partner ID and Application Format = DLGTYPES -
DLGTYPES. Transaction handle, code, mode, and function = 19931013111038000000 - DL1 - PRODUCTION - SEND. Interchange,
group, and transaction control numbers = 0000000056 - 56 - 0155. Current Loop-ID and repetitions = 110000 - 1 - 1.
Standard segment and field ID = SEG3(003) - 3 - 1 - 11. Application field ID = SEG3 - 61. Data type and value = Z -
123456. Validation table name = DLGP2V.

Interchange Control Number = 00000000000056

FF0011 Transaction number 2 translated with errors

Message: TR0004 Severity: 04
Code in ID type field not found in validation table. Internal Trading Partner ID and Application Format = DLGTYPES -
DLGTYPES. Transaction handle, code, mode, and function = 19931013111039000000 - DL1 - PRODUCTION - SEND. Interchange,
group, and transaction control numbers = 0000000056 - 56 - 0156. Current Loop-ID and repetitions = 110000 - 1 - 1.
Standard segment and field ID = SEG3(003) - 3 - 1 - 11. Application field ID = SEG3 - 61. Data type and value = Z -
123456. Validation table name = DLGP2V.
```

Report File (RPTFILE)

The DataInterchange Utility opens the report file for OUTPUT for the first report and as EXTEND for successive reports. Use the JCL DISP options to control whether the file is cleared or appended to during the first use. The report file specifications are:

Action	Description
Use	Contains reports requested during Transaction Store processing.
Specified	Does not apply.
ddname	RPTFILE. This ddname can be overridden and an MQSeries Queue used instead. The optional parameter MQRPT= allows the specification of a DataInterchange MQSeries Queue profile member to be used instead of a sequential file.
	In CICS, the report file (RPTFILE) name and type is specified in the DataInterchange Utility control information. See Table 5-3 on page 5-19 for details.
Suggested format	Record format: FBA or VBA. Record length: 132 bytes.
Remarks	Reports requested from the Transaction Store Facility or DataInterchange Utility are written to this file.

Query File (EDIQUERY)

The query file is the output file for the following commands:

- **PERFORM QUERY**

See “Query Transaction” on page 1-47 for a description of the record layout.

- **PERFORM TRANSACTION DATA EXTRACT**

See “Transaction Store Data Extract Information Categories” on page 2-80 for a description of the record layout.

- **PERFORM ENVELOPE DATA EXTRACT**

See “Transaction Store Data Extract Information Categories” on page 2-80 for a description of the record layout.

- **PERFORM TRADING PARTNER PROFILE DATA EXTRACT**

See “Management Reporting” on page 2-76 for a description of the record layout.

- **PERFORM TRADING PARTNER CAPABILITY DATA EXTRACT**

See “Management Reporting” on page 2-76 for a description of the record layout.

- **PERFORM NETWORK ACTIVITY DATA EXTRACT**

See “Management Reporting” on page 2-76 for a description of the record layout.

- **PERFORM TRANSACTION ACTIVITY DATA EXTRACT**

See “Management Reporting” on page 2-76 for a description of the record layout.

The query file is opened for OUTPUT for the first PERFORM command executed and opened for EXTEND for all PERFORM commands thereafter within a single execution of the DataInterchange Utility. Use the JCL DISP options to control whether the file is cleared or appended to during the first use. The query file specifications are:

Action	Description
Use	Output file for PERFORM commands stated above.
Specified	Does not apply.
ddname	EDIQUERY (required if one of the above commands is used and the command generated output). This ddname can be overridden and an MQSeries Queue used instead. The optional parameter MQQUERY= allows the specification of a DataInterchange MQSeries Queue profile member to be used instead of a sequential file. In CICS, the query file (EDIQUERY) name and type are specified in the DataInterchange Utility control information. See Table 5-3 on page 5-19 for details.
Suggested format	Record format: Fixed or variable. Record length: 32756 to ensure largest record will not be truncated.
Remarks	This file is used for the output of many different types of records. Therefore, it is suggested to allocate the file as variable length record format with a maximum record length of 32756. If you use this file for a specific set of records that do not require this maximum record length, you can allocate the file to meet your requirements. Records are truncated if the record length supplied is not large enough.

Work File (FFSWORK)

The work file is an internal work file DataInterchange uses during send translate processing. The DataInterchange Utility opens this file for OUTPUT only, and it should always be empty.

Note: This file is handled internally by DataInterchange when working in a CICS environment.

The work file specifications are:

Action	Description
Use	Sending: Holds the current transaction for transfer to the exception file if translation is not successful.
Specified	Does not apply.
ddname	FFSWORK (required). External specification is not made in CICS. DataInterchange takes care of this file internally.
Suggested format	Record format: Variable blocked (VB) Record length: 32756 bytes.
Remarks	A temporary file used only during send-translate processing. This file is used extensively as a temporary file during a translate-to-standard function and is a prime candidate for a virtual I/O data set.

Pageable Translation Work File (EDIVAX)

EDIVAX is a temporary work file used by DataInterchange when Pageable Translation is enabled and virtual storage usage for EDI and application data reaches 28 MB. Pageable Translation is enabled by using the PAGE keyword on Utility PERFORM commands and, for API applications, by setting the VAXFLAG field in the TRCB to X. Pageable Translation will ensure that virtual storage usage for EDI and application data does not exceed 28 MB by paging any excess to the EDIVAX file.

The amount of space allocated to this file will be dependent upon the maximum amount of data to be translated. This can be calculated by adding the following components:

- Number of bytes of largest interchange
- Number of bytes of largest application transaction image
- 4 MB overhead
- Number of structures in largest interchange multiplied by 120 bytes

Pageable Translation deals with the first two components, and ensures that the amount of virtual storage required for them does not exceed 28 MB. The other components are not addressed by Pageable Translation. The maximum amount of data that DataInterchange can page with Pageable Translation is approximately one gigabyte (specifically, 32,000 multiplied by 28,632 bytes).

The Pageable Translation work file specifications are:

Action	Description
Use	Holds paged EDI and application data once virtual storage to hold this data reaches 28 MB.
Specified	Does not apply.
ddname	EDIFAX (required for Pageable Translation).

Suggested format	DCB statement should not be specified (this is under DataInterchange control).
Remarks	Omit the data set name to allow MVS to assign a temporary data set name.

Enveloping Options File for Functional Acknowledgments (FAENV)

The enveloping options file is an optional QSAM file that you can use if you need more flexibility in the enveloping of functional acknowledgments.

Note: For this file, a functional acknowledgment is either an ANSI X12, 997 or UCS 999 transaction set, or an EDIFACT CONTRL message.

With this file you can:

- Specify interchange and group envelope overrides for functional acknowledgments
- Specify a standard profile member to fill envelope data elements other than the sender and receiver IDs

To use the file, specify the ddname FAENV in the JCL for any request that includes the DEENVELOPE, RECEIVE AND DEENVELOPE, or RECEIVE AND TRANSLATE command.

Note: In CICS, FAENV is a VSAM entry sequence data set named EDIFAENV.

The enveloping options file for functional acknowledgments specification are:

Action	Description
Use	Receiving: Provides flexibility in determining the data used in the enveloping segments when functional acknowledgments are generated.
Specified	Does not apply.
ddname	FAENV (optional) for MVS. EDIFAENV (optional) for CICS.
Suggested format	Record format: Fixed or variable. Record length: As large as the longest record in the file.
Remarks	Optional file used only during deenveloping to control the data used to build the enveloping (service) segments, such as ISA and GS segments, for the functional acknowledgment being returned to your trading partner.

File Format

The following is the format of an entry in the FAENV file:

ISID,IRID,GSID,GRID = ISIDFA,IRIDFA,GSIDFA,GRIDFA>EPM

Where:

- , Separates envelope fields and indicates that a value is not listed
- = Separates key fields from override fields
- > Separates override fields from an envelope profile name

Table 2-1 on page 2-10 describes each field in the file format.

Table 2-1. Fields in the Enveloping Options File for Functional Acknowledgments

Field	Maximum length	Description
ISID	35	Interchange sender ID from the inbound envelope
IRID	35	Interchange receiver ID from the inbound envelope
GSID	35	Group sender ID from the inbound envelope
GRID	35	Group receiver ID from the inbound envelope
ISIDFA	35	Interchange sender ID override for outbound functional acknowledgment envelope
IRIDFA	35	Interchange receiver ID override for outbound functional acknowledgment envelope
GSIDFA	35	Group sender ID override for outbound functional acknowledgment envelope.
GRIDFA	35	Group receiver ID override for outbound functional acknowledgment envelope
EPM	8	Standard envelope profile member name
Note: All fields are optional and variable in length.		

The search key for an entry is ISID, IRID, GSID, and GRID, corresponding to the inbound envelope that is being received or deenveloped. The remaining fields of the entry specify the overrides you want to use for the outbound envelope containing the functional acknowledgments. If the FAENV file exists, DataInterchange searches it for a matching entry. You do not have to include all of the key values in an entry or any overrides. If you do not provide overrides, an envelope profile member name is expected. You can use overrides in conjunction with a profile member. The applicable fields taken from the profile member are the same as those normally used during translation to send. If the member name and overrides are both present, the overrides are used. Entries on the right side of the equal sign override entries from the envelope profile member.

For more information, see the *DataInterchange Administrator's Guide*.

Sample Entries

Each of the following is a valid entry in FAENV:

```
ISID1,IRID1,=ISID1X,IRID1X,GSID1X,GRID1X
ISID2,,GSID2=ISID2X>EPM2
,IRID3,GSID3,GRID3=,,,GRID3X>EPM3
ISID4,IRID4,,GRID4=ISID4X,,,GRID4X>EPM4
ISID5,IRID5,GSID5=ISID5X,IRID5X,,GRID5X>EPM5
ISID6,IRID6=ISID6,IRID6X,GSID6X>PM6
ISID7=>PM7
```

A Practical Example

Two inbound X12 interchanges and one EDIFACT interchange contain transactions with the following envelope values:

Interchange 1 (X12): ISA06 = ISID = TPDUNS#
 ISA08 = IRID = MYDUNS#DIV1
 GS02 = GSID = TPDUNS#
 GS03 = GRID = MYDUNS#DIV1

Interchange 2 (X12): ISA06 = ISID = TPDUNS#
 ISA08 = IRID = MYDUNS#DIV2
 GS02 = GSID = TPDUNS#
 GS03 = GRID = MYDUNS#DIV2

Interchange 3 (EDIFACT): UNB03 = ISID = ACCX ACCX01
 UNB06 = IRID = ACCY ACCY03
 UNG02 = GSID = ACCOUNTING
 UNG04 = GRID = BOOKING

Without the FAENV file, the interchange and group envelopes created for the functional acknowledgments would be the same for interchange 1 as for interchange 2. In addition, if they are deenveloped one after the other, both functional acknowledgments are placed in the same outbound group and interchange envelopes. This happens because there is no distinction made between the different inbound interchange receiver IDs. The envelopes for the outbound acknowledgments are produced using the following values:

FA interchange: ISA06 (or UNB03) = value from first envelope profile member
 ISA08 (or UNB06) = value of account number and user ID from
 trading partner profile member
 GS02 (or UNG02) = value from first envelope profile member
 GS03 (or UNG04) = value from first envelope profile member

The first envelope profile member is obtained from the receive usage of the first transaction set of the interchange. If both inbound interchanges are processed one after the other, it is the envelope member obtained for the first transaction set for the first interchange.

If you need to make a distinction between sender IDs for outbound functional acknowledgments, use the FAENV file to modify the normal procedure.

For the previous examples, the following entries are needed in FAENV:

TPDUNS#,MYDUNS#DIV1=MYDUNS#DIV1,TPDUNS#,MYDUNS#DIV1,TPDUNS#
 TPDUNS#,MYDUNS#DIV2=MYDUNS#DIV2,TPDUNS#,MYDUNS#DIV2,TPDUNS#
 ACCX ACCX01,ACCY ACCY03,ACCOUNTING=ACCY ACCY03,ACCX ACCX01,BOOKING,ACCOUNTING

The key specifies only the ISID and IRID. Omitting a key value indicates that it is not important. Therefore, any GSID and GRID values meet the conditions, and the overrides are used. As a result, the functional acknowledgments that are produced have the following separate envelopes:

FA interchange 1: ISA06 = MYDUNS#DIV1
 ISA08 = TPDUNS#
 GS02 = MYDUNS#DIV1
 GS03 = TPDUNS#

FA interchange 2: ISA06 = MYDUNS#DIV2
 ISA08 = TPDUNS#
 GS02 = MYDUNS#DIV2
 GS03 = TPDUNS#

FA interchange 3: UNB03 = ACCY ACCY03
 UNB06 = ACCX ACCX01
 UNG02 = BOOKING
 UNG04 = ACCOUNTING

Using the FAENV file allows you to separate the functional acknowledgments into interchange and group envelopes with the values you specify.

Export/Import Utility Function

DataInterchange provides an Export/Import utility function for updating DataInterchange's databases in a batch environment. The Export/Import batch function uses three files:

1. **Batch Control File (CTLFILE)**

Contains the control information that describes the data being exported or imported.

2. **Export/Import Files (E/I File)**

Contains the data that is being imported or exported (in tagged or fixed format).

3. **Print File (PRTFILE)**

Contains a report on Export/Import activity.

The Export/Import facility allows you to choose one, any combination, or all of the following categories of DataInterchange data for extraction (Export) or loading (Import) of EDI data:

1. Standards/Standard transaction
2. Application transaction data formats
3. Trading partner transactions
4. Control strings
5. Document definitions
6. Document layouts
7. Profiles
8. Tables

Each category is a complete set with or without the associated objects depending on your choice. For example, for export/import of a standard, the complete set includes:

- standard definition
- transaction definitions
- segment definitions
- segment usages
- data element definitions
- data element usages

The associated objects are:

- the standard envelope
- the validation tables (if validation is required)

The complete set and associated objects for each category are listed on the following pages.

Export/Import Control File (CTLFILE)

The export/import control file describes what data is being exported or imported. The control file can process multiple export or import requests. For example, to export seven different application data formats, you would have seven records in your control file. The control file provides the same information to the DataInterchange Utility that you enter when using the export and import panels, subject to certain member selection exceptions. For information about the exceptions, see the *DataInterchange Administrator's Guide*.

The control file is used with PERFORM EXPORT and PERFORM IMPORT commands. It is specified by way of the CTLFILE keyword. The suggested record format can be fixed or variable, but the record length should not exceed 84. The CTLTYPE keyword can be used to specify the type of file.

Table 2-2 describes each label in the export/import control file.

Table 2-2. Export/Import Control File

Label	Location	Type	Length	Description
CATEGORY	1	Character	1	Category of the transaction
REPLACE	2	Character	1	Replace named object
KEYID	3-18	Character	16	ID of object
ASSOBJ	19-33	Character	15	Associated objects
USAGETID	40-55	Character	16	TPT ID for usage import
MBRNAME	60-83	Character	24	Member name for profile export

Export/Import Control File Label Descriptions

Export/Import control file label descriptions are:

Label	Description																				
CATEGORY	Specifies the category of the transaction and is required for both export and import. Valid values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>1</td><td>Standards/transaction sets</td></tr> <tr> <td>2</td><td>Application data formats</td></tr> <tr> <td>3</td><td>Trading partner transactions</td></tr> <tr> <td>4</td><td>Control strings</td></tr> <tr> <td>5</td><td>Document definitions</td></tr> <tr> <td>6</td><td>Document layouts</td></tr> <tr> <td>7</td><td>Profiles</td></tr> <tr> <td>8</td><td>Tables</td></tr> <tr> <td>9</td><td>All categories (import only)</td></tr> </table>	Value	Description	1	Standards/transaction sets	2	Application data formats	3	Trading partner transactions	4	Control strings	5	Document definitions	6	Document layouts	7	Profiles	8	Tables	9	All categories (import only)
Value	Description																				
1	Standards/transaction sets																				
2	Application data formats																				
3	Trading partner transactions																				
4	Control strings																				
5	Document definitions																				
6	Document layouts																				
7	Profiles																				
8	Tables																				
9	All categories (import only)																				
REPLACE	For import only, indicates whether the same named object in the database should be replaced. This flag does not apply to trading partner transaction usages or to profile members. It does apply to the parent categories of entire trading partner transaction and entire profiles. This flag applies only to the primary object identified by the category code. All associated objects that are imported will overwrite duplicate entries currently in DataInterchange without issuing a warning message. Valid values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>1</td><td>Replace</td></tr> <tr> <td>(other)</td><td>Do not replace</td></tr> </table>	Value	Description	1	Replace	(other)	Do not replace														
Value	Description																				
1	Replace																				
(other)	Do not replace																				
KEYID	Specifies the ID of the object to be exported or imported. The value must be left justified. For exporting standard transactions, the standard ID is left justified in the first eight bytes, and the transaction ID is left justified in the second eight bytes. For importing specific																				

standard transaction sets, the import file must contain a complete standard with all transaction sets included. If no key value is present for Application Data Formats, Trading Partner Transactions, or Standards, then all records will be exported for the specified category.

For importing Transaction usages only, the key value must not be blank. For other imports, if no key value is present, all the records for the category (specified in position 1) are processed.

ASSOBJ

For export only, specifies an array specifying the associated objects. The array contains a 1 (yes) and anything else (no) for each associated object, in the following order:

- For standards
 1. Validation tables
 2. Envelope profile
 3. Envelope standard
- For application data formats
 - None
- For trading partner transactions and control strings
 1. Transaction usages
 2. Control string
 3. Standard transaction
 4. Application data format
 5. Validation tables
 6. Translation tables
 7. User exit routines
 8. Trading partner profile
 9. Translation exit routines
 10. Security profile
 11. Network profile
 12. Network operations
 13. Envelope profile
 14. Envelope standard
 15. Mapping

Note: To export a usage without its mapping, specify a 1 for transactions (item 1) and a 0 for mapping (item 15).
- For business document definitions
 1. Application data format
 2. Requestor profile
 3. Business document layout
- For business document layouts
 1. Application data format
- For profiles
 1. None
- For tables
 1. None

ASSOBJ	For import only, specifies an array specifying the associated objects. The array contains a 1 (yes) and anything else (no) for each associated object in the following order: <ul style="list-style-type: none"> • For trading partner transactions <ul style="list-style-type: none"> – Transaction usages only (the KEYID value must not be blank)
USAGETID	For import only, specifies the trading partner transaction ID under which to import the usages. If blank, the imported TPT ID is used.
MBRNAME	Specifies the profile member name to export or import.

The Export/Import Files

The export/import files are used for output when you export DataInterchange data or are used as input when you import DataInterchange data. Export/Import files are sequential, contain variable length records, and must be allocated with a record format of V, LRECL=8152, and BLKSIZE=8156. There are several export/import files, each associated with the category of data they hold. The ddnames are:

ddname	Category
EDIEISTD	Standard/Standard Transaction
EDIEIADF	Application transaction data format
EDIEITBL	Tables
EDIEITPT	Trading Partner Transactions
EDIEICST	Control Strings
EDIEIDDF	Document Definitions
EDIEIPRF	Profiles
EDIEIMAP	Document Layouts

You can maintain export files and the data in those files by using the following MVS/TSO ISPF utility functions:

Use:	To:
DATASET	Allocate, rename, delete, and display export data set information
COPY	Copy export data set
DSLIST	Print, rename, delete, browse, and display export data set information

Note: For CICS, if maintenance of the export files is desired, define them as QSAM extra-partitioned transient data queues at DataInterchange system generation. To perform maintenance, close the queues in CICS, and use the ISPF utility functions in the MVS/TSO environment. Then run batch JCL to execute the utility functions in the MVS/TSO environment. CICS has the export/import files allocated and the data set names cannot be shared with ISPF utilities. When export queue maintenance is complete, open the queues in CICS. If the ddname associated with the queue specifies DISP=OLD, the queue is cleared when opened. If the ddname associated with the queue specifies DISP=MOD, the queue is not cleared when opened, and new export data is appended to the end of the queue.

DataInterchange requires that the records contained within each Export/Import file follow a certain sequence. The first record of the Export/Import file must be the **Common Control Record (0C1 or 0C2)**. This record contains information such as the date, time and system release level. The last record of each group must be the **Common End of Group Record (000)**. In between the **0C1 or 0C2** and the first **000** record and also between each **000** record and its subsequent **000** record are the data records. These

records vary in purpose and length, and they must be written in the format described in the following tables.

The first time the user exported 7 trading partner profiles in tagged format. This resulted in DataInterchange writing out the Common Control Record (0C1) and then a Trading Partner Profile Header Record (7P1) to specify that the following records are part of a trading partner profile export. This is then followed by 7 Trading Partner Profile Detail Records (7P2), one for each trading partner. It is then terminated with a Common End of Group Record (000).

The following diagram is an example of a set of records written to the EDIEIPRF ddname when exporting multiple Trading Partner Profiles.

Export/Import Common Control Record (0C1)
First Trading Partner Profile Header Record (7P1)
First Trading Partner Profile Detail Record (7P2)
Next Trading Partner Profile Detail Record (7P2)
Next Trading Partner Profile Detail Record (7P2)
Next Trading Partner Profile Detail Record (7P2)
Next Trading Partner Profile Detail Record (7P2)
Next Trading Partner Profile Detail Record (7P2)
Last Trading Partner Profile Detail Record (7P2)
Export/Import Common End of Group Record (000)

Export/Import Common Control Record (0C1/0C2)

The Common Control Record defines the format of the records (0C1 defines a tagged file and 0C2 defines a fixed format file), the user ID, date, time, and DataInterchange version/release number. This record is required and it **MUST** be the first record in the Export/Import record data set.

Note: The first three bytes of this record are 0C1 or 0C2, starting in column 1. The 0C1 or 0C2 occurs only once in an export/import data set and is always the first record.

Table 2-3 (Page 1 of 2). Export/Import Common Control Record (0C1/0C2) Fields

Field Name	Length	Type	Field Description	Position
Category	1	CHAR	Export/Import Record Category Code Category Code = '0'	001-001
Rectype	2	CHAR	Export/Import Record Type Code Record Type = 'C1' (Tagged format) Record Type = 'C2' (Fixed format)	002-003
User ID	8	CHAR	User ID of person creating the Export/Import record data set	004-011
Date	6	CHAR	Creation date of Export/Import data set YYMMDD	012-017

Table 2-3 (Page 2 of 2). Export/Import Common Control Record (0C1/0C2) Fields

Field Name	Length	Type	Field Description	Position
Time	6	CHAR	Creation time of Export/Import data set HHMMSS	018-023
Language	3	CHAR	DataInterchange language code English (United States) = 'ENU'	024-026
DIVersRel	12	CHAR	DataInterchange Version and Release Vers 1 Rel 3 = '010103000000' Vers 1 Rel 4 = '010104000000' Vers 1 Rel 5 = '010105000000' Vers 2 Rel 1 = '010201000000'	027-038

An example of a completed 0C1 record is below. This example depicts SMITH as the user ID, a creation date of 940419, a creation time of 161616, the language as English, and DataInterchange Version 1 Release 4.

0C1 SMITH 940419161616ENU010104000000

Export/Import Common End of Group Record (000)

The Common End of Group Record indicates the end of an Export/Import group of records. This record is required and it MUST be the last record of an Export/Import group.

Table 2-4. Export/Import Common End of Group Record (000) Fields

Field Name	Length	Type	Field Description	Position
Category	1	CHAR	Export/Import Record Category Code Category Code = '0'	001-001
Rectype	2	CHAR	Export/Import Record Type Code Record Type = '00'	002-003

The Common End of Group Record contains only '000', starting in column 1. The following is an example:

000

E/I File Data Area

The data area varies depending upon which category of data you are importing or exporting. The following categories are supported:

1. Standards/Standard transactions
2. Application transaction data formats
3. Trading partner transactions
4. Control strings

5. Document definitions
6. Document layouts
7. Profiles
8. Tables

Regardless of the category chosen, the data portion (tagged or fixed) of an E/I file starts in column 4 of the record. Columns 1-3 are reserved for the Export/Import record ID (RECID). As discussed earlier, the Common Control Record (0C1/0C2) and Common End of Group Record (000) are fixed records. All records within the E/I file data area are either tagged records or fixed format records, except for the control string (G1) and document layout (L1), which are native database format. For tagged format records, the fields can be placed anywhere in the record and are not column-dependent. For fixed format records, the fields are in a fixed position and have a fixed length.

Note: Fixed format records are subject to change due to design changes within DataInterchange. When fields are added, they are added to the end of the records.

The following tables define the order in which the fields are exported (position of fixed format fields), the length of fixed format fields (maximum length for tagged fields), and the type of data. Data with type CHAR can have any valid characters. Data with type DEC must have only decimal characters (0-9). Data with type HEX when exported is expanded to two hex representation characters (0-9,A-F) for each hex byte.

Export/Import of Trading Partner Profile

The trading partner profile contains information about the companies that exchange business transactions with you. There must be one record for each trading partner that you wish to define.

The Export/Import format for defining trading partners consists of the following record types:

- A single Trading Partner Profile Header Record (7P1)
- One or more Trading Partner Profile Records (7P2)
- Optionally, one or more Contact Records (7Z1)
- Optionally, one or more Trading Partner Contact Records (7P3)
- One or more Trading Partner Control Number Records (7P4)
- At most, one Trading Partner Comment Record (7A1)

The trading partner profile record group must be followed by the Export/Import Common End of Group Record (000). The following example illustrates a valid sequence of records written to ddname EDIEIPRF when exporting multiple Trading Partner Profiles.

```

0C1.....etc.    (E/I Common Control Record)
7P1.....etc.    (Trading Partner Header Record)
7P2.....etc.    (Trading Partner Profile Record)
7P4.....etc.    (Trading Partner Control Number Record)
7A1.....etc.    (Trading Partner Comment Record)
7P2.....etc.    (Trading Partner Profile Record)
7Z1.....etc.    (Contact Record)
7Z1.....etc.    (Contact Record)
7P3.....etc.    (Trading Partner Contact Record)
7P3.....etc.    (Trading Partner Contact Record)
7P3.....etc.    (Trading Partner Contact Record)
7P4.....etc.    (Trading Partner Control Number Record)
7A1.....etc.    (Trading Partner Comment Record)
7P2.....etc.    (Trading Partner Profile Record)
7Z1.....etc.    (Contact Record)
7P3.....etc.    (Trading Partner Contact Record)

```

	7P4.....etc.	(Trading Partner Control Number Record)
	7P2.....etc.	(Trading Partner Profile Record)
	7P4.....etc.	(Trading Partner Control Number Record)
	7P2.....etc.	(Trading Partner Profile Record)
	7P4.....etc.	(Trading Partner Control Number Record)
	000	(E/I Common End of Group Record)

Export/Import Trading Partner Profile Header Record (7P1)

The trading partner profile header record (7P1) is required and **MUST** be the first record of a trading partner profile record group. The trading partner profile group is defined as one trading partner profile header (7P1) record followed by one or more trading partner profile records (7P2).

The trading partner profile header record (7P1) contains the trading partner profile indicator and profile description (see Table 2-5).

Profile Definition (7P1)

Table 2-5. Trading Partner Profile Header Record

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile indicator=TPPROF	004-011
PROFDESC	35	CHAR	Profile/table description	012-046

Export/Import Trading Partner Profile Record (7P2)

The trading partner profile record defines descriptive and control data to identify an individual trading partner and their processing options. Multiple 7P2 records can be created (one record per trading partner), to support creation of multiple trading partner profiles.

For tagged files, the tags for the trading partner profile record (7P2) can be placed in any column or any order you choose. When importing data, all variables must be delimited by parentheses (). When exporting, variable data will be placed within the parentheses. All possible profile member tags and their associated maximum record lengths are listed in Table 2-6.

For fixed format records, the fields must be in the position specified and for the length specified in the table.

Trading Partner Profile Member (TPPROF-P2)

Table 2-6 (Page 1 of 3). Trading Partner Profile Members

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile ID	004-011
TPNICKN	16	CHAR	Trading partner nickname	012-027
NETID	8	CHAR	Network ID	028-035
SYSQUAL	1	CHAR	System qualifier	036-036
SYSID	8	CHAR	System ID	037-044
ACCTID	32	CHAR	Account number	045-076
USERID	32	CHAR	User ID	077-108
INTQUAL	4	CHAR	Interchange ID qualifier	109-112

Table 2-6 (Page 2 of 3). Trading Partner Profile Members

Tag	Length	Type	Description	Fixed Position
INTID	35	CHAR	Interchange ID	113-147
CONAME	40	CHAR	Company name	148-187
ADDR1	40	CHAR	Company address line 1	188-227
ADDR2	40	CHAR	Company address line 2	228-267
PHONE	25	CHAR	Contact phone	268-292
CONTACT	30	CHAR	Contact name	293-322
SNDPASS	14	CHAR	Interchange send password	323-336
RCVPASS	14	CHAR	Interchange receive password	337-350
SECPROF	8	CHAR	Security profile ID	351-358
NETCLS	1	CHAR	Network message class	359-359
NETCHG	1	CHAR	Network charges code	360-360
NETACK	1	CHAR	Network acknowledgment	361-361
NETVCHK	1	CHAR	Destination verification code	362-362
NETRETN	3	CHAR	Retention period	363-365
NETEDIO	1	CHAR	EDI receive option	366-366
NETEDIP	1	CHAR	EDI processing override	367-367
STGFMTOV	1	CHAR	Storage format override	368-368
MACHTYPE	1	CHAR	Machine type	369-369
STGFMT	1	CHAR	Storage format	370-370
EOTID	1	CHAR	End of Text/Message delimiter	371-371
LOGENV	1	CHAR	Log standard data	372-372
FNCGRP	1	CHAR	Functional group code	373-373
SEDLM	2	CHAR	Subelement delimiter	374-375
DEDLM	2	CHAR	Data element delimiter	376-377
SEGDLM	2	CHAR	Segment delimiter	378-379
SEGSEP	2	CHAR	Segment ID separator	380-381
DECNOT	1	CHAR	Decimal notation	382-382
RLSCHAR	2	CHAR	Release character	383-384
INTCTLNO	9	CHAR	Interchange mask	385-393
GRPCTLNO	9	CHAR	Group mask	394-402
TRXCTLNO	9	CHAR	Transaction mask	403-411
COMMENT1	40	CHAR	Comment line 1	412-451
COMMENT2	40	CHAR	Comment line 2	452-491
NETCMDS	8	CHAR	Network commands file	492-499
TPDATAINE	32	CHAR	TP data phone number	500-531
TIMEOUT	4	DEC	Data line communication timeout	532-535
SEGMENTED	1	CHAR	Segmented output option	536-536
SUFFIX	2	CHAR	File suffix	537-538

Table 2-6 (Page 3 of 3). Trading Partner Profile Members

Tag	Length	Type	Description	Fixed Position
TPENVSUF	2	CHAR	Envelope profile suffix	539-540
TPGENRCV	1	CHAR	Allow generic receive	541-541
TPCMPRES	1	CHAR	Compression	542-542
TPSUPAD3	40	CHAR	Address line 3	543-582
TPSUPCTY	30	CHAR	City name	583-612
TPSUPST	2	CHAR	State code	613-614
TPSUPPST	15	CHAR	Postal code	615-629
TPSUPCON	30	CHAR	Country	630-659
TPSUPFAX	25	CHAR	Fax number	660-684
TPSUPU3	40	CHAR	Comment line 3	685-724
TPSUPU4	40	CHAR	Comment line 4	725-764
TPSUPU5	40	CHAR	Comment line 5	765-804
TPSUPU6	40	CHAR	Comment line 6	805-844
TPSUPU7	40	CHAR	Comment line 7	845-884
TPSUPU8	40	CHAR	Comment line 8	885-924
TPSUPU9	40	CHAR	Comment line 9	925-964
TPSUPU10	40	CHAR	Comment line 10	965-1004
PRIORITY	1	CHAR	Priority	1005-1005
DESCRIPT	30	CHAR	Description	1006-1035
TPTYPE	1	CHAR	Trading partner type	1036-1036

The following shows a partial example of an Export/Import file data set that contains records for exporting/importing three trading partner profiles.

```

0C1SMITH 940419161616ENU010103000000
7P1PROFID(TPPROF) PROFDESC(Trading partner profile)
7P2PROFID(TPPROF) TPNICKN(IBM) NETID(IINR3) SYSQUAL(4) NETACK(N) PASSWORD(N WPASS)
7P4TPNICKN(IBM) APPRECID(JONES) APPRECQ(01) INTCTLNO(000010000)
GRPCTLNO(000015000) TRXCTLNO(000020000)
7P2PROFID(TPPROF) TPNICKN(TPT40) NETACK(Y) SYSQUAL(4) NETID(IINR5) PHONE(555-1234)
7P2PROFID(TPPROF) TPNICKN(VAN04PR) CONTACT(John Doe) SNDPASS(Y) RCVPASS(Y)
7P4TPNICKN(VAN04PR) APPRECID(SMITH) APPRECQ(02) APPDOCID(850) INTCTLNO(000030001)
GRPCTLNO(000040000) TRXCTLNO(000045000)
000

```

The above example shows how the 0C1 record is the first record in the Export/Import file, followed by profile data and a 000 record. The 7P1 and 7P2 fields are the record IDs (RECIIDs). They must be the first three bytes of the data record. In this example, the 7 indicates the category code, the P indicates the object being exported/imported (in this case, Profile), and the 1 and 2 are internal DataInterchange indicators.

Trading Partner Profile Member Field Descriptions

Category Code For trading partner profiles, this field is a 7.

Record Type For trading partner profiles, this field is P2.

Profile ID (PROFID) For trading partner profiles, this field is TPPROF.

Trading Partner Nickname (TPNICKN)

The name you use to refer to this trading partner. Use the same name throughout DataInterchange to refer to this trading partner by "nickname."

Network ID (NETID) The name that identifies the network used to communicate with this trading partner. It must match the name of a member of the network profile.

System Qualifier (SYSQUAL) For IBM Global Network users, enter an I if intersystem addressing is required for this trading partner (IBM Global Network reference: DTBLTYP). Enter the ID of the other system in the next field.

System ID (SYSID) For intersystem addressing, the ID of the system responsible for the receiver's account. For IBM Global Network users, the ID is limited to three characters (IBM Global Network reference: DTBLID).

Account Number (ACCTID) The account number assigned to your trading partner by the network. The entry must be left-justified. For sending and receiving EDI in ISA/IEA envelopes, the last position must be blank. The combination of this field and the User ID field must be unique for all members in the profile.

If the interchange ID is blank, the account number and user ID make up the receiver (recipient) ID in the interchange envelope. UCS (BG/EG) envelopes are an exception. For these, phone number contains the receiver ID.

User ID (USERID) The user ID assigned to your trading partner by the network. The entry must be left-justified. The combination of this field and the account number must be unique for all members in the profile.

If the interchange ID is blank, the account number and user ID make up the receiver (recipient) ID in the interchange envelope. UCS (BG/EG) envelopes are an exception. For these, the phone number contains the receiver ID.

Interchange ID Qualifier (INTQUAL)

A code that indicates the type of interchange ID used in the interchange ID field. A Dun and Bradstreet (DUNS) number is one example; the EDI standard defines these codes. If the interchange ID (next field) is blank, the enveloper takes the qualifier from the envelope profile member. The combination of this field and the interchange ID field must be unique for each member unless they are blank.

Interchange ID (INTID) The ID used to fill in the interchange receiver (recipient) when you send to this partner and to identify the interchange sender when you receive from this partner. If this field is blank, the enveloper uses the account number and user ID (or phone number for BG/EG interchanges). The combination of this field and the interchange ID field must be unique for each member unless they are blank.

Company Name (CONAME) The name of the trading partner's company.

Address Line 1 (ADDR1)	Line 1 of the trading partner's address.														
Address Line 2 (ADDR2)	Line 2 of the trading partner's address.														
Contact Phone (PHONE)	The trading partner's phone number (free-form). For UCS (BG/EG) enveloping, if the interchange ID is blank, the enveloper uses the phone number as the interchange ID.														
Contact Name (CONTACT)	The name of the person you talk to when dealing with this trading partner.														
Interchange Send Password (SNDPASS)	The password agreed upon by you and your trading partner for sending to this trading partner. This value maps to the password data type in the interchange envelope.														
Interchange Receive Password (RCVPASS)	The password agreed upon by you and your trading partner for receiving from this trading partner. If this value matches the interchange password (indicated by the password data type) that was received, translation occurs.														
Security Profile ID (SECPROF)	The ID of the default security profile member that specifies the encryption and authentication processes that apply to EDI data for this trading partner. This profile member is always used when receiving from this partner. It is used by default when sending to this partner unless the send usage for the transaction specifies a different member in the group security profile member name field or the trans security profile member name field.														
Network Message Class (NETCLS)	<p>A code that indicates any special status (for example, test) of the data being sent. It does not apply to receiving. For IBM Global Network users (IBM Global Network reference: MSGNCLS), T indicates test status, and a blank indicates normal status.</p> <p>If your request does not specify a trading partner, the requestor profile provides this information.</p>														
Network Charges Code (NETCHG)	<p>A code that indicates how charges are shared between sender and receiver. For IBM Global Network (IBM Global Network reference: MSGCHRG), the codes are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>1</td><td>Receiver pays all charges.</td></tr> <tr> <td>2</td><td>Receiver pays all charges if agreed to; otherwise, charges are split between sender's and receiver's domains.</td></tr> <tr> <td>3</td><td>Receiver pays all charges if agreed to; otherwise, charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges.</td></tr> <tr> <td>4</td><td>Charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges.</td></tr> <tr> <td>5</td><td>Charges are split between sender's and receiver's domains.</td></tr> <tr> <td>6</td><td>Sender pays all charges.</td></tr> </table>	Value	Description	1	Receiver pays all charges.	2	Receiver pays all charges if agreed to; otherwise, charges are split between sender's and receiver's domains.	3	Receiver pays all charges if agreed to; otherwise, charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges.	4	Charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges.	5	Charges are split between sender's and receiver's domains.	6	Sender pays all charges.
Value	Description														
1	Receiver pays all charges.														
2	Receiver pays all charges if agreed to; otherwise, charges are split between sender's and receiver's domains.														
3	Receiver pays all charges if agreed to; otherwise, charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges.														
4	Charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges.														
5	Charges are split between sender's and receiver's domains.														
6	Sender pays all charges.														

If your request does not specify a trading partner, the requestor profile provides this information.

Network Acknowledgment (NETACK)

A code that indicates which network acknowledgments (receipt, delivery, purge) you want to receive when sending to this partner. For IBM Global Network users (IBM Global Network reference: MSGRCPT), the codes are:

Value	Description
(blank)	To request no acknowledgments
R	To request receipt acknowledgments only
D	To request delivery acknowledgments only
B	To request both receipt and delivery acknowledgments
A	To request purge acknowledgments only
C	To request both receipt and purge acknowledgments
E	To request either purge or delivery acknowledgments
F	To request receipt acknowledgments and either delivery or purge acknowledgments

Your choice here also applies for documents you send using the Interactive Entry Facility (IEF) unless you override the choice when defining the IEF document.

If your request does not specify a trading partner, the requestor profile provides this information.

Destination Verification Code (NETVCHK)

A code that indicates whether or not the destination is to be verified before sending occurs. For IBM Global Network users (IBM Global Network reference: MSGVCHK), the codes are:

Value	Description
N	To request no verification (the default)
Y	To require verification
F	To request verification and sending even if the destination is not verified (useful for intersystem addressing)

If your request does not specify a trading partner, the requestor profile provides this information.

Retention Period (NETRETN)

The number of days that the data is to be kept in the network mailbox before it is purged, if it is not received.

Because you may be using one or more networks, you will need to contact a representative from the network you are using or refer to an appropriate manual to determine the valid values for this field. The text that follows pertains to the IBM Global Network and Expedite Base 4.2 and higher.

Valid (RETAIN) values are 0 through 180. For installations within the U.S., the default retention period is 30 days. For installations outside the U.S., contact your marketing representative for your default value.

If you specify 0 or blank, Information Exchange retains the data for the default period. In addition, if you specify a value larger than the maximum for your system, Information Exchange retains the data for the default period.

If your request does not specify a trading partner, the requestor profile provides this information.

Note: The size of this field for Expedite/CICS is two digits, meaning the valid values for Expedite/CICS are 0 through 99. The two left-most digits entered in this field are used by Expedite/CICS as the retention period. For example: if you enter 180, the retention period value sent to Expedite/CICS will be 18.

EDI Receive Option (NETEDIO)

A code that indicates whether or not you want EDI segments to be stored in the file as separate records. For IBM Global Network users (IBM Global Network reference: EDIOPT), the codes are:

Value	Description
--------------	--------------------

Y	End records at the segment delimiter (the default)
---	--

N	Do not end records at the segment delimiter
---	---

If your request does not specify a trading partner, the requestor profile provides this information.

EDI Processing Override (NETEDIP)

A code that indicates whether or not you want EDI data you receive to have special EDI processing, which breaks records by the segment delimiter. For IBM Global Network users (IBM Global Network reference: EDIPROC), the codes are:

Value	Description
--------------	--------------------

Y	To perform EDI processing if the common data header indicates that the data is in standard format (the default)
---	---

N	To omit EDI processing, regardless of the common data header
---	--

If your request does not specify a trading partner, the requestor profile provides this information.

Storage Format Override (STGFMTOV)

Because you may be using one or more networks, you will need to contact a representative from the network you are using to determine the available options.

If you are using the IBM Global Network, common data header control information is used by IBM Global Network Information Exchange and allows users to send or receive information electronically. It contains information such as the type of record, original record format, sending system type, type of data being sent, whether or not the data is in an EDI format, and a unique record number for tracking.

If you are using IBM Expedite/MVS Base Version 1.1 (IEBASE), refer to the DLMOVERRIDE command option keyword for an up-to-date list of acceptable values. For example:

Value	Description
Y	Is used to format the data according to the DELIMITED parameter, even if the common data header(CDH) indicates a delimiter type.
N	This is the default. If the common data header(CDH) indicates a record type, the data will be formatted according to the CDH.

If you are using IBM Expedite/MVS Host Version 1.3 (TPMAIN) refer to the STGFORMO command option keyword for an up-to-date list of acceptable values. For example:

Value	Description
Y	This is the default and is to store the format as defined in the common data header defined by the IBM Global Network.
N	Use this to ignore the common data header and use the value you specified in the storage format field.

If there is no common data header, the format indicated in the Storage format field is used. If your request does not specify a trading partner, the requestor profile provides this information.

Machine Type (MACHTYPE)

A '1' indicates your trading partner is using the Personal Computer/Information Exchange (PC/IE) product to receive your data. If not, this value is a (blank).

Storage Format (STGFMT)

A code that indicates to the network how data is stored for free-form messages and files. The type of data you want to send and how the file will be received should be considered when determining what option to select. Because you may be using one or more networks, you will need to contact a representative from the network you are using to determine the available options.

Value	Description
C	Stores each record with a carriage return and line-feed character (CRLF) and uses the end of file (EOF) character to mark the end of file. These characters are represented and stored as hex values 0D0A and 1A respectively. Sending a file containing program source code which is defined with variable length records is the type of file generally sent with this option. Output records will include up to the carriage return and line feed characters (CRLF).
L	Indicates that each record should be preceded by two bytes which will contain the record length. This value is represented and stored as a hex value. You would probably select this option when sending as fixed format. This option is also appropriate if you are sending binary data. The output record is determined by the value in the two bytes containing the record length.
N	This is the default and is used to store the data as it is received. Output records are built based on the record length of the data set allocated to receive the data.

If you are using Expedite/CICS, refer to the DTYPE field for an up-to-date list of acceptable values. For example:

Value	Description
A	Stores each record with a carriage return and line-feed character (CRLF) and uses the end of file (EOF) character to mark the end of file. These characters are represented and stored as hex values 0D0A and 1A respectively. A file containing program source code defined with variable length records is the type of file generally sent with this option. Output records will include up to the carriage return and line feed characters (CRLF).
B	Indicates that each record should be preceded by two bytes which will contain the record length. This value is represented and stored as a hex value. You would probably select this option when sending as fixed format. This option is also appropriate if you are sending binary data. The output record is determined by the value in the two bytes containing the record length.
O	Other. Free format.

If you are using IBM Expedite/MVS Host Version 1.3 (TPMAIN), refer to the STGFORMO command option keyword for an up-to-date list of acceptable values. For example:

Value	Description
A	Stores each record with a carriage return and line-feed character (CRLF) and uses the end of file (EOF) character to mark the end of file. These characters are represented and stored as hex values 0D0A and 1A respectively. A file containing program source code defined with variable length records is the type of file generally sent with this option. Output records will include up to the carriage return and line feed characters (CRLF).
B	Indicates that each record should be preceded by two bytes which will contain the record length. This value is represented and stored as a hex value. You would probably select this option when sending a data set defined as fixed format. This option is also appropriate if you are sending binary data. The output record is determined by the value in the two bytes containing the record length.
C	This is the default and is used to store the data as it is received. Output records are built based on the record length of the data set allocated to receive the data.

If your request does not specify a trading partner, the information is taken from the requestor profile.

End of Text or Message Delimiter (EOTID)

The character that signifies to the network the end of the data. It applies to free-form messages and data files, not to standard transactions. (IBM Global Network reference: EORCHAR, EOMIND).

Log Standard Data (LOGENV) A code (Y or N) that tells the translator whether or not to include standard (untranslated) data when logging an image of a transaction. For EDI data with errors, the standard data is logged even if you enter N.

This field can be overridden by the Log standard data field of the APPDEFS profile.

Functional Group Code (FNCGRP)

A code (Y or N) that tells the enveloper whether or not to create functional groups for transactions with type E envelopes. (Functional groups are always created for types I, U, and X; they are never created for type T). If you choose not to have functional groups, different message types are enveloped together.

Subelement Delimiter (SEDLM)

The character (in hex notation) that separates subelements (also called component data elements) in a transaction set. An entry here (other than 00 or 40) overrides the character specified by the standard. If entered, this character must be different from the characters specified for the data element delimiter, segment delimiter, segment ID separator, decimal notation, and release character fields, with this exception: the subelement delimiter and the data element delimiter can be the same for type I, U, and X envelopes.

Data Element Delimiter (DEDLM)

The character (in hex notation) that separates the data elements in a transaction set. An entry here (other than 00 or 40) overrides the character specified by the standard.

If entered, this character must be different from the characters specified for the subelement delimiter, segment delimiter, segment ID separator, decimal notation, and release character fields, with this exception: the subelement delimiter and the data element delimiter can be the same for type I, U, and X envelopes.

Segment Delimiter (SEGDLM) The character (in hex notation) that marks the end of each segment in a transaction set. An entry here (other than 00 or 40) overrides the character specified by the standard.

If entered, this character must be different from the characters specified for the subelement delimiter, data element delimiter, segment ID separator, decimal notation, and release character fields.

Segment ID Separator (SEGSEP)

The character (in hex notation) that separates the segment ID and the first data element in a segment. An entry here (other than 00 or 40) overrides the character specified by the standard.

If entered, this character must be different from the characters specified for the subelement delimiter, data element delimiter, segment delimiter, decimal notation, and release character fields.

Decimal Notation (DECNOT) The character that represents the decimal point in a transaction set. For type E envelopes, an entry here overrides the character specified by the standard. For all other types, a period represents the decimal point.

If entered, this character must be different from the characters specified for the subelement delimiter, data element delimiter, segment delimiter, segment ID separator, and release character fields.

Release Character (RLSCHAR)

For type E and T envelopes, the character (in hex notation) that is used to indicate where a delimiter is being used as part of the data. An entry here (other than 00 or 40) overrides the character specified by the standard.

If entered, this character must be different from the characters specified for the subelement delimiter, data element delimiter, segment delimiter, segment ID separator, and decimal notation fields.

Interchange Mask (INTCTLNO)

The initial reference number that the enveloper maps to the CN data type in the interchange header and trailer. This value will be used as the base value for each trading partner, receiver ID combination. This field does not represent the current control number for this trading partner. Use the control number action from the member list panel to request control number information.

Group Mask (GRPCTLNO)

The initial reference number or special codes that the enveloper maps to the CN data type in the functional group header and trailer. This value will be used as the base value for each trading partner, receiver ID combination. Use caution when updating this field. This value does not represent the current control number for this trading partner. Use the control number action from the member list panel to request control number information.

Transaction Mask (TRXCTLNO)

The initial reference number or special codes that the enveloper maps to the CN data type in the transaction set header and trailer. This value will be used as the base value for each trading partner, receiver ID combination. Use caution when updating this field. This value does not represent the current control number for this trading partner. Use the control number action from the member list panel to request control number information.

Comment Line 1 (COMMENT1)

Enter any free-form notes about the trading partner.

Comment Line 2 (COMMENT2)

Enter any free-form notes about the trading partner.

Network Commands File (NETCMDS)

The name of a member of a Partitioned Data Set (PDS) that will be allocated to the MVS DDname of EDINTCMD. This member will contain the commands that you want to pass to the network (i.e., the TI Gateway).

DataInterchange will read the commands from the PDS member and write the commands to the Network input file specified in the network profile member after all substitutable variable tags have been resolved by DataInterchange.

TP Data Phone Number (TPDATALINE)

The phone number to dial to connect to your trading partner directly. This is the number needed by your computer to talk directly to your trading partner's computer. See the Contact phone for the voice number if you want to call a human (not a computer) at your trading partner's site.

Data Line Communication Timeout (TIMEOUT)

A value which will be used as a maximum allowable time that the communications line can be idle without being dropped. If the data line is idle for a time greater than the value entered, then the line will be dropped. If you specify a trading partner when requesting network activity, the value for this field is taken from the trading partner profile. Otherwise, the value for this field is taken from the requestor profile.

Segmented Output Option (SEGMENTED)

A code that indicates whether or not you want EDI segments to be stored in the output file as separate records.

Value	Description
--------------	--------------------

Y	To end records at the segment delimiter
---	---

N	To not end at the segment delimiter (the default)
---	---

File Suffix (SUFFIX)

A 2 character suffix may be entered which will be used as a suffix for the ddname used to store the results of a fixed-to-fixed translation. The basic part of the ddname is taken from the Application file name field of the target application data format.

This 2 character suffix will be appended to the Application file name or will overlay the last two characters of the Application file name if that value exceeds 6 characters. The suffix is provided so that results of a fixed-to-fixed translation can be separated by trading partner. If this separation is not wanted, then either the suffix can be left blank or all the ddnames can be assigned to the same physical file.

In a CICS environment the suffix can be used to identify a unique TS queue for each trading partner.

Envelope Profile Suffix (TPENVSUF)

A 2 character suffix may be entered which will be used as a suffix for a generic standard envelope profile member name. A generic envelope profile name is one that has an ampersand (&) as the first character of the name followed by a 1 to 6 character common name.

This 2 character suffix will be appended to the 1 to 6 character common name to generate the envelope profile member name to be used when enveloping transactions. The suffix is provided so that different standard envelope profile members can be used for different trading partners. To use the same envelope profile member for several trading partners, enter the same suffix in each of the TP profiles, or leave the suffix blank to use a common envelope profile member (with the same name as the common name in the usage). An envelope profile member must be defined for each combination of common name plus suffix value specified in a TP profile.

The standard envelope profile member name is specified on the send and receive TP usage panels. The generic form of the name is most useful for generic TP usages but can be used for any usage.

Allow Generic Receive (TPGENRCV)

A code that indicates whether or not you want to allow transactions received from this trading partner to be translated using generic receive TP usages.

	Value	Description
	Y	Allow translation via generic receive TP usages
	N	Do not allow translation via generic usages (the default)
Compression (TPCMPRES)	A code that indicates whether or not you want Expedite Base/MVS or Expedite/CICS to compress your data prior to transmission.	
	Value	Description
	N	Data is not to be compressed (the default)
	Y	Data is to be compressed prior to transmission. If the file to be sent contains multiple EDI envelopes, all envelopes are sent compressed.
	T	Data is conditionally compressed prior to transmission. For Expedite Base/MVS, compression will occur if the SENDER, RECEIVER, and COMPRESS parameters are listed in the CPLOOKUP EXPFILE file. This file defines a series of paired receivers and senders, and for each pair indicates whether compression should be performed. For Expedite/CICS, compression will occur if the sender/receiver pair is found in the lookup table and Compress=Y in the table.
	For additional information, see Appendix F in the <i>Expedite Base/MVS Programming Guide</i> or Chapter 6 in the <i>Customizing and Developing Applications with Expedite/CICS</i> manual.	
	Note: The compression value will come from the Trading Partner Profile, if a Trading Partner Profile member is used on the SEND. Otherwise, the compression value will come from the Requestor Profile.	
Output file (TPOUTFIL)	Reserved.	
Address Line 3 (TPSUPAD3)	Line 3 of the trading partner's address.	
City Name (TPSUPCTY)	The trading partner's city.	
State Code (TPSUPST)	The trading partner's state.	
Postal Code (TPSUPPST)	The trading partner's zip code.	
Country (TPSUPCON)	The trading partner's country.	
Fax Number (TPSUPFAX)	The trading partner's fax number.	
Comment Line 3 (TPSUPU3)	Use this area for remarks.	
Comment Line 4 (TPSUPU4)	Use this area for remarks.	
Comment Line 5 (TPSUPU5)	Use this area for remarks.	
Comment Line 6 (TPSUPU6)	Use this area for remarks.	
Comment Line 7 (TPSUPU7)	Use this area for remarks.	
Comment Line 8 (TPSUPU8)	Use this area for remarks.	
Comment Line 9 (TPSUPU9)	Use this area for remarks.	
Comment Line 10 (TPSUPU10)	Use this area for remarks.	

Priority (PRIORITY)	<p>Enter a blank (for normal priority) or a P (for high priority). These codes are used by Expedite Base/MVS and Expedite/CICS to prioritize delivery of messages. High priority messages will be delivered to your trading partners before normal priority messages.</p> <p>For additional information, see the SEND commands in the <i>Expedite Base/MVS Programming Guide</i> or Chapter 6 in the <i>Customizing and Developing Applications with Expedite/CICS</i> manual.</p> <p>Note: The priority value will come from the Trading Partner Profile, if a Trading Partner Profile member is used on the SEND. Otherwise, the priority value will come from the Requestor Profile.</p>								
Description (DESCRIPT)	A free-form field that describes the member.								
Trading Partner Type (TPTYPE)	<p>Enter a value that indicates the type of trading partner this definition represents. Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>E</td><td> <p>EDI trading partner</p> <p>An EDI trading partner is a trading partner considered to be external to your business organization. The interchange control numbers are generated using the EDI trading partner. This type of trading partner is the default and will have no effect on your current trading partner setup.</p> </td></tr> <tr> <td>A</td><td> <p>Application trading partner</p> <p>An application trading partner is considered an internal trading partner within your business organization. For example, divisions or departments may be internal trading partners. The application trading partner may be used when no specific EDI trading partner is defined or in combination with the EDI trading partner. The interchange control numbers are generated using the application and EDI trading partner combination. This type of trading partner should be used with centralized EDI when the application trading partners do business with the same EDI trading partner.</p> </td></tr> <tr> <td>B</td><td> <p>Both EDI trading partner and application trading partner</p> <p>This trading partner is considered both an external (EDI) and internal (application) trading partner. This type of trading partner should be used when your organization provides EDI translation services to customers who are trading partners. The value of this field is directly related to the trading partner usage setup.</p> <p>More information may be found in the <i>DataInterchange Administrator's Guide</i>.</p> </td></tr> </table>	Value	Description	E	<p>EDI trading partner</p> <p>An EDI trading partner is a trading partner considered to be external to your business organization. The interchange control numbers are generated using the EDI trading partner. This type of trading partner is the default and will have no effect on your current trading partner setup.</p>	A	<p>Application trading partner</p> <p>An application trading partner is considered an internal trading partner within your business organization. For example, divisions or departments may be internal trading partners. The application trading partner may be used when no specific EDI trading partner is defined or in combination with the EDI trading partner. The interchange control numbers are generated using the application and EDI trading partner combination. This type of trading partner should be used with centralized EDI when the application trading partners do business with the same EDI trading partner.</p>	B	<p>Both EDI trading partner and application trading partner</p> <p>This trading partner is considered both an external (EDI) and internal (application) trading partner. This type of trading partner should be used when your organization provides EDI translation services to customers who are trading partners. The value of this field is directly related to the trading partner usage setup.</p> <p>More information may be found in the <i>DataInterchange Administrator's Guide</i>.</p>
Value	Description								
E	<p>EDI trading partner</p> <p>An EDI trading partner is a trading partner considered to be external to your business organization. The interchange control numbers are generated using the EDI trading partner. This type of trading partner is the default and will have no effect on your current trading partner setup.</p>								
A	<p>Application trading partner</p> <p>An application trading partner is considered an internal trading partner within your business organization. For example, divisions or departments may be internal trading partners. The application trading partner may be used when no specific EDI trading partner is defined or in combination with the EDI trading partner. The interchange control numbers are generated using the application and EDI trading partner combination. This type of trading partner should be used with centralized EDI when the application trading partners do business with the same EDI trading partner.</p>								
B	<p>Both EDI trading partner and application trading partner</p> <p>This trading partner is considered both an external (EDI) and internal (application) trading partner. This type of trading partner should be used when your organization provides EDI translation services to customers who are trading partners. The value of this field is directly related to the trading partner usage setup.</p> <p>More information may be found in the <i>DataInterchange Administrator's Guide</i>.</p>								

TP Contacts Definition (7P3)

The TP contacts are exported or imported automatically when a Trading Partner is exported or imported. Each TP contact record specifies a contact that is related to a Trading Partner. Table 2-7 on page 2-33 describes the tags and field lengths.

Table 2-7. TP Contacts Definition

Tag	Length	Type	Description	Fixed Position
TPNICKN	16	CHAR	TP Nickname	004-019
TPCONNM	40	CHAR	TP Contact Name	020-059
TPCONSEQ	11	CHAR	TP Contact Sequence number for TP	060-070
TPCONLBL	30	CHAR	TP Contact Label	071-100

TP Control Numbers (7P4)

The TP control numbers are exported or imported automatically when a Trading Partner is exported or imported. Each TP control number specifies a control number that is related to a Trading Partner. Table 2-8 describes the tags and field lengths.

Table 2-8. TP Control Numbers (7P4)

Tag	Length	Type	Description	Fixed Position
TPNICKN	16	CHAR	TP Nickname	004-019
APPRECID	35	CHAR	Application Receiver ID	020-054
APPRECQ	4	CHAR	Application Receiver Qualifier	055-058
APPDOCID	8	CHAR	Transaction ID or Message ID	059-066
INTCTLNO	9	CHAR	Interchange Control Number	067-075
GRPCTLNO	9	CHAR	Group Control Number	076-084
TRXCTLNO	9	CHAR	Transaction Control Number	085-093

Comments Definition (7A1)

The comments are exported or imported automatically when a Trading Partner is exported or imported. The comment record specifies textual data about a Trading Partner. Table 2-9 describes the tags and field lengths.

Table 2-9. Comments Definition

Tag	Length	Type	Description	Fixed Position
CMTTYPE	8	CHAR	Comments Entity Type	004-011
CMTKEY	35	CHAR	Comments Entity Type Key	012-046
CMTTEXT	2048	CHAR	TP Contact Comments Text	397-2444

Contacts Definition (7Z1)

The contacts are exported or imported automatically when a Trading Partner is exported or imported. The contact record specifies data about a contact. Table 2-10 describes the tags and field lengths.

Table 2-10 (Page 1 of 2). Contacts Definition

Tag	Length	Type	Description	Fixed Position
TPCONNM	40	CHAR	TP Contact Name	004-043
TPCONTLE	40	CHAR	TP Contact Job Title	044-083

Table 2-10 (Page 2 of 2). Contacts Definition

Tag	Length	Type	Description	Fixed Position
TPCONAD1	40	CHAR	TP Contact Address Line 1	084-123
TPCONAD2	40	CHAR	TP Contact Address Line 2	124-163
TPCONAD3	40	CHAR	TP Contact Address Line 3	164-203
TPCONCTY	30	CHAR	TP Contact City	204-233
TPCONST	2	CHAR	TP Contact State	234-235
TPCONPST	15	CHAR	TP Contact Postal Code	236-250
TPCONCON	30	CHAR	TP Contact Country	251-280
TPCONPHN	25	CHAR	TP Contact Phone	281-305
TPCONFAX	25	CHAR	TP Contact Fax Number	306-330
TPCONOTH	25	CHAR	TP Contact Other Phone	331-355
TPCONEML	40	CHAR	TP Contact E-mail Address	356-395
TPCONTYP	1	CHAR	TP Contact Type Code	396-396
CMTTEXT	2048	CHAR	TP Contact Comments Text	397-2444

Export/Import of Standards Records

EDI Standards can be exported/imported using tagged or fixed format files. The Standard/Standard Transaction set includes:

S1 or E1	Standard Definition or Envelope Definition
S2	Transaction Definition
S3	Segment Usage
S4 or E4	Segment Definition or Envelope Definition
S5 or E5	Data Element Usage or Envelope Data Element Usage
S6 or E6	Data Element Definition or Envelope Data Element Definition

There are three types of associated objects that can be exported using tags:

1. validation tables
2. envelope profile
3. envelope standard

In the example below, you can export all of the data elements within a particular segment. Since a segment is made up of many data element usages and definitions, multiple S5 and S6 records are shown here.

```

00C1.....etc. (E/I Common Control Record)
1S1.....etc. (Standard Definition Record)
1S2.....etc. (Standard Transaction Definition Record)
1S3.....etc. (Standard Segment Usage)
1S4.....etc. (Standard Segment Definition)
1S5.....etc. (Standard Data Element Usage)
1S6.....etc. (Standard Data Element Definition)
1S5.....etc. (Standard Data Element Usage)
1S6.....etc. (Standard Data Element Definition)
000 (E/I Common End of Group Record)

```

Standard/Envelope Definition (1S1/1E1)

The standard/envelope definition defines the standard or envelope being used. Table 2-11 on page 2-35 lists the tags and field lengths.

Table 2-11. Standard/Envelope Definition

Tag	Length	Type	Description	Fixed Position
STDID	8	CHAR	Standard (or envelope) ID	004-011
VER	2	CHAR	Version of standard	012-013
REL	2	CHAR	Level within release of standard	014-015
ENVTYPE	1	CHAR	Default enveloping for the standard	016-016
DESC	50	CHAR	Description of standard	017-066
GROUPOPT	1	CHAR	Functional group optional (Y/N)	067-067
DEDLM	2	CHAR	Element delimiter	068-069
SEDLM	2	CHAR	Subelement delimiter	070-071
SEGTRM	2	CHAR	Segment Terminator	072-073
DECNOT	1	CHAR	Decimal notation	074-074
RLSCHAR	2	CHAR	Release character	075-076
SEGDLM	2	CHAR	Segment delimiter (e.g. = for LIN=)	077-078
DLMSEGID	8	CHAR	Delimiter segment ID (e.g. UNA)	079-086
INTERHDR	8	CHAR	Interchange header segment ID	087-094
GROUHDR	8	CHAR	Group header segment ID	095-102
TRANSHDR	8	CHAR	Transaction header segment ID	103-110
TRANSTRL	8	CHAR	Transaction trailer segment ID	111-118
GROUPTRL	8	CHAR	Group trailer segment ID	119-126
INTERTRL	8	CHAR	Interchange trailer segment ID	127-134
ALLTRXS	1	CHAR	All transactions? (Y/N)	135-135

Standard Transaction Definition (1S2)

The tags available for the standard transaction definition and the associated field lengths are specified in Table 2-12.

Table 2-12. Standard Transaction Definition

Tag	Length	Type	Description	Fixed Position
STDID	8	CHAR	Standard ID	004-011
STDTRID	8	CHAR	Standard Transaction ID	012-019
FNGRPID	8	CHAR	Functional Group ID	020-027
DESC	50	CHAR	Description of the Transaction	028-077
FNCACK	1	CHAR	Functional Acknowledgment Type (Unused)	078-078

Standard Segment Usage (1S3)

The tags available for the standard segment usages are defined in Table 2-13 on page 2-36. If you are importing a transaction set, you can have multiple segment usage records defined.

Table 2-13. Standard Segment Usage

Tag	Length	Type	Description	Fixed Position
STDID	8	CHAR	Standard ID	004-011
STDTRID	8	CHAR	Standard Transaction ID	012-019
SEGID	8	CHAR	Segment ID	020-027
SEQNO	3	CHAR	Sequence Number in standard	028-030
SEGREQ	1	CHAR	Required/Mandatory/Optional/Float Indicator	031-031
MAXUSE	6	DEC	Maximum Use (Limit)	032-037
LOOPID	6	CHAR	Loop Control ID	038-043
LOOPREP	11	DEC	Loop Repeat	044-054

Standard/Envelope Segment Definition (1S4/1E4)

The standard segment definition and standard envelope definition tags are listed in Table 2-14.

Table 2-14. Standard/Envelope Segment Definition

Tag	Length	Type	Description	Fixed Position
STDID	8	CHAR	Standard ID	004-011
SEGID	8	CHAR	Segment ID	012-019
DESC	50	CHAR	Description of the Standard Segment	020-069

Standard/Envelope Data Element Usages (1S5/1E5)

The standard data element usage and standard envelope element usage tags are listed in Table 2-15.

Table 2-15 (Page 1 of 2). Standard/Envelope Data Element Usages

Tag	Length	Type	Description	Fixed Position
STDID	8	CHAR	Standard ID	004-011
SEGID	8	CHAR	Segment ID	012-019
DEID	8	CHAR	Data Element ID	020-027
SEQNO	6	CHAR	Sequence Number (6 numeric characters)	028-033
DEREQ	1	CHAR	Data Element Requirement	034-034
DEREL	1	CHAR	Related Data Element Flag	035-035
DERNO1	6	CHAR	Data Element Reference ID 1	036-041
DERNO2	6	CHAR	Data Element Reference ID 2	042-047
DERNO3	6	CHAR	Data Element Reference ID 3	048-053
DERNO4	6	CHAR	Data Element Reference ID 4	054-059

Table 2-15 (Page 2 of 2). Standard/Envelope Data Element Usages

Tag	Length	Type	Description	Fixed Position
DERNO5	6	CHAR	Data Element Reference ID 5	060-065

Standard/Envelope Data Element Definition (1S6/1E6)

The standard data element definition and standard envelope element definition tags are listed in Table 2-16.

Table 2-16. Standard/Envelope Data Element Definition

Tag	Length	Type	Description	Fixed Position
STDID	8	CHAR	Standard ID	004-011
DEID	8	CHAR	Data Element ID	012-019
DESC	50	CHAR	Data Element Description	020-069
DETYPE	2	CHAR	Type of Data Element	070-071
DEMIN	6	DEC	Minimum Data Element Length	072-077
DEMAX	6	DEC	Maximum Data Element Length	078-083
DEVAL	16	CHAR	Name of Validation List	084-099

The following shows a partial example of an E/I file data set that contains records for exporting/importing one transaction of a standard, that has one segment with three data elements.

```
0C1SMITH 940419161616ENU010103000000
1S1STDID(AARV3R21) VER(03) REL(21) ENVTYPE(X) DESC(Rail Carrier Ind)
1S2STDID(AARV3R21) STDTRID(426) FNGRPID(SR) DESC(Rail Revenue Waybill)
1S3STDID(AARV3R21) STDTRID(426) SEGID(ZR) SEQNO(001) SEGREQ(M) MAX1234)
1S4STDID(AARV3R21) SEGID(ZR) DESC(Waybill Reference Identification)
1S5STDID(AARV3R21) SEGID(ZR) DEID(762) SEQNO(01) DEREQ(M)
1S6STDID(AARV3R21) DEID(762) DESC(Waybill Request Response Code)
1S5STDID(AARV3R21) SEGID(ZR) DEID(206) SEQNO(02) DEREQ(M)
1S6STDID(AARV3R21) DEID(206) DESC(Equipment Initial) DETYPE(AN)
1S5STDID(AARV3R21) SEGID(ZR) DEID(207) SEQNO(03) DEREQ(M)
1S6STDID(AARV3R21) DEID(207) DESC(Equipment Number) DETYPE(AN)
000
```

Export/Import of Application Data Formats

Application Data Formats can be exported/imported using tagged or fixed format files. The application data format set includes:

2F1 Application Data Format Definitions

2F2 Application Data Format Structure

There are no associated objects for application data formats.

Application Data Format Definitions (2F1)

The tags for application data format definitions are listed in Table 2-17 on page 2-38.

Table 2-17. Application Data Format Definition

Tag	Length	Type	Description	Fixed Position
FORMAT	16	CHAR	Application data format ID (key)	004-019
DESC	50	CHAR	Application data format description	020-069
BASESTR	16	CHAR	Base structure ID	070-085
FILEID	8	CHAR	Logical receiving data set name	086-093
FILETYPE	2	CHAR	Type of receive file (MQ, PG, TD, TM, TS, TX, VS)	094-095
DATE	8	CHAR	Last change date stamp	096-103
TIME	6	CHAR	Last change time stamp	104-109
RIDPOS	6	DEC	Record ID offset	110-115
RIDLENG	6	DEC	Record ID length	116-121
RIDTYPE	2	CHAR	Record ID data type	122-123
TPIDFLD	16	CHAR	Name of format field containing the internal trading partner	124-139
BEGSTRID	16	CHAR	Structure ID to signal the beginning of a transaction	140-155
ENDSTRID	16	CHAR	Structure ID to signal the end of a transaction	156-171
VARREC	1	CHAR	Reserved for future use	172-172
GENRTCDE	16	CHAR	Generic usages routing code field	173-188
ISIDQUAL	4	CHAR	Interchange sender ID qualifier	189-192
ISID	35	CHAR	Interchange sender ID	193-227
IRIDQUAL	4	CHAR	Interchange receiver ID qualifier	228-231
IRID	35	CHAR	Interchange receiver ID	232-266
APPLTPID	16	CHAR	Application trading partner ID	267-282
TPNICKN	16	CHAR	EDI trading partner ID	283-298

Application Data Format Structures (2F2)

The tags for application data format structure records are listed in the following tables.

Table 2-18 (Page 1 of 2). Application Data Format Structure Header

Tag	Length	Type	Description	Fixed Position
FORMAT	16	CHAR	Application data format ID	004-019
STRID	16	CHAR	Format structure ID	020-035
SEQNO	3	DEC	Sequence number (000)	036-038
RECID	16	CHAR	Record ID	039-054
STRTYPE	2	CHAR	Structure type (SH)	055-056
MAXUSE	6	DEC	Maximum use count	057-062

Table 2-19. Application Data Format Structure Entry

Tag	Length	Type	Description	Fixed Position
FORMAT	16	CHAR	Application data format ID	004-019
STRID	16	CHAR	Format structure ID	020-035
SEQNO	3	DEC	Sequence number	036-038
STRNAME	16	CHAR	Structure name	039-054
STRTYPE	2	CHAR	Structure type (ST)	055-056
PASSEP	6	DEC	Passed separately? (1 = No, 0 = Yes)	057-062

Table 2-20. Application Data Format Field Entry

Tag	Length	Type	Description	Fixed Position
FORMAT	16	CHAR	Application data format ID	004-019
STRID	16	CHAR	Format structure ID	020-035
SEQNO	3	DEC	Sequence number	036-038
FLDNAME	16	CHAR	Field name	039-054
DATATYPE	2	CHAR	Data type (other than SH or ST)	055-056
FLDLLEN	6	DEC	Field length	057-062

The following shows a partial example of an E/I file data set that contains records for exporting/importing one application data format with one structure that has three fields.

```
0C1SMITH 940419161616ENU010103000000
2F1FORMAT(AAAP012) BASESTR(LINEITEM) TIME(232654)
2F2FORMAT(AAAP012) STRID(LINEITEM) SEQNO(000) RECID(LIN) STRTYPE(SH)
2F2FORMAT(AAAP012) STRID(LINEITEM) SEQNO(001) FLDNAME(PARTNO)
2F2FORMAT(AAAP012) STRID(LINEITEM) SEQNO(002) FLDNAME(QUANTITY)
2F2FORMAT(AAAP012) STRID(LINEITEM) SEQNO(003) FLDNAME(PRICE)
000
```

Export/Import of Trading Partner Transactions

Trading Partner Transactions can be exported/imported using tagged or fixed format files. The trading partner transaction set includes:

- 3U1 Trading Partner Transaction Usage only; Map not included
- 3M1 Trading Partner Transaction Map with or without Usage
- 3T2 Trading Partner Mapping Segment
- 3T3 Trading Partner Mapping Data Element
- 3T4 Trading Partner Mapping Rules
- 3T5 Trading Partner Send Usage
- 3T6 Trading Partner Receive Usage

There are fifteen associated objects for exporting trading partner transactions. Either mapping or usages must be selected.

1. Transaction usages
2. Control string
3. Standard transaction
4. Application data format
5. Validation tables
6. Translation tables
7. User exit routines
8. Trading partner profile
9. Translation exit routines
10. Security profile
11. Network profile
12. Network operations
13. Envelope profile
14. Network standard
15. Mapping

Note: In trading partner-to-trading partner transactions, DataInterchange stores control numbers by ISA SenderQualifier, ISA ReceiverQualifier, and ISA Receiver. This ensures that the receiver gets a contiguous set of control numbers from every sender.

The following tables define the tags and lengths for exporting/importing trading partner transactions.

Trading Partner Transaction (M1/U1)

Table 2-21 (Page 1 of 2). Trading Partner Transaction

Tag	Length	Type	Description	Fixed Position
TPTID	16	CHAR	Mapping transaction ID	004-019
STDID	8	CHAR	Standard ID	020-027
STDTRID	8	CHAR	Standard transaction ID	028-035
FORMAT	16	CHAR	Application data format ID	036-051
DIR	1	CHAR	Send/receive flag (S/R)	052-052
GENREQD	1	CHAR	Generate required (Y/N)	053-053
DESC	46	CHAR	Mapping transaction description	054-099
ACFLD1	16	CHAR	First application control field name	100-115
ACLEN1	6	DEC	First post conversion substring length	116-121
ACFLD2	16	CHAR	Second application control field name	122-137
ACLEN2	6	DEC	Second post conversion substring length	138-143
ACFLD3	16	CHAR	Third application control field name	144-159
ACLEN3	6	DEC	Third post conversion substring length	160-165
ACFLD4	16	CHAR	Fourth application control field name	166-181
ACLEN4	6	DEC	Fourth post conversion substring length	182-187
ACFLD5	16	CHAR	Fifth application control field name	188-203
ACLEN5	6	DEC	Fifth post conversion substring length	204-209
ACFLD6	16	CHAR	Sixth application control field name	210-225

Table 2-21 (Page 2 of 2). Trading Partner Transaction

Tag	Length	Type	Description	Fixed Position
ACLEN6	6	DEC	Sixth post conversion substring length	226-231
ACFLD7	16	CHAR	Seventh application control field name	232-247
ACLEN7	6	DEC	Seventh post conversion substring length	248-253
ACFLD8	16	CHAR	Eighth application control field name	254-269
ACLEN8	6	DEC	Eighth post conversion substring length	270-275
TPTMAUI	1	CHAR	Type flag for mapping	276-276
COMPCHK	1	CHAR	Compliance flag for syntax checking	277-277

Trading Partner Mapping Segment (T2)

Table 2-22. Trading Partner Mapping Segment

Tag	Length	Type	Description	Fixed Position
TPTID	16	CHAR	Mapping Transaction ID	004-019
CURSGSEQ	11	DEC	Unique sequence to identify this segment in the mapping	020-030
NXTLPOCC	11	DEC	Unique sequence to identify first segment of repeated loop	031-041
NXTSGOCC	11	DEC	Unique sequence to identify repeated segment	042-052
FLTSGSEQ	11	DEC	Unique sequence to identify a copied floating segment	053-063
SELECTED	1	CHAR	Segment selected (Y/N)	064-064
LASTSEG	1	CHAR	Flag to identify last segment in group ('L' or blank)	065-065
STDID	8	CHAR	Standard ID	066-073
SEGID	8	CHAR	Standard transaction segment ID	074-081
SEQNO	3	DEC	Sequence number of segment in standard	082-084
SEGREQ	1	CHAR	Required/Mandatory/Optional/Floating indicator	085-085
MAXUSE	6	DEC	Maximum use count	086-091
LOOPID	6	CHAR	Loop ID	092-097
LOOPREP	11	DEC	Loop repeat count	098-108
LPQUATYP	1	CHAR	Loop occurrence qualifier type	109-109
LOOPQUAL	16	CHAR	Loop occurrence qualifier	110-125
SGQUATYP	1	CHAR	Segment occurrence qualifier type	126-126
SEGQUAL	16	CHAR	Segment occurrence qualifier	127-142
GENFLAG	1	CHAR	Generate segment (Y/N)	143-143

Trading Partner Mapping Data Element (T3)

Table 2-23. Trading Partner Mapping Data Element

Tag	Length	Type	Description	Fixed Position
TPTID	16	CHAR	Trading Partner Mapping Transaction ID	004-019
CURSGSEQ	11	DEC	Unique sequence to identify the segment in mapping	020-030
CURDESEQ	11	DEC	Unique sequence to identify the element in mapping	031-041
NXTDEOCC	11	DEC	Sequence to identify next occurrence of repeated element	042-052
STDID	8	CHAR	Standard ID	053-060
DEID	8	CHAR	Data element ID	061-068
SEQNO	6	CHAR	Standard element sequence (6 numeric characters)	069-074
DEREQ	1	CHAR	Data element requirement	075-075
DEREL	1	CHAR	Related element definition	076-076
DERNO1	6	CHAR	Related element 1 sequence	077-082
DERNO2	6	CHAR	Related element 2 sequence	083-088
DERNO3	6	CHAR	Related element 3 sequence	089-094
DERNO4	6	CHAR	Related element 4 sequence	095-100
DERNO5	6	CHAR	Related element 5 sequence	101-106
DETYPE	2	CHAR	Data type	107-108
DEMIN	6	DEC	Minimum length	109-114
DEMAX	6	DEC	Maximum length	115-120
DEREP	6	CHAR	Data element repetition count	121-126
SELECTED	1	CHAR	Selected (Y/N)	127-127
STRID	16	CHAR	Structure name to qualify the application field	128-143
FLDSTRN	16	CHAR	Application field or structure name associated with element	144-159
RULES	1	CHAR	Rules associated with mapping (Y/N)	160-160
LASTDE	1	CHAR	Flag to denote last element in group ('L' or blank)	161-161
QUADEOCC	11	DEC	Sequence of next qualified data element	162-172
QUADESEQ	11	DEC	Sequence of qualifying element	173-183

Trading Partner Mapping Rules (T4)

Table 2-24 (Page 1 of 2). Trading Partner Mapping Rules

Tag	Length	Type	Description	Fixed Position
TPTID	16	CHAR	Trading Partner Mapping Transaction ID	004-019

Table 2-24 (Page 2 of 2). Trading Partner Mapping Rules

Tag	Length	Type	Description	Fixed Position
CURSGSEQ	11	DEC	Unique sequence to identify seg in mapping	020-030
CURDESEQ	11	DEC	Unique sequence to identify element in mapping	031-041
TRNTAB	8	CHAR	Translation table name	042-049
USEREXIT	8	CHAR	User exit routine name	050-057
VALTAB	8	CHAR	Validation table name	058-065
DATEDIT	6	DEC	Date edit selection number	066-071
SUBPOS	6	DEC	Substring or concatenation position	072-077
SUBLEN	6	DEC	Substring or concatenation length	078-083
LITLEN	6	DEC	Length of literal text	084-089
LITVAL	80	CHAR	Literal text	090-169
QUALLEN	6	DEC	Length of qualifier text	170-175
QUALTEXT	45	CHAR	Character string for qualified mapping	176-220
ACCUM1	2	CHAR	First Accumulator ID (G0-G9, T0-T9)	221-222
ACTION1	2	CHAR	Action to perform on first accumulator	223-224
ACCUM2	2	CHAR	Second Accumulator ID (G0-G9, T0-T9)	225-226
ACTION2	2	CHAR	Action to perform on second accumulator	227-228
ACCUM3	2	CHAR	Third Accumulator ID (G0-G9, T0-T9)	229-230
ACTION3	2	CHAR	Action to perform on third accumulator	231-232
ACCUM4	2	CHAR	Fourth Accumulator ID (G0-G9, T0-T9)	233-234
ACTION4	2	CHAR	Action to perform on fourth accumulator	235-236

Trading Partner Send Usage (T5)

Table 2-25 (Page 1 of 2). Trading Partner Send Usage

Tag	Length	Type	Description	Fixed Position
INTPID	35	CHAR	Internal trading partner ID	004-038
FORMAT	16	CHAR	Application data format ID	039-054
TPTID	16	CHAR	Trading Partner Mapping Transaction ID	055-070
TPNICKN	16	CHAR	Trading partner nickname	071-086
ENVTYPE	1	CHAR	Standard envelope ID	087-087
STDPROF	8	CHAR	Standard profile member name	088-095
APPSNDID	35	CHAR	Application sender ID	096-130
APPRECID	35	CHAR	Application receiver ID	131-165
APPASSWD	14	CHAR	Application password	166-179
PSTTRXIT	8	CHAR	Post-translation exit routine	180-187
ACTIVE	1	CHAR	Active usage for this INTPID/FORMAT (Y/N)	188-188

Table 2-25 (Page 2 of 2). Trading Partner Send Usage

Tag	Length	Type	Description	Fixed Position
ACKREQ	1	CHAR	Acknowledgment expected flag (Y/N)	189-189
TESTIND	1	CHAR	Usage indicator (P/T/I)	190-190
LOGAPP	1	CHAR	Log application data (Y/N)	191-191
ENFORCEHIER	1	CHAR	Enforce structure hierarchy (Y/N)	192-192
STRUCTDATACHK	1	CHAR	Structure must produce data (Y/N)	193-193
TRVALVL	1	DEC	Validation level (0,1,2)	194-194
TRERLVL	1	DEC	Acceptable error level (0,1,2)	195-195
GRPSECPR	8	CHAR	Group Security profile member name	196-203
GRPENCKY	16	CHAR	Group Encryption key name	204-219
GRPAUTKY	16	CHAR	Group Authentication key name	220-235
TRNSECPR	8	CHAR	Trans Security profile member name	236-243
TRNENCKY	16	CHAR	Trans Encryption key name	244-259
TRNAUTKY	16	CHAR	Trans Authentication key name	260-275
DESCRIPT	30	CHAR	Member description	276-305
APPLTPID	16	CHAR	Application Trading Partner ID	306-321
ALPHANUM	8	CHAR	Name of override ALPHANUM table	322-329
CHARSET	8	CHAR	Name of override CHARSET table	330-337
CNTRX	1	CHAR	Control numbers utilization flag (Y/N)	338-338

Trading Partner Receive Usage (T6)

Table 2-26 (Page 1 of 2). Trading Partner Receive Usage

Tag	Length	Type	Description	Fixed Position
TPNICKN	16	CHAR	Trading partner nickname	004-019
STDTRID	8	CHAR	Standard transaction ID	020-027
TPTID	16	CHAR	Trading Partner Mapping Transaction ID	028-043
APSNDRCV	35	CHAR	Application sender/receiver ID	044-078
AGENCY	8	CHAR	Responsible agency code	079-086
VER	8	CHAR	Version	087-094
REL	8	CHAR	Release	095-102
SNDRCVFL	1	CHAR	Sender or receive ID flag (S/R)	103-103
INTPID	35	CHAR	Internal trading partner ID	104-138
STDPROF	8	CHAR	Standard Profile member name	139-146
APPASSWD	14	CHAR	Application password	147-160
PRETRXIT	8	CHAR	Pre-translation exit routine	161-168
FILEID	8	CHAR	Logical receiving data set name	169-176
FILETYPE	2	CHAR	Type of receive file (MQ, PG, TD, TM, TS, TX, VS)	177-178

Table 2-26 (Page 2 of 2). Trading Partner Receive Usage

Tag	Length	Type	Description	Fixed Position
ACTIVE	1	CHAR	Active usage for this Trading Partner TPNICKN/STDTRID (Y/N)	179-179
ACKTRN	8	CHAR	Acknowledgment transaction ID. The G appended at the end of the ACKTRN identifier specifies the group level acknowledgments.	180-187
TESTIND	1	CHAR	Usage indicator (P/T/I)	188-188
LOGAPP	1	CHAR	Log application data (Y/N)	189-189
OVERLAYCHK	1	CHAR	Consider data overlay an error (Y/N)	190-190
UNEXPFLDCHK	1	CHAR	Consider unexpected field an error (Y/N)	191-191
UNEXPSEGCHK	1	CHAR	Consider unexpected segment an error (Y/N)	192-192
SWITCHROUTING	1	CHAR	Switch appl routing IDs on Func Ack (Y/N)	193-193
TRVALVL	1	DEC	Validation level (0,1,2)	194-194
TRERLVL	1	DEC	Acceptable error level (0,1,2)	195-195
INTFA	1	CHAR	Inbound envelope used on FA	196-196
DESCRIPT	30	CHAR	Member description	197-226
APPLTPID	16	CHAR	Application Trading Partner ID	306-321
ALPHANUM	8	CHAR	Name of override ALPHANUM table	322-329
CHARSET	8	CHAR	Name of override CHARSET table	330-337
CNTRX	1	CHAR	Control numbers utilization flag (Y/N)	338-338

The following shows an example of an export/import file that contains tags for a trading partner transactions and an associated send usage.

```
0C1SMITH 940419161616ENU010103000000
3M1TPTID(HDWMRINV) STDID(X12V3R2) STDTRID(810) FORMAT(HDWMRINV) DIR(S)
3T2TPTID(HDWMRINV) CURSGSEQ(0) NXTLPOCC(0) NXTSGOCC(0) FLTSGSEQ(0)
3T2TPTID(HDWMRINV) CURSGSEQ(1) NXTLPOCC(0) NXTSGOCC(0) FLTSGSEQ(0)
3T2TPTID(HDWMRINV) CURSGSEQ(2) NXTLPOCC(0) NXTSGOCC(0) FLTSGSEQ(0)
3T3TPTID(HDWMRINV) CURSGSEQ(1) CURDESEQ(1) NXTDEOCC(0) STDID(X12V3R2)
3T4TPTID(HDWMRINV) CURSGSEQ(1) CURDESEQ(1) DATEDIT(3) SUBPOS(0)
3T5INTPID(CUSTN000) FORMAT(HDWMRINV) TPTID(HDWMRINV) TPNICKN(HDWTEST)
000
```

Export/Import of Table Definitions

Table Definitions can be exported/imported using tagged or fixed format files. The Table Definitions set includes:

B1 Table Definition

B2 Table Entry

Table Definition (B1)

Table 2-27. Table Definition

Tag	Length	Type	Description	Fixed Position
TBLID	8	CHAR	Table ID	004-011
DATECRT	6	CHAR	Date created	012-017
TBLDSC	35	CHAR	Table description	018-052
PUBAUTH	6	CHAR	Authority	053-058
OWNER	8	CHAR	Owner	059-066
FLAGS	2	CHAR	Flags (always "00")	067-068
KEYID	18	CHAR	Key field ID	069-086
KEYLN	6	DEC	Key field length	087-092
RECLN	6	DEC	Record length	093-098
NUMFLDS	6	DEC	Number of fields	099-104
TBLTYPE	1	CHAR	Table type	105-105
FLD1ID	18	CHAR	Field 1 ID	106-123
FLD1OFF	6	DEC	Field 1 offset	124-129
FLD1LEN	3	DEC	Field 1 length	130-132
FLD1TYPE	2	CHAR	Field 1 type	133-134
FLD1DSC	35	CHAR	Field 1 description	135-169
FLD2ID	18	CHAR	Field 2 ID	170-187
FLD2OFF	6	DEC	Field 2 offset	188-193
FLD2LEN	3	DEC	Field 2 length	194-196
FLD2TYPE	2	CHAR	Field 2 type	197-198
FLD2DSC	35	CHAR	Field 2 description	199-233
TAG	8	CHAR	Future index tag	234-241

Table Definition (B1) Field Descriptions

Table ID (TBLID)	A name for referring to the table.
Date Created (DATECRT)	Date this table was created.
Table Description (TBLDSC)	A phrase that describes this table.
Authority (PUBAUTH)	DataInterchange use only.
Owner	User ID of the owner of this table.
Flags	Always set to "00".
Key ID (KEYID)	ID of the field that is the key (the field that the lookup performs on). Validation and type T translation tables use the first (or only) field as the key field. Type R translation tables use the second field.
Key field length (KEYLN)	Length of the key field.
Record length (RECLN)	The length of a row in this table.

Number of fields (NUMFLDS)	The total number of fields in a row in this table. Validation tables have one field. Translation tables have two.								
Table Type (TBLTYPE)	<p>A code that indicates the type of table:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>T</td><td>Translation table. This table contains pairs of values: a value to be translated and the value to be substituted for it. It can be used, for example, to translate a purchaser's part number to a supplier's part number before sending a purchase order. It is organized with the application (local) value as a key value and the standard or trading partner value as data. Therefore, you can have multiple application values that translate to the same standard value, but each standard value will translate to one and only one application value. This table gives the best performance when translating from an application value to a standard value.</td></tr> <tr> <td>R</td><td>Translate table. Serves the same purpose as the T table but is organized with the standard value as the key and the application value as data. With this type of table, multiple standard values can translate to the same application value, but each application value will translate to one and only one standard value. This table gives the best performance when translating from a standard value to an application value.</td></tr> <tr> <td>V</td><td>Validation table. This table contains a list of acceptable values for a field. It can contain several values or only one value. It is used to test whether a field contains a correct value.</td></tr> </table>	Value	Description	T	Translation table. This table contains pairs of values: a value to be translated and the value to be substituted for it. It can be used, for example, to translate a purchaser's part number to a supplier's part number before sending a purchase order. It is organized with the application (local) value as a key value and the standard or trading partner value as data. Therefore, you can have multiple application values that translate to the same standard value, but each standard value will translate to one and only one application value. This table gives the best performance when translating from an application value to a standard value.	R	Translate table. Serves the same purpose as the T table but is organized with the standard value as the key and the application value as data. With this type of table, multiple standard values can translate to the same application value, but each application value will translate to one and only one standard value. This table gives the best performance when translating from a standard value to an application value.	V	Validation table. This table contains a list of acceptable values for a field. It can contain several values or only one value. It is used to test whether a field contains a correct value.
Value	Description								
T	Translation table. This table contains pairs of values: a value to be translated and the value to be substituted for it. It can be used, for example, to translate a purchaser's part number to a supplier's part number before sending a purchase order. It is organized with the application (local) value as a key value and the standard or trading partner value as data. Therefore, you can have multiple application values that translate to the same standard value, but each standard value will translate to one and only one application value. This table gives the best performance when translating from an application value to a standard value.								
R	Translate table. Serves the same purpose as the T table but is organized with the standard value as the key and the application value as data. With this type of table, multiple standard values can translate to the same application value, but each application value will translate to one and only one standard value. This table gives the best performance when translating from a standard value to an application value.								
V	Validation table. This table contains a list of acceptable values for a field. It can contain several values or only one value. It is used to test whether a field contains a correct value.								
Field 1 ID (FLD1ID)	ID of the first column in a row of this table.								
Field 1 offset (FLD1OFF)	Offset into a row of this table where field 1 begins.								
Field 1 length (FLD1LEN)	<p>The maximum number of characters one entry in this column will contain. The limit is 35. The length for both columns (if there are two columns) cannot exceed 68 characters. If 20 is used as the length of the local value, for example, the length of the translation cannot exceed 48.</p> <p>For data type R, count the decimal point and sign as part of the data length. For example, the value -12.34 requires a length of 6.</p>								
Field 1 data type (FLD1TYPE)	<p>A code that indicates the data type for entries in this column:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>CH</td><td>Character. Any combination of characters.</td></tr> <tr> <td>R</td><td>Real. A numeric field that requires a decimal point for fractional values. The decimal point is optional for integers. A sign (+ or -) is optional.</td></tr> </table>	Value	Description	CH	Character. Any combination of characters.	R	Real. A numeric field that requires a decimal point for fractional values. The decimal point is optional for integers. A sign (+ or -) is optional.		
Value	Description								
CH	Character. Any combination of characters.								
R	Real. A numeric field that requires a decimal point for fractional values. The decimal point is optional for integers. A sign (+ or -) is optional.								
Field 1 description (FLD1DSC)	A brief description of this field.								
Field 2 ID (FLD2ID)	ID of the second column in a row of this table (if there is one).								
Field 2 offset (FLD2OFF)	Offset into a row of this table where field 2 begins.								

Field 2 length (FLD2LEN) The maximum number of characters one entry in this column will contain. The limit is 63 (35 for R type translate table). The length for both columns cannot exceed 68 characters. If 20 is used as the length of the local value, for example, the length of the translation cannot exceed 48.

For data type R, count the decimal point and sign as part of the data length. For example, the value -12.34 requires a length of 6.

Field 2 data type (FLD2TYPE) A code that indicates the data type for entries in this column:

Value	Description
CH	Character. Any combination of characters.
R	Real. A numeric field that requires a decimal point for fractional values. The decimal point is optional for integers. A sign (+ or -) is optional.

Field 2 description (FLD2DSC)

A brief description of this field.

Tag (TAG)

A future index tag.

Table Entry (B2)

Table 2-28. Table Entry

Tag	Length	Type	Description	Fixed Position
TBLID	8	CHAR	Table ID	004-011
VAR1	variable	CHAR	Table member record - depends on field descriptions	012-variable
VAR2	variable	CHAR	Table member record - depends on field descriptions	variable

Table Entry (B2) Field Descriptions

Table ID (TBLID) A name for referring to the table.

VAR1 For validation tables, this is a valid value to be entered into the table. For translation type tables, this is the application value (user defined) to be associated with a value from the standards.

VAR2 Only used for translation type tables. This is the standard value that is to be associated with an application value (user defined).

The following shows an example of an export/import file which contains tags for table definitions.

```
0C1SMITH 940419161616ENU010103000000
8B1TBLID(AAAAA) DATECRT(940325) TBLDSC(dd) PUBAUTH(ALL) FLAGS(00)
8B2TBLID(AAAAA) VAR1(A)
000
```

Additional Profile Layouts

Additional profile layouts are contained within the following tables.

Requestor Profile Member (REQPROF-P2)

Table 2-29. Requestor Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile indicator = REQPROF	004-011
REQID	16	CHAR	Requestor ID	012-027
NETID	8	CHAR	Network identification	028-035
ACCTID	32	CHAR	Network account number	036-067
USERID	32	CHAR	Network user ID	068-099
NETPASS	16	CHAR	Network password (current/new)	100-115
MSGCLS	8	CHAR	Network message user class	116-123
FILEID	8	CHAR	ddname for transaction receive	124-131
NETCLS	1	CHAR	Network classification	132-132
NETCHG	1	CHAR	Network charges code	133-133
NETACK	1	CHAR	Network acknowledgment code	134-134
NETVCHK	1	CHAR	Validate destination	135-135
NETRETN	3	CHAR	Message retention	136-138
NETEDIO	1	CHAR	EDI processing option	139-139
NETEDIP	1	CHAR	EDI processing override	140-140
STGFMTOV	1	CHAR	Storage format override	141-141
STGFMT	1	CHAR	Storage format	142-142
NETCMDS	8	CHAR	Member name of network commands file	143-150
TIMEOUT	4	DEC	Command line timeout value	151-154
NETACKPG	8	CHAR	Program to handle NA from remote net	155-162
ALTNETPH	32	CHAR	Alternate dial connection phone num	163-194
COMPRESS	1	CHAR	Compression flag (T,Y,N)	195-195
PRIORITY	1	CHAR	Delivery priority (blank or P)	196-196
DESCRIPT	30	CHAR	Member description	197-226

Security Profile Member (SECUPROF-P2)

Table 2-30 (Page 1 of 2). Security Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = SECUPROF	004-011
SECID	8	CHAR	Security ID	012-019
SECORIGN	16	CHAR	Security originator	020-035
SECRECIP	16	CHAR	Security recipient	036-051
AUTTYPE	1	CHAR	Authorization type	052-052
AUTCODE	1	CHAR	Authorization code	053-053
ENCTYPE	1	CHAR	Encryption code	054-054
FILTYPE	1	CHAR	Filtering type	055-055

Table 2-30 (Page 2 of 2). Security Profile

Tag	Length	Type	Description	Fixed Position
ENCPROG	8	CHAR	Encryption program	056-063
AUTPROG	8	CHAR	Authentication program	064-071
COMPROG	8	CHAR	Compression program	072-079
FILPROG	8	CHAR	Filtering program	080-087
BUFSIZE	5	DEC	Buffer size	088-092
DESCRIPT	30	CHAR	Member description	093-122

Network Profile Member (NETPROF-P2)

Table 2-31. Network Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = NETPROF	004-011
NETID	8	CHAR	Network identification	012-019
NETNAME	30	CHAR	Network name	020-049
COMMRTN	8	CHAR	Communication routine name	050-057
NETPROG	8	CHAR	Network send/receive program name	058-065
NETPARM	57	CHAR	Network program parameters	066-122
CMDINDD	8	CHAR	Network prog command input file name	123-130
CMDRECLN	4	DEC	Network command record length	131-134
QFILEDD	8	CHAR	Transaction data queue file name	135-142
QRECLN	4	DEC	Transaction data record length	143-146
TIMEZONE	5	CHAR	Time zone	147-151
SYSTYPE	8	CHAR	System type	152-159
SYSLVL	4	CHAR	System level	160-163
MSGTXTH	1	CHAR	Message text header character	164-164
CMDOUTDD	8	CHAR	Network prog command output file name	165-172
MSGHNDLR	8	CHAR	Program to process messages	173-180
NETSEQ	5	CHAR	Sequence number for network	181-185
NETAKFILE	8	CHAR	File for network acknowledgements	186-193
NETPHONE	32	CHAR	Dial connection phone number	194-225
SCRIPT	8	CHAR	Communication script name	226-233
DESCRIPT	30	CHAR	Member description	234-263

Network Operations Profile Member (NETOP-P2)

Table 2-32 (Page 1 of 2). Network Operations Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = NETOP	004-011
NETID	8	CHAR	Network ID	012-019

Table 2-32 (Page 2 of 2). Network Operations Profile

Tag	Length	Type	Description	Fixed Position
NETOP	8	CHAR	Network Operation	020-027
NETFS	8	CHAR	Operation sequence number	028-035
BLKNM	8	CHAR	Block name	036-043
BLKPOS	4	DEC	Block position	044-047
CMDSEQ	4	DEC	Command line sequence number	048-051
CMDPOS	4	DEC	Command field position	052-055
CMDLEN	4	DEC	Command field length	056-059
CMDVAL	58	CHAR	Command field value	060-117
DESCRIPT	30	CHAR	Member description	118-147

Activity Log Profile Member (ACTLOGS-P2)

Table 2-33. Activity Log Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = ACTLOGS	004-011
APPLID	8	CHAR	Application ID	012-019
LOGFLNM	8	CHAR	Logical name of file	020-027
PRGNAME	8	CHAR	File handler pgm name (EDIELAD)	028-035
LANG	1	CHAR	Language type of log handler (always C)	036-036
LOGFLAG	2	CHAR	Log profile activity flag	037-038
DESCRIPT	30	CHAR	Member description	039-068

Application Definition Profile Member (APPDEFS-P2)

Table 2-34 (Page 1 of 2). Application Definition Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = APPDEFS	004-011
APPLICID	8	CHAR	Application ID	012-019
MEMBER	8	CHAR	ACTLOGS member name	020-027
MRFLAG	1	CHAR	Management reporting active (Y/N)	028-028
TSFLAG	1	CHAR	Transaction Store active (Y/N/S/E)	029-029
TIFLAG	1	CHAR	Transaction image wanted (Y/N/S/E)	030-030
MONITOR	8	CHAR	CICS performance monitor exit name	031-038
USFLAG	1	CHAR	Test trans use prod usage (P/T/I)	039-039
LOGENV	1	CHAR	Log standard data (Y/N)	040-040
FAIFLAG	1	CHAR	FA image wanted (Y/N/S/E)	041-041
ELFLAG	1	CHAR	Envelope log active flag (Y/N)	042-042
DESCRIPT	30	CHAR	Member description	043-072
ALPHANUM	8	CHAR	Name of override ALPHANUM table	073-080

Table 2-34 (Page 2 of 2). Application Definition Profile

Tag	Length	Type	Description	Fixed Position
CHARSET	8	CHAR	Name of override CHARSET table	081-088
CTRLYY	2	CHAR	Century control year	089-090

User Program Information Profile Member (ADAMCTL-P2)

Table 2-35. User Program Information Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = ADAMCTL	004-011
LOGPGNM	8	CHAR	Logical name of program	012-019
LOADMOD	8	CHAR	Load module name	020-027
LANG	1	CHAR	Language used in user program	028-028
ADUEFLD	1	CHAR	Field exit type	029-029
ADUEPST	1	CHAR	Post-translate type	030-030
ADUEPRE	1	CHAR	Pre-translate type	031-031
ADUEENC	1	CHAR	Encryption type	032-032
ADUEAUT	1	CHAR	Authentication type	033-033
ADUECMP	1	CHAR	Compression type	034-034
ADUEFLT	1	CHAR	Filtering type	035-035
ADUEMNT	1	CHAR	Monitor type	036-036
ADUECOM	1	CHAR	Communication type	037-037
ADUEMSG	1	CHAR	Message process type	038-038
ADUEPTP	1	CHAR	Point-to-point type	039-039
ADUEENV	1	CHAR	Envelope type	040-040
DESCRIPT	30	CHAR	Member description	041-070

Language Profile Member (LANGPROF-P2)

Table 2-36. Language Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = LANGPROF	004-011
LANGID	6	CHAR	Language Profile ID	012-017
CPAGE	5	CHAR	Code page ID	018-022
DMASK	10	CHAR	Date edit/display mask	023-032
TMASK	8	CHAR	Time edit/display mask	033-040
DECNOT	1	CHAR	Edit/display decimal notation	041-041
SIGN	2	CHAR	Preferred negative sign (disp)	042-043
FOLD	1	CHAR	Substitute for non-display character	044-044
DESCRIPT	30	CHAR	Member description	045-074

EDIFACT Profile Member (E-P2)

Table 2-37 (Page 1 of 2). EDIFACT Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = E	004-011
EDIFKEY	8	CHAR	Profile key	012-019
UNB01	4	CHAR	Syntax identifier	020-023
UNB02	1	CHAR	Syntax version number	024-024
UNB03	35	CHAR	Sender ID	025-059
UNB04	4	CHAR	Sender ID qualifier	060-063
UNB05	14	CHAR	Reverse routing address	064-077
UNB06	35	CHAR	Recipient ID	078-112
UNB07	4	CHAR	Recipient ID qualifier	113-116
UNB08	14	CHAR	Routing address	117-130
UNB09	6	CHAR	Date	131-136
UNB10	4	CHAR	Time	137-140
UNB11	14	CHAR	Interchange control reference	141-154
UNB12	14	CHAR	Password	155-168
UNB13	2	CHAR	Password qualifier	169-170
UNB14	14	CHAR	Application reference	171-184
UNB15	1	CHAR	Processing priority	185-185
UNB16	1	CHAR	Acknowledgment request	186-186
UNB17	35	CHAR	Communications agreement ID	187-221
UNB18	1	CHAR	Test indicator	222-222
UNZ01	6	CHAR	Interchange control count	223-228
UNZ02	14	CHAR	Interchange control reference	229-242
UNG01	6	CHAR	Functional group ID	243-248
UNG02	35	CHAR	Sender ID	249-283
UNG03	4	CHAR	Sender ID qualifier	284-287
UNG04	35	CHAR	Recipient ID	288-322
UNG05	4	CHAR	Recipient ID qualifier	323-326
UNG06	6	CHAR	Date	327-332
UNG07	4	CHAR	Time	333-336
UNG08	14	CHAR	Functional group reference number	337-350
UNG09	2	CHAR	Controlling agency	351-352
UNG10	3	CHAR	Message version	353-355
UNG11	3	CHAR	Message release	356-358
UNG12	6	CHAR	Association assigned	359-364
UNG13	14	CHAR	Application password	365-378
UNE01	6	CHAR	Number of messages	379-384

Table 2-37 (Page 2 of 2). EDIFACT Profile

Tag	Length	Type	Description	Fixed Position
UNE02	14	CHAR	Functional group reference number	385-398
UNH01	14	CHAR	Message reference number	399-412
UNH02	6	CHAR	Message type	413-418
UNH03	3	CHAR	Message version number	419-421
UNH04	3	CHAR	Message release number	422-424
UNH05	2	CHAR	Controlling agency	425-426
UNH06	6	CHAR	Association assigned	427-432
UNH07	35	CHAR	Common access reference	433-467
UNH08	2	CHAR	Sequence of transfer	468-469
UNH09	1	CHAR	First and last	470-470
UNT01	6	CHAR	Number of segments in message	471-476
UNT02	14	CHAR	Message reference number	477-490
DESCRIPT	30	CHAR	Member description	491-520

ICS Profile Member (I-P2)

Table 2-38 (Page 1 of 2). ICS Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = I	004-011
ICSKEY	8	CHAR	Profile key	012-019
ICS01	2	CHAR	Subelement separator	020-021
ICS02	4	CHAR	Control standards ID	022-025
ICS03	5	CHAR	Control version number	026-030
ICS04	2	CHAR	Sender ID qualifier	031-032
ICS05	15	CHAR	Information sender ID	033-047
ICS06	2	CHAR	Receiver ID qualifier	048-049
ICS07	15	CHAR	Information receiver ID	050-064
ICS08	6	CHAR	Date	065-070
ICS09	4	CHAR	Time	071-074
ICS10	9	CHAR	Interchange control number	075-083
ICE01	6	CHAR	Number of groups	084-089
ICE02	9	CHAR	Interchange control number	090-098
GS01	2	CHAR	Functional group ID	099-100
GS02	15	CHAR	Application sender's code	101-115
GS03	15	CHAR	Application receiver's code	116-130
GS04	6	CHAR	Date	131-136
GS05	8	CHAR	Time	137-144
GS06	9	CHAR	Functional group control number	145-153

Table 2-38 (Page 2 of 2). ICS Profile

Tag	Length	Type	Description	Fixed Position
GS07	2	CHAR	Responsible agency code	154-155
GS08	12	CHAR	Version/release/industry ID	156-167
GE01	6	CHAR	Number of included sets	168-173
GE02	9	CHAR	Functional group control number	174-182
ST01	3	CHAR	Transaction set ID	183-185
ST02	9	CHAR	Transaction set control number	186-194
SE01	10	CHAR	Number of included segments	195-204
SE02	9	CHAR	Transaction set control number	205-213
DESCRIPT	30	CHAR	Member description	214-243

UNTDI Profile Member (T-P2)

Table 2-39. UNTDI Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = T	004-011
TDIKEY	8	CHAR	Profile key	012-019
STX01	4	CHAR	Syntax identifier	020-023
STX02	1	CHAR	Syntax version number	024-024
STX03	14	CHAR	Sender code	025-038
STX04	35	CHAR	Sender name	039-073
STX05	14	CHAR	Recipient code	074-087
STX06	35	CHAR	Recipient name	088-122
STX07	6	CHAR	Date	123-128
STX08	6	CHAR	Time	129-134
STX09	14	CHAR	Interchange control reference	135-148
STX10	14	CHAR	Recipient's reference/password	149-162
STX11	14	CHAR	Application reference	163-176
STX12	1	CHAR	Transmission priority code	177-177
END01	5	CHAR	Total number of messages	178-182
MHD01	12	CHAR	Message reference	183-194
MHD02	6	CHAR	Message type	195-200
MHD03	1	CHAR	Message version	201-201
MHD04	35	CHAR	Common access reference	202-236
MHD05	2	CHAR	Sequence of transfers	237-238
MHD06	1	CHAR	First and last transfers	239-239
MTR01	4	CHAR	Number of segments in message	240-243
DESCRIPT	30	CHAR	Member description	244-273

UCS Profile Member (U-P2)

Table 2-40. UCS Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = U	004-011
UCSKEY	8	CHAR	Profile key	012-019
BG01	10	CHAR	Comm ID	020-029
BG02	10	CHAR	Comm password	030-039
BG03	15	CHAR	Application sender ID	040-054
BG04	15	CHAR	Application receiver ID	055-069
BG05	6	CHAR	Date	070-075
BG06	6	CHAR	Time	076-081
BG07	5	CHAR	Transmission control number	082-086
EG01	5	CHAR	Transmission control number	087-091
EG02	5	CHAR	Number of included groups	092-096
EG03	6	CHAR	Number of included transaction sets	097-102
EG04	10	CHAR	Number of included segments	103-112
GS01	2	CHAR	Functional group ID	113-114
GS02	15	CHAR	Application sender's code	115-129
GS03	15	CHAR	Application receiver's code	130-144
GS04	6	CHAR	Date	145-150
GS05	8	CHAR	Time	151-158
GS06	9	CHAR	Functional group control number	159-167
GS07	2	CHAR	Responsible agency code	168-169
GS08	12	CHAR	Version/release/industry ID	170-181
GE01	6	CHAR	Number of included sets	182-187
GE02	9	CHAR	Functional group control number	188-196
ST01	3	CHAR	Transaction set ID	197-199
ST02	9	CHAR	Transaction set control number	200-208
SE01	10	CHAR	Number of included segments	209-218
SE02	9	CHAR	Transaction set control number	219-227
DESCRIPT	30	CHAR	Member description	228-257

X12 Profile Member (X-P2)

Table 2-41 (Page 1 of 2). X12 Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = X	004-011
X12KEY	8	CHAR	Profile key	012-019
ISA01	2	CHAR	Auth info qualifier	020-021
ISA02	10	CHAR	Auth info	022-031

Table 2-41 (Page 2 of 2). X12 Profile

Tag	Length	Type	Description	Fixed Position
ISA03	2	CHAR	Security info qualifier	032-033
ISA04	10	CHAR	Security info	034-043
ISA05	2	CHAR	Interchange sender ID qualifier	044-045
ISA06	15	CHAR	Interchange sender ID	046-060
ISA07	2	CHAR	Interchange receiver ID qualifier	061-062
ISA08	15	CHAR	Interchange receiver ID	063-077
ISA09	6	CHAR	Date	078-083
ISA10	4	CHAR	Time	084-087
ISA11	1	CHAR	Interchange standards ID	088-088
ISA12	5	CHAR	Interchange version ID	089-093
ISA13	9	CHAR	Interchange control number	094-102
ISA14	1	CHAR	Acknowledgment requested	103-103
ISA15	1	CHAR	Test indicator	104-104
ISA16	2	CHAR	Subelement separator	105-106
IEA01	5	CHAR	Number of included groups	107-111
IEA02	9	CHAR	Interchange control number	112-120
GS01	2	CHAR	Functional group ID	121-122
GS02	15	CHAR	Application sender's code	123-137
GS03	15	CHAR	Application receiver's code	138-152
GS04	6	CHAR	Date	153-158
GS05	8	CHAR	Time	159-166
GS06	9	CHAR	Functional group control number	167-175
GS07	2	CHAR	Responsible agency code	176-177
GS08	12	CHAR	Version/release/industry ID	178-189
GE01	6	CHAR	Number of included sets	190-195
GE02	9	CHAR	Functional group control number	196-204
ST01	3	CHAR	Transaction set ID	205-207
ST02	9	CHAR	Transaction set control number	208-216
SE01	10	CHAR	Number of included segments	217-226
SE02	9	CHAR	Transaction set control number	227-235
DESCRIPT	30	CHAR	Member description	236-265

Note: The following two profiles are for CICS only.

CONTRECV Member (CONTRECV-P2)

Table 2-42 (Page 1 of 2). CONTRECV Member

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = CONTRECV	004-011

Table 2-42 (Page 2 of 2). CONTRECV Member

Tag	Length	Type	Description	Fixed Position
CONTRCV	16	CHAR	Profile member identifier	012-027
ACTIVE	1	CHAR	Profile member active (Y,N)	028-028
REQID	16	CHAR	Requestor ID	029-044
TPNICKN	16	CHAR	Trading Partner nickname	045-060
MSGCLS	8	CHAR	Network message user class	061-068
TRANSLATE	1	CHAR	Deenvelope and translate (Y,N)	069-069
RAWDATA	1	CHAR	Generate raw data (Y,N)	070-070
PRINTNM	8	CHAR	Print file name	071-078
PRINTTYP	2	CHAR	Print file type	079-080
EXCPNM	8	CHAR	Exception file name	081-088
EXCTYP	2	CHAR	Exception file type	089-090
ADDLRECS	5	CHAR	Additional Records (IEGTQ)	091-095
DENVONLY	1	CHAR	Deenvelope only (Y,N)	096-096
DELAYFA	1	CHAR	Delay FA enveloping (Y,N)	097-097
FATSQ	8	CHAR	Enveloped FA TS queue name	098-105
RESPNM	8	CHAR	Name of response App/TD queue	106-113
RESPTP	2	CHAR	Type of response (PG,TX)	114-115
USERFLD	16	CHAR	User field passed to response	116-131
APPLID	8	CHAR	Application ID	132-139
NLSID	6	CHAR	National Language ID	140-145
SYNCPTS	1	CHAR	Allow syncpoints (Y,N)	146-146
DUPENVL	1	CHAR	Allow dup envelopes (Y,N)	147-147
NETAKONLY	1	CHAR	Process network acks (Y,N)	148-148
PURGEINT	4	DEC	Transaction purge interval	149-152
DESCRIPT	30	CHAR	Member description	153-182
SAPUPDT	1	CHAR	SAP update flag	183-183
PAGE	1	CHAR	Pageable translation active flag (Y/N)	184-184

System Profile Member (SYSPROF-P2)

Table 2-43. System Profile

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile Indicator = SYSPROF	004-011
SYSID	8	CHAR	System ID	012-019
PERSACTV	1	CHAR	Persistent Environ active (Y/N)	020-020
PERSIZE	4	DEC	Persistent Environ size	021-024
PERSTHDS	2	DEC	Number Persistent Environ threads	025-026
DESCRIPT	30	CHAR	Member description	027-056

MQSeries Queue Profile Member (MQSERIES-P2)

Table 2-44. MQSeries Queue

Tag	Length	Type	Description	Fixed Position
PROFID	8	CHAR	Profile indication = MQSERIES	004-011
QUEUEID	8	CHAR	The logical name associated with the Queue Name	012-019
REALNAME	48	CHAR	The real MQSeries queue name	020-067
MGRNAME	48	CHAR	Override queue manager name	068-115
READFLAG	1	CHAR	Destructive read flag indicator	116-116
MQSYNCFLAG	1	CHAR	Syncpoint control flag	117-117
MAXMSGLN	8	CHAR	Maximum message length	118-125
DESCRIPT	30	CHAR	Member description	126-155

Importing the New Definitions to DataInterchange

After the E/I file data set has been updated, you can either submit the updates using the batch utility or by selecting the DataInterchange Import function.

Mapping Migration Input Control File (INCNTL)

The mapping migration input control file contains records to assist the service in matching old segments and new segments. Because the command language syntax contained within this file is not free format, you should run the migration once with a mapping migration output control file (OUTCNTL), and no mapping migration input control file (INCNTL). The mapping migration output control file is described in "Mapping Migration Output Control File (OUTCNTL)" on page 2-60. If changes to the mapping migration are needed, edit the mapping migration output control file and submit only the changed records as the mapping migration input control file for the next run of mapping migration. You can use the same file and data definition name (ddname) for both the mapping migration input control file and the mapping migration output control file.

Notes:

1. By placing an asterisk (*) in column one of a record, you are indicating the record is a comment. When you use the mapping migration output control file as your input file, only the from and to segment sequence numbers are used by the mapping migration command. The from and the to segment IDs are for your use only and are ignored by the mapping migration command.
2. The segment IDs for the FROMSEG and the TOSEG can be different.
3. It is useful to use an INCNTL record when migrating different transactions or messages in which many segments inside a loop are identical, but have a different loop or segment group starting segment. The elements of the starting segment may not be migrated, but at least the segments and elements inside the loop that match up will be migrated.
4. The following are the mapping migration criteria:
 - The level of looping for the FROMSEG and the TOSEG must be the same.
 - If the FROMSEG segment starts a loop, the TOSEG must start a loop.
 - If the FROMSEG segment does not start a loop, the TOSEG must not start a loop.

5. If the previous criteria are not met, the control record will define the starting location for the normal search to begin.

The mapping migration input control file specifications are:

Action	Description
Use	Input file containing mapping migration control records.
Specified	INCNTL keyword.
ddname	User defined.
Suggested format	Record format: Fixed or variable. Record length: 80 bytes or greater.
Remarks	The processing of this file adjusts to the file attributes. Therefore, you can define this file as it best fits with your conventions.

File Format

The following is the format of an entry in the mapping migration input control file:

FROMSEG(fsn) TOSEG(tsn)

Where:

- fsn Is the sequence number of the segment in the standard in the from map. See "Migrating Trading Partner Transactions" on page 1-61 for more information.
- tsn Is the sequence number of the segment in the standard in the to map. See "Migrating Trading Partner Transactions" on page 1-61 for more information. The number 999 can be specified as a "TO SEGment sequence number" to indicate the mapping for a specified segment should not be migrated.

The following record tells the service to migrate the 40th segment in the transaction in the from mapping to the 39th segment in the transaction in the new or updated standard.

FROMSEG(40) TOSEG(39)

See "Migrating Trading Partner Transactions" on page 1-61 for more information.

Mapping Migration Output Control File (OUTCNTL)

- | The mapping migration control records help you understand how the migration was performed. In
| addition, you can edit the records and any changed records can be resubmitted as input control records
| during a mapping migration run to correct any problems.

The mapping migration output control file specifications are:

Action	Description
Use	Output file to which mapping migration control records are written.
Specified	OUTCNTL keyword.
ddname	User defined.
Suggested format	Record format: Fixed or variable. Record length: 80 bytes or greater.

Remarks The processing of this file adjusts to the file attributes. Therefore, you can define this file as it best fits with your conventions.

File Format

The following is the format of an entry in the mapping migration output control file:

```
FROMSEG(fsn) TOSEG(tsn) fsg            T0 tsg
```

Where:

fsn Is the sequence number of the segment in the standard in the from map. See “Migrating Trading Partner Transactions” on page 1-61 for more information.

tsn Is the sequence number of the segment in the standard in the to map. See “Migrating Trading Partner Transactions” on page 1-61 for more information. 999 means the mapping for a from specified segment could not be migrated.

fsg Is the standard segment ID in the from map.

tsn Is the standard segment ID in the to map (or NONE if the segment could not be migrated).

The following record tells you that the PO1 segment, which is the 40th segment in the transaction in the from mapping, migrates to the POC segment. The POC segment is the 39th segment in the transaction in the new or updated standard.

```
FROMSEG(40) TOSEG(39) P01            T0 P0C
```

See “Migrating Trading Partner Transactions” on page 1-61 for more information.

DataInterchange Utility Records Format

This section describes the format of the DataInterchange Utility records, which are:

- Control (C)
- Data (D)
- End transaction (Z)
- Raw data
- Optional

Control Record (C)

Table 2-45 describes the C record format. This format is used on both Send and Receive transactions. Use location 67 and greater to provide the override values you want the translator or enveloper to use. The expanded fields, 67 and greater, are optional. All fields are left-justified and case-sensitive.

Table 2-45 (Page 1 of 3). C Record for Translating to Standard Format

Label	Location	Type	Length	Description
RECID	1	Character	1	Record identifier
INTPID	2-36	Character	35	Internal trading partner ID
FORMATID	37-52	Character	16	Data format ID
TRANRC	53-56	Integer	4	Translator return code
TRANXRC	57-60	Integer	4	Translator extended return code
UTILRC	61-64	Integer	4	DataInterchange Utility return code

Table 2-45 (Page 2 of 3). C Record for Translating to Standard Format

Label	Location	Type	Length	Description
TESTIND	65	Character	1	Usage indicator (P/T/I/U)
MUWIND	66	Character	1	Multiple D records indicator
XPANDED	67	Character	1	Expanded control block indicator (Send translation only)
ITYPE	68	Character	1	Return information record indicator (Send translation only)
ETYPE	69	Character	1	Return envelope header record indicator (Send translation only)
GTYPE	70	Character	1	Return group header record indicator (Send translation only)
TTYPE	71	Character	1	Return transaction set header record indicator (Send translation only)
QTYPE	72	Character	1	Return queuing totals record indicator (Send translation only)
ISID	73-107	Character	35	Interchange sender ID
IRID	108-142	Character	35	Interchange receiver ID
IVERREL	143-147	Character	5	Interchange version and release number
ISPW	148-161	Character	14	Interchange password
IAPREF	162-175	Character	14	Interchange application reference
GSID	176-210	Character	35	Group application sender ID
GRID	211-245	Character	35	Group application receiver ID
GVER	246-257	Character	12	Group version
GREL	258-269	Character	12	Group release number
GAPW	270-283	Character	14	Group password
TVER	284-289	Character	6	Transaction version
TREL	290-295	Character	6	Transaction release number
HOLDFLAG	296	Character	1	Held status indicator
BNDLFLAG	297	Character	1	Bundle indicator
ROUTCODE	298-300	Character	3	Generic routing code
ISYNTAXID	301-304	Character	4	Interchange syntax ID (E and T envelopes)
ISYNTAXVER	305	Character	1	Interchange syntax VER (E and T envelopes)
ISIDQUAL	306-309	Character	4	Interchange sender ID qualifier
ISENDNAME	310-323	Character	14	Interchange sender name (T envelope) Application sender code (U envelope)
IREVROUT	324-337	Character	14	Interchange reverse routing (E envelope)
IRIDQUAL	338-341	Character	4	Interchange receiver ID qualifier
IrecvNAME	342-355	Character	14	Interchange receiver name (T envelope) Application receiver code (U envelope)
IROUTEADDR	356-369	Character	14	Interchange routing address (E envelope)

Table 2-45 (Page 3 of 3). C Record for Translating to Standard Format

Label	Location	Type	Length	Description
ISTDID	370-373	Character	4	Interchange standard IE (X and I envelopes)
IPRIOR	374	Character	1	Interchange priority (E and T envelopes)
ICOMMAGREE	375-409	Character	35	Interchange communication agreement (E envelope)
GSIDQ	410-413	Character	4	Group application sender ID qualifier
GRIDQ	414-417	Character	4	Group application receiver ID qualifier
GRESAG	418-419	Character	2	Group responsible agency code
DUPTRANS (Receive transaction only)	420	Character	1	Duplicate transaction indicator
FORCEC (Send translation only)	421	Character	1	Force C record on Send translation
APPLTPID	422-437	Character	16	Application trading partner
EDITPID	438-453	Character	16	EDI trading partner
RSRVD1	454-512	Character	59	Reserved
CUSERDATA	513-768	Character	256	User data area
RSRVD2	769-1024	Character	256	Reserved

Control Record Label Descriptions: The following are descriptions of the labels for the control record.

Label	Description
RECID	Specifies this record as the transaction set control record with the value C.
INTPID	Specifies the internal trading partner ID used to define the trading partner transaction.
FORMATID	Specifies the data format ID used to describe the application data.
TRANRC	Indicates the translation status of the transaction. The code is in binary format.
TRANXRC	Defines the translation status of the transaction. The code is in binary format.
UTILRC	For Send translation, specifies if a transaction is written to the exception file. The DataInterchange Utility returns the value 8 in UTILRC if it writes the transaction to the exception file. For Receive translation, UTILRC is always 0. The code is in binary format.
TESTIND	Specifies the usage indicator of the transaction. Valid values are:
	Value Description
I	Specifies that this is an information transaction and that an information usage should be used. If an information usage does not exist, a production usage will be used instead. Even when a production usage is used, the transaction is flagged as an information transaction.
P	Specifies that this is a production transaction and that only a production usage should be used. This is the default.
T	Specifies that this is a test transaction and that a test usage should be used, if one exists. If a test usage does not exist, a production usage will be used instead.

	Even when a production usage is used, the transaction is flagged as a test transaction.
U	Specifies that the translator should determine if the transaction is test, information, or production based on the usage found. If a test usage exists, the transaction is a test transaction, and value of T is returned. If an information usage exists, the transaction is an information transaction, and a value of I is returned. If only a production usage exists, the transaction is a production transaction, and a value of P is returned.
MUWIND	Specifies whether a single D record or multiple D records are involved in the transaction. Valid values are:
	Value Description
Y	Specifies multiple D records are involved in the transaction.
N	Specifies a single D record is involved in the translation.
XPANDED	Specifies the version of the C record being used. Valid values :
	Value Description
Y	Specifies the C record contains fields located through column 297.
N	Specifies the C record contains fields located through column 67.
1	Specifies the C record contains fields located through column 432.
2	Specifies the C record contains fields located through column 1024.
	This field is used on Send translation only. Receiver translation will always write all fields through column 1024.
ITYPE	Specifies whether you want an information record written to the exception/tracking file. Generally used for Send translation. Valid values are:
	Value Description
Y	Specifies you want the DataInterchange Utility to write an information record to the exception/tracking file for each translated transaction. For more information, see "Information (I) Record" on page 2-74.
N	No record written.
	On receive translation, the field will contain the same values as shown in the discussion of the DUPTRANS field on page 2-65.
ETYPE	For Send translation only. Specifies whether you want an interchange header image written to the exception/tracking file. Valid values are:
	Value Description
Y	Specifies you want the DataInterchange Utility to write an image of the interchange header segment to the exception/tracking file for each interchange processed. For more information, see "Interchange Header (E) Record" on page 2-75.
N	No record written.

GTYPE	For Send translation only. Specifies whether you want a group header image written to the exception/tracking file. Valid values are:												
	<table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Specifies that you want the DataInterchange Utility to write an image of the group header segment to the exception/tracking file for each group processed. For more information, see "Group Header (G) Record" on page 2-75.</td></tr> <tr> <td>N</td><td>No record written.</td></tr> </table>	Value	Description	Y	Specifies that you want the DataInterchange Utility to write an image of the group header segment to the exception/tracking file for each group processed. For more information, see "Group Header (G) Record" on page 2-75.	N	No record written.						
Value	Description												
Y	Specifies that you want the DataInterchange Utility to write an image of the group header segment to the exception/tracking file for each group processed. For more information, see "Group Header (G) Record" on page 2-75.												
N	No record written.												
TTYPE	For Send translation only. Specifies whether you want a transaction header image written to the exception/tracking file. Valid values are:												
	<table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Specifies that you want the DataInterchange Utility to write an image of the transaction header segment to the exception/tracking file for each transaction processed. For more information, see "Transaction Set Header (T) Record" on page 2-76.</td></tr> <tr> <td>N</td><td>No record written.</td></tr> </table>	Value	Description	Y	Specifies that you want the DataInterchange Utility to write an image of the transaction header segment to the exception/tracking file for each transaction processed. For more information, see "Transaction Set Header (T) Record" on page 2-76.	N	No record written.						
Value	Description												
Y	Specifies that you want the DataInterchange Utility to write an image of the transaction header segment to the exception/tracking file for each transaction processed. For more information, see "Transaction Set Header (T) Record" on page 2-76.												
N	No record written.												
QTYPE	For Send translation only. Specifies whether you want a record containing totals relative to an interchange written to the exception/tracking file. Valid values are:												
	<table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Specifies you want the DataInterchange Utility to write information relative to totals in an interchange to the exception/tracking file for each interchange processed. For more information see, "Queuing Totals (Q) Record" on page 2-76.</td></tr> <tr> <td>N</td><td>No record written.</td></tr> </table>	Value	Description	Y	Specifies you want the DataInterchange Utility to write information relative to totals in an interchange to the exception/tracking file for each interchange processed. For more information see, "Queuing Totals (Q) Record" on page 2-76.	N	No record written.						
Value	Description												
Y	Specifies you want the DataInterchange Utility to write information relative to totals in an interchange to the exception/tracking file for each interchange processed. For more information see, "Queuing Totals (Q) Record" on page 2-76.												
N	No record written.												
ISID	Overrides the ID provided in the envelope profile member or the trading partner profile member. It maps to a type IS standard data element. The envelope profile field that is overridden is:												
	<table> <tr> <th>This entry:</th><th>Overrides this envelope:</th></tr> <tr> <td>UNB03</td><td>E</td></tr> <tr> <td>ICS05</td><td>I</td></tr> <tr> <td>STX03 or STX04</td><td>T</td></tr> <tr> <td>BG03</td><td>U</td></tr> <tr> <td>ISA06</td><td>X</td></tr> </table> <p>DataInterchange will provide the equivalent data on Receive translation.</p>	This entry:	Overrides this envelope:	UNB03	E	ICS05	I	STX03 or STX04	T	BG03	U	ISA06	X
This entry:	Overrides this envelope:												
UNB03	E												
ICS05	I												
STX03 or STX04	T												
BG03	U												
ISA06	X												
IRID	Overrides the ID provided in the trading partner profile or envelope profile member. It maps to a type IR standard data element. The envelope profile field that is overridden is:												
	<table> <tr> <th>This entry:</th><th>Overrides this envelope:</th></tr> <tr> <td>UNB06</td><td>E</td></tr> <tr> <td>ICS07</td><td>I</td></tr> <tr> <td>STX05 or STX06</td><td>T</td></tr> <tr> <td>BG04</td><td>U</td></tr> <tr> <td>ISA08</td><td>X</td></tr> </table> <p>DataInterchange will provide the equivalent data on Receive translation.</p>	This entry:	Overrides this envelope:	UNB06	E	ICS07	I	STX05 or STX06	T	BG04	U	ISA08	X
This entry:	Overrides this envelope:												
UNB06	E												
ICS07	I												
STX05 or STX06	T												
BG04	U												
ISA08	X												
IVERREL	Overrides the version and release provided in the envelope profile member. It maps to type VR and LV standard data elements. DataInterchange will provide the equivalent data on Receive translation.												

- | **ISPW** Overrides the password that the trading partner profile member provides. It maps to a type PW standard data element. The envelope profile field that is overridden is:
- | This entry: | Overrides this envelope: |
|--------------------|---------------------------------|
| UNB12 | E |
| STX10 | T |
| ISA04 | X |
- DataInterchange will provide the equivalent data on Receive translation.
- | **IAPREF** Overrides the application reference that the envelope profile member provides. DataInterchange uses this reference as the message user class for EDIFACT and UNTDI. This reference maps to a type AP standard data element. As distributed by DataInterchange, no fields have the AP data type. However, the following fields are frequently customized to have a data type of AP.
- | This entry: | Overrides this envelope: |
|--------------------|---------------------------------|
| UNB14 | E |
| STX11 | T |
- DataInterchange will provide the equivalent data on Receive translation.
- | **GSID** Overrides the data format ID that trading partner transaction provides. If the trading partner transaction provides the name of an envelope profile member, this entry overrides the sender ID that the envelope profile member provides. The group application sender ID maps to a type AS standard data element. The envelope profile field that is overridden is:
- | This entry: | Overrides this envelope: |
|--------------------|---------------------------------|
| UNG02 | E |
| GS02 | X |
| GS02 | I |
| GS02 | U |
- DataInterchange will provide the equivalent data on Receive translation.
- | **GRID** Overrides the trading partner application name that the trading partner transaction provides. If the trading partner transaction provides the name of an envelope profile member, this entry overrides the receiver ID that the envelope profile member provides. The group application ID maps to a type AR standard data element. The envelope profile field that is overridden is:
- | This entry: | Overrides this envelope: |
|--------------------|---------------------------------|
| UNG04 | E |
| GS03 | X |
| GS03 | I |
| GS03 | U |
- DataInterchange will provide the equivalent data on Receive translation.
- | **GVER** Overrides the group version that the envelope profile member provides. The value in the group version maps to a type VR standard data element. DataInterchange will provide the equivalent data on Receive translation.
- | **GREL** Overrides the group release number that the envelope profile member provides. The group release number maps to type LV standard data element. DataInterchange will provide the equivalent data on Receive translation.
- | **GAPW** Overrides the group password that the trading partner usage provides. This usage overrides the password in the envelope profile member. The group password maps to a type PW standard data element. The envelope profile field that is overridden is:

This entry:	Overrides this envelope:
UNG13	E

DataInterchange will provide the equivalent data on Receive translation.

| **TVER** Overrides the transaction version that the envelope profile member provides. The transaction version maps to a type VR standard data element. DataInterchange will provide the equivalent data on Receive translation.

| **TREL** Overrides the transaction release number that the envelope profile member provides. The transaction release number maps to a type LV standard data element. DataInterchange will provide the equivalent data on Receive translation.

| **HOLDFLAG** Specifies if the transaction is in hold status. Valid values are:

Value	Description
-------	-------------

Y	Specifies DataInterchange should place the transaction in held status.
---	--

N	Specifies DataInterchange should not hold the transaction. Transactions that are not on hold and do not have a date specified for enveloping are available immediately for enveloping and sending.
---	--

DataInterchange will provide the equivalent data on Receive translation.

| **BNDLFLAG** Specifies if the transaction starts a bundle. BNDLFLAG valid values are:

Value	Description
-------	-------------

Y	Specifies the transaction starts a group of related transactions (a bundle). All transactions that follow are part of the bundle until the translator encounters another transaction with this field set to Y or N.
---	---

N	Specifies the translator ends the bundle without starting a new one.
---	--

(blank)

Specifies the translator continues processing as usual. The translator ends a bundle automatically if the data forces the start of a new group or interchange envelope. Any action that changes the transaction or store status of one member of the bundle such as envelope, send, and hold occurs to all members of the bundle.

Note: If the current trading partner is not using functional groups (value of N in the Functional Group field of the trading partner profile), the DataInterchange ignores the changes in the data. This would cause you to create a new group and does not end the bundles.

DataInterchange will provide the equivalent data on Receive translation.

| **ROUTCODE** A three-character generic routing code provided by the application and used by DataInterchange to select a generic send usage. A blank specifies a default generic send usage. DataInterchange will provide the equivalent data on Receive translation.

| **ISYNTAXID** Overrides the interchange syntax ID provided in the envelope profile member. The envelope profile fields that are overridden are:

Entry	Overrides Envelope:
UNB01	E
STX01	T

DataInterchange will provide the equivalent data on Receive translation.

| **ISYNTAXVER** Overrides the interchange syntax version provided in the envelope profile member. The envelope profile fields that are overridden are:

Entry	Overrides Envelope:
UNB02	E
STX02	T

DataInterchange will provide the equivalent data on Receive translation.

| **ISIDQ** Overrides the interchange sender ID qualifier provided in the envelope profile member or trading partner profile member. The envelope profile fields that are overridden are:

Entry	Overrides Envelope:
UNB04	E
ICS04	I
ISA05	X

DataInterchange will provide the equivalent data on Receive translation.

| **ISENDNAME** Overrides the interchange sender name provided in the envelope profile member. The envelope profile fields that are overridden are:

Entry	Overrides Envelope:
STX04	T
UCS03	U (If not IS data type)

DataInterchange will provide the equivalent data on Receive translation.

| **IREVROUT** Overrides the interchange reverse routing provided in the envelope profile member. The envelope profile field that is overridden is:

Entry	Overrides Envelope:
UNB05	E

DataInterchange will provide the equivalent data on Receive translation.

| **IRIDQ** Overrides the interchange receiver ID qualifier provided in the trading partner profile or envelope profile member. The envelope profile fields that are overridden are:

Entry	Overrides Envelope:
UNB06	E
ICS06	I
ISA07	X

DataInterchange will provide the equivalent data on Receive translation.

| **IrecvNAME** Overrides the interchange receiver name provided in the envelope profile member. The envelope profile fields that are overridden are:

Entry	Overrides Envelope:
STX06	T
UCS04	U (If not IR data type)

DataInterchange will provide the equivalent data on Receive translation.

| **IROUTADDR** Overrides the interchange routing address provided in the envelope profile member. The envelope profile field that is overridden is:

Entry	Overrides Envelope:
UNB08	E

DataInterchange will provide the equivalent data on Receive translation.

	ISTDID	Overrides the interchange standard ID provided in the envelope profile member. The envelope profile fields that are overridden are:
	Entry	Overrides Envelope:
	ICS02	I
	ISA11	X
		DataInterchange will provide the equivalent data on Receive translation.
	IPRIOR	Overrides the interchange processing priority provided in the envelope profile member. The envelope profile fields that are overridden are:
	Entry	Overrides Envelope:
	UNB15	E
	STX12	T
		DataInterchange will provide the equivalent data on Receive translation.
	ICOMMAGREE	Overrides the interchange communication agreement provided in the envelope profile member. The envelope profile field that is overridden is:
	Entry	Overrides Envelope:
	UNB17	E
		DataInterchange will provide the equivalent data on Receive translation.
	GSIDQ	Overrides the group sender ID qualifier provided in the envelope profile member. The envelope profile field that is overridden is:
	Entry	Overrides Envelope:
	UNG03	E
		Note: This entry must be left-justified.
		DataInterchange will provide the equivalent data on Receive translation.
	GRIDQ	Overrides the group receiver ID qualifier provided in the envelope profile member. The envelope profile field that is overridden is:
	Entry	Overrides Envelope:
	UNG05	E
		Note: This entry must be left-justified.
		DataInterchange will provide the equivalent data on Receive translation.
	GRES PAG	Overrides the group responsible agency code and controlling agency provided in the envelope profile member. The envelope profile fields that are overridden are:
	Entry	Overrides Envelope:
	UNG09	E
	GS07	I
	GS07	U
	GS07	X
		DataInterchange will provide the equivalent data on Receive translation.
	DUPTRANS	For Receive translation only. Specifies if a transaction is part of a duplicate envelope. Valid values are:
	Value	Description
	Y	Specifies a transaction that is part of a duplicate envelope.
	N	Specifies a transaction that is not part of a duplicate envelope.

FORCEC	For Send translation only, forces C record to be written. Valid values are:	
	Value	Description
	Y	Specifies that the C record should always be written.
	N	Specifies that the C record will be written only when an error occurs. This is default if no value is provided.
APPLTPID	The trading partner nickname of the application trading partner as it appears in the trading profile partner.	
EDITPID	The trading partner nickname of the EDI trading partner as it appears in the trading partner profile.	
CUSERDATA	This field is copied to the TRCB where it can be modified by user exits. Before any C record is output by DataInterchange, the value in this field is copied from the equivalent field in the TCRB. On Receive translation, the value in the TRCB can be set using the DataInterchange reserved variable DICUSERDATA. DataInterchange does not use this field.	

Data (D) Records

There are two formats for data (D) records. One format is used when all the data for a transaction is provided by a single structure. The other format is used when data for a transaction is provided by multiple structures. Table 2-46 and Table 2-47 on page 2-71 describe the D records.

Note: The largest record your application or the translator can handle is 32 K bytes, which is the largest LRECL allowed for QSAM files. During translate to application processing, DataInterchange divides any records that are larger than 32 K bytes into one or more X records, and uses one D record as the last record of the structure. For example, a structure that is 80 K bytes in length requires two X data records followed by one D record. During translate-to-standard processing, it is the responsibility of the program that creates the application file to create X and D records for those structures exceeding 32 K bytes in length. Define the application file as variable blocked (VB) if the application file is to contain X records.

Table 2-46. D Record — All Structures Passed Together

Label	Location	Type	Length	Description
RECID	1	Character	1	Record identifier
DATARCD	2-32750	Character	32749	Application transaction data

Note: When translating D records, use the map named in MAPID, and populate ISA and GS with the values passed in the C record.

The following are descriptions of the labels for the data record when all structures are passed together.

Label	Description
RECID	Specifies if a transaction exceeds 32 K bytes. Valid values are:
	Value Description
	D Specifies an entire transaction, or the last record of a transaction that exceeds 32 K bytes.
	X Specifies the first and middle records of a transaction that exceed 32 K bytes.
DATARCD	Specifies application transaction data.

Table 2-47. D Record — Structures Passed Separately

Label	Location	Type	Length	Description
RECID	1	Character	1	Record identifier
STRUCNAM	2-17	Character	16	Structure name
DATARCD	18-32750	Character	32733	Application transaction data

Note: When translating D records, use the map named in MAPID, and populate ISA and GS with the values passed in the C record.

The following are descriptions of the labels for the data record when the data structures are passed separately.

Label	Description
RECID	Specifies if a transaction exceeds 32 K bytes. Valid values are:
	Value Description
	D Specifies an entire transaction, or the last record of a transaction that exceeds 32 K bytes.
	X Specifies the first and middle records of a transaction that exceed 32 K bytes.
STRUCNAM	The structure name defined in the application data format for this transaction.
DATARCD	Application transaction data.

Note: When translating D records, use the map named in MAPID, and populate ISA and GS with the values passed in the C record.

For a Fixed-to-Fixed mapping when there is no target application data format, the STRUCTNAME is the segment ID value from the standard transaction definition. The D records output during Fixed-to-Fixed mapping are always in the format described by Table 2-47.

End Transaction and Interchange (Z) Records

The Z record is optional and its use need only be considered in the CICS environment when recoverable resources are being used or in the MVS environment when the application wants to control the size of an interchange.

The Z record has two purposes:

1. The Z record marks the end of the application data for a transaction. If a Z record is not present, DataInterchange can only detect the end of a transaction from reading the next transaction's C record. If your input data is in recoverable intra-partition transient data (TD) queues, the use of the Z record keeps the reading of the C record for the next transaction from being part of the syncpoint interval for the current transaction. The Z record provides a formal end of a transaction rather than the implied transaction end achieved when the next C record is read.
2. A Z record with the ENDINTERCH field set to 1 (Z1) not only marks the end of the application data for a transaction but also indicates that the current transaction is the last transaction for an interchange. If a Z1 record is not present, DataInterchange can only detect the end of a interchange from reading the next transaction's C record. If your input data is in recoverable intra-partition transient data (TD) queues, the use of the Z1 record keeps the reading of the C record for the next transaction from being part of the syncpoint interval for the current interchange. The Z1 record provides a formal end of an interchange rather than the implied end achieved when the next C record is read. The Z1 record can

be used to artificially limit the size of an interchange. A Z1 record causes the current interchange to be completed. A new interchange starts with the next transaction.

Z and Z1 records are only possible when using the C and D record format. Use the Z and Z1 records when the conditions mentioned below are true. In other conditions, use of Z and Z1 is optional.

- The DataInterchange Utility is executing in the CICS environment.
- Application data is given to the DataInterchange Utility in recoverable intra-partition TD queues.
- DataInterchange is allowed to issue CICS SYNCPOINT commands.

Table 2-48 describes the format of Z records.

Table 2-48. Z Record

Label	Location	Type	Length	Description
RECID	1	Character	1	Record identifier
ENDINTERCH	2	Character	1	End of Interchange indicator

Z Record Label Descriptions: The following are descriptions of the labels for the Z record.

Label	Description
RECID	RECID uses the value Z to identify this record as a transaction or interchange terminator.
ENDINTERCH	A character value of 1 indicates the preceding transaction the last in the current interchange. The next C record begins a new transaction and new interchange.

Raw Data Records

Table 2-49 describes the format of raw data records. The application data format must indicate the location, length, and type of record ID field within the data. The application data format can also provide the field within the application data that contains the internal trading partner ID value. For TRANSLATE TO STANDARD functions, the application data format ID is provided by the RAWFMTID keyword on the PERFORM command. During translate-to-application processing, the translator automatically supplies the record ID values and the value of the internal trading partner ID taken from the trading partner receive usage record when raw data is requested.

DataInterchange can create raw data output during TRANSLATE TO STANDARD functions when Fixed-to-Fixed mapping applies. For Fixed-to-Fixed mappings, the RAWDATA keyword indicates if data should be written in a raw data format. Record ID values and the internal trading partner ID value are provided as default values but may be overridden in the mapping process.

Table 2-49. Raw Data Record

Label	Location	Type	Length	Description
DATARCD	1-32750	Character	32750	Application transaction data

Optional Records

You can request the optional records using the TYPE fields in the control record (see “Control Record (C)” on page 2-61), the Additional Records window (TF53), or the OPTRECS keyword (see OPTRECS on page 1-98) in one of the following commands:

- TRANSLATE TO STANDARD
- TRANSLATE AND ENVELOPE

- TRANSLATE AND SEND
- TRANSLATE TO APPLICATION
- DEENVELOPE
- DEENVELOPE AND TRANSLATE
- RECEIVE AND TRANSLATE

The text below shows which PERFORM commands support each of the optional records. All of the optional records do not apply to every command.

- Information (I) record
 - TRANSLATE TO STANDARD
 - TRANSLATE AND ENVELOPE
 - TRANSLATE AND SEND
 - ENVELOPE
 - DEENVELOPE
 - TRANSLATE TO APPLICATION
- Envelope header (E) records
 - TRANSLATE AND ENVELOPE
 - TRANSLATE AND SEND
 - ENVELOPE
 - DEENVELOPE
- Group header (G) records
 - TRANSLATE AND ENVELOPE
 - TRANSLATE AND SEND
 - ENVELOPE
 - DEENVELOPE
- Transaction set header (T) records
 - TRANSLATE AND ENVELOPE
 - TRANSLATE AND SEND
 - ENVELOPE
 - DEENVELOPE
 - TRANSLATE TO APPLICATION
- Queuing totals (Q) records
 - TRANSLATE AND ENVELOPE
 - TRANSLATE AND SEND
 - ENVELOPE
 - DEENVELOPE

For TRANSLATE TO STANDARD operations, the DataInterchange Utility writes the optional records to a file as described below.

- Writes the records to FFSTRAK if it exists.
- Writes the records to FFSEXCP if FFSTRAK does not exist.
- Writes no records if FFSTRAK does not exist and raw data (RAWFMTID keyword) is requested.

For TRANSLATE TO APPLICATION operations, the DataInterchange Utility writes the optional records to a file as described below.

- Writes the records to the application output file if C and D records are requested.
- Writes the records to FFSEXCP if the raw data records are requested (RAWDATA keyword).

For DEENVELOPE operations, the DataInterchange Utility writes the optional records to the FFSEXCP file.

Information (I) Record

Table 2-50 describes the format and contents of the information (I) record. You can request it using the Optional Record Options window, the ITYPE field in the control record, or the OPTRECS keyword in a command. The DataInterchange Utility returns one record for each transaction. The record contains the values that were present at the time the transaction was translated.

Table 2-50 (Page 1 of 2). Information Record

Label	Location	Type	Length	Description
RECID	1	Character	1	Record identifier. The value I identifies this record as an information record.
IHXCTL	2-15	Character	14	Control number of interchange header.
ISID	16-50	Character	35	Interchange sender ID.
IRID	51-85	Character	35	Interchange receiver ID.
IDATE	86-91	Character	6	Interchange date.
ITIME	92-97	Character	6	Interchange time.
IVERREL	98-102	Character	5	Interchange version/release.
IGT	103-108	Character	6	For outbound, the total number of groups within the interchange at the current time. For inbound, the total number of groups processed so far within the interchange.
ITT	109-114	Character	6	For outbound, the total number of transactions within the interchange at the current time. For inbound, the total number of transactions processed so far within the interchange.
IST	115-124	Character	10	For outbound, the total number of segments within the interchange at the current time. For inbound, the total number of segments processed so far within the interchange.
IBT	125-132	Character	8	For outbound, the total number of bytes within the interchange at the current time. For inbound, the total number of bytes processed so far within the interchange.
ISPW	133-146	Character	14	Interchange password.
IAPREF	147-160	Character	14	Interchange application reference.
GHXCTL	161-174	Character	14	Control number of group header.
GFGID	175-180	Character	6	Group functional group ID.
GSID	181-215	Character	35	Group application sender ID.
GRID	216-250	Character	35	Group application receiver ID.
GDATE	251-256	Character	6	Group date.
GTIME	257-262	Character	6	Group time.
GVER	263-274	Character	12	Group version.

Table 2-50 (Page 2 of 2). Information Record

Label	Location	Type	Length	Description
GREL	275-286	Character	12	Group release.
GTT	287-292	Character	6	Current group transaction total (send). Total transactions in group (receive).
GAPW	293-306	Character	14	Group password.
THXCTL	307-320	Character	14	Control number of transaction set header.
TTC	321-326	Character	6	Transaction code.
TVER	327-332	Character	6	Transaction version.
TREL	333-338	Character	6	Transaction release.
TST	339-348	Character	10	Transaction segment total.
AC	349-383	Character	35	Application control field value.
THANDLE	384-403	Character	20	The expanded 20-byte value of the handle assigned to this transaction in the Transaction Store.
TPNICKN	404-419	Character	16	Trading partner nickname.
GDATE2	420-427	Character	8	Group date with century.
RESERVED	428-483	Character	56	Reserved.

Interchange Header (E) Record

Table 2-51 shows the format and contents of the interchange (E) header record. You can request it using ETYPE in the control record or the OPTRECS keyword in a command. The DataInterchange Utility returns a record for each interchange that is created or received.

Table 2-51. Interchange Header

Label	Location	Type	Length	Description
RECID	1	Character	1	Record identifier. The value E identifies this record as an interchange header.
EDATA	2-256	Character	255	Standard interchange header image.

Group Header (G) Record

Table 2-52 shows the format and contents of the group header (G) record. You request it using GTYPE in the control record or the OPTRECS keyword in a command. The DataInterchange Utility returns a record for each group created or received.

Table 2-52. Group Header

Label	Location	Type	Length	Description
RECID	1	Character	1	Record identifier. The value G identifies this record as a group header.
GDATA	2-256	Character	255	Functional group header image.

Transaction Set Header (T) Record

Table 2-53 on page 2-76 shows the format and contents of the transaction set header (T) record. You can request it using TTYPE in the control record or the OPTRECS keyword in a command. The DataInterchange Utility returns a record for each transaction created or received.

Table 2-53. Transaction Set Header

Label	Location	Type	Length	Description
RECID	1	Character	1	Record identifier. The value T identifies this record as a transaction set header.
TDATA	2-256	Character	255	Transaction set header image.

Queuing Totals (Q) Record

Table 2-54 shows the format and contents of the queuing totals record. You can request it using QTYPE in the control record or the OPTRECS keyword in a command. The DataInterchange Utility returns a record to the exception file each time an interchange is queued or deenveloped.

Table 2-54. Queuing Totals

Label	Location	Type	Length	Description
RECID	1	Character	1	Record identifier. The value Q identifies this record as the queuing totals for an envelope.
QBT	2-9	Character	8	Number of bytes queued for the envelope.
QST	10-19	Character	10	Number of segments in the envelope.
QTT	20-25	Character	6	Number of transactions in the envelope.
QGT	26-31	Character	6	Number of groups in the envelope.
QDSNAME	32-87	Character	56	The physical data set name from which transactions were read or to which transactions were written. The name is terminated with a NULL character (X'00').

Management Reporting

The management reporting data extracts are formatted as a sequential file of fixed-length records. The output is tabular with columns representing categories of information and rows being the actual data entries. The format of the data extracts is described in this section. All data extracts are written to the EDIQUERY file. All management reporting extracts records are fixed length 1024. Since other commands use the EDIQUERY file though, it should be defined variable block 32756 rather than fixed block 1024 to accommodate commands that output larger records; for example, IMAGE records created by PERFORM ENVELOPE DATA EXTRACT and PERFORM TRANSACTION DATA EXTRACT.

Trading Partner Profile Data Extract

Table 2-55 on page 2-77 describes the Trading Partner Profile data extract.

Table 2-55. Trading Partner Information Data Extract

Label	Location	Type	Length	Description
Record ID	1-3	Character	3	"TPI" for Trading Partner Information
Trading Partner Nickname	4-19	Character	16	The trading partner ID within DataInterchange
Company Name	20-59	Character	40	Company name of the trading partner
Address line 1	60-99	Character	40	First line of the company's address
Address line 2	100-139	Character	40	Second line of the company's address
Comment line 1	140-179	Character	40	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Comment line 2	180-219	Character	40	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Contact Name	220-249	Character	30	Name of a contact
Contact Phone	250-274	Character	25	Phone number of the contact
Network ID	275-282	Character	8	ID of the network used to communicate with this trading partner
Interchange ID	283-317	Character	35	Receiver ID used in interchanges to this trading partner
Account ID	318-349	Character	32	Network Account ID
User ID	350-381	Character	32	Network User ID
Direction	382	Character	1	Direction of the map (inbound or outbound)
Date of Last Transmission	383-390	Character	8	Interchange date of last transmission to this trading partner
Interchange control number	391-404	Character	14	Interchange control number of last transmission to this trading partner
Group control number	405-418	Character	14	Group control number of the last transmission to this trading partner
Transaction control number	419-432	Character	14	Transaction control number of the last transaction to this trading partner
Filler	433-1024	Character	592	Filler for expansion

Trading Partner Capability Data Extract

Table 2-56 describes the Trading Partner Capability data extract.

Table 2-56 (Page 1 of 2). Trading Partner Capability Information Data Extract

Label	Location	Type	Length	Description
Record ID	1-3	Character	3	"TPC" for Trading Partner Capability
Trading Partner Nickname	4-19	Character	16	The trading partner ID within DataInterchange
Internal Trading Partner ID	20-54	Character	35	ID used for this trading partner internally (customer number, supplier number)

Table 2-56 (Page 2 of 2). Trading Partner Capability Information Data Extract

Label	Location	Type	Length	Description
Company Name	55-94	Character	40	Company name of the trading partner
Address line 1	95-134	Character	40	First line of the company's address
Address line 2	135-174	Character	40	Second line of the company's address
Comment line 1	175-214	Character	40	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Comment line 2	215-254	Character	40	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Direction	255	Character	1	Direction of the map (inbound or outbound)
Standard ID	256-263	Character	8	ID of the standard (X12, EDIFACT)
Version ID	264-265	Character	2	Version of the standard
Release ID	266-267	Character	2	Release of the standard
Description	268-217	Character	50	Description of the standard/version/release
Transaction ID	318-325	Character	8	ID of the transaction (ORDERS, DISPATCH, 850, 860)
Map ID	326-341	Character	16	Trading partner transaction ID
Measurement date	342-349	Character	8	Date testing or production started with this map/trading partner
Measurement ID	350-353	Character	4	Type of statistic, CPTR = Cumulative Production Transactions, CTTR = Cumulative Test Transactions
Total number of transactions	354-368	Character	15	Total number of test or production transactions exchanged with this map/trading partner
Total errors	369-383	Character	15	Total number of test or production transactions exchanged with this map/trading partner that had errors
Filler	384-1024	Character	641	Filler for expansion

Network Activity Data Extract

Table 2-57 describes the Network Activity data extract.

Table 2-57 (Page 1 of 2). Network Traffic Data Extract

Label	Location	Type	Length	Description
Record ID	1-3	Character	3	"NTA" for Network Traffic Activity
Requestor ID	4-19	Character	16	Requestor identification
Network ID	20-27	Character	8	Network identification
Network name	28-57	Character	30	Network descriptive name
Account number	58-89	Character	32	Network Account Number
User ID	90-121	Character	32	Network User ID

Table 2-57 (Page 2 of 2). Network Traffic Data Extract

Label	Location	Type	Length	Description
Direction	122	Character	1	Direction of the map (inbound or outbound)
Charge Code	123	Character	1	Network charge code
Measurement ID	124-127	Character	4	Type of statistic, DACM = Daily Communications Measurement
Day	128-135	Character	8	Measurement date
Interchange envelopes	136-150	Character	15	The total number of interchange envelopes
Total characters	151-167	Character	15	The total number of characters sent
Filler	166-1024	Character	859	Filler for expansion

Transaction Activity Data Extract

Table 2-58 describes the Transaction Activity data extract.

Table 2-58 (Page 1 of 2). Transaction Activity Report Data

Label	Location	Type	Length	Description
Record ID	1-3	Character	3	TPA for Trading Partner Activity
Trading Partner Nickname	4-19	Character	16	The trading partner ID within DataInterchange
Internal Trading Partner ID	20-54	Character	35	ID used for this trading partner internally (customer number, supplier number)
Company Name	55-94	Character	40	Company name of the trading partner
Address line 1	95-134	Character	40	First line of the company's address
Address line 2	135-174	Character	40	Second line of the company's address
Comment line 1	175-214	Character	40	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Comment line 2	215-254	Character	40	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Direction	255	Character	1	Direction of the transactions (inbound or outbound)
Standard ID	256-263	Character	8	ID of the standard (X12, EDIFACT)
Version ID	264-265	Character	2	Version of the standard
Release ID	266-267	Character	2	Release of the standard
Description	268-317	Character	50	Description of the standard/version/release
Transaction ID	318-325	Character	8	ID of the transaction (ORDERS, DESPATCH, 850, 860)
Map ID	326-341	Character	16	Trading partner transaction ID
Application data format ID	342-357	Character	16	Name of the application data format.

Table 2-58 (Page 2 of 2). Transaction Activity Report Data

Label	Location	Type	Length	Description
Measurement ID	358-361	Character	4	Type of statistic: DPTR = Daily Production Transactions, DTTR = Daily Test Transactions
Measurement Date	362-369	Character	8	Date of this statistic
Total transactions	370-384	Character	15	Total transactions for the indicated date
Total errors	385-399	Character	15	Total transactions in error for the indicated date
Filler	400-1024	Character	625	Filler for expansion

Transaction Store Data Extract Information Categories

Information can be extracted from the DataInterchange Transaction Store databases using the PERFORM ENVELOPE DATA EXTRACT and the PERFORM TRANSACTION DATA EXTRACT commands. See “Reporting Using Management Reporting and Transaction Store” on page 1-48 for a description of these PERFORM commands. There are various categories of information that may be requested which are described in the table below.

Category	Description	PERFORM KEYWORD
E	Interchange data	INTERCHANGE(Y)
G	Group data	GROUP(Y)
T	Transaction data	TRANSACTION(Y)
A	Application data	APPLICATION(Y)
R	Transaction image	IMAGE(Y)
E,G,T	Send Acknowledgment data	SENDACKDATA(Y)
E,G,T	Receive Acknowledgment data	RECEIVEACKDATA(Y)
F,K	Send Acknowledgment image	SENDACKIMAGE(Y)
F,K	Receive Acknowledgment image	RECEIVEACKIMAGE(Y)

All data extracted from the Transaction Store is written to the EDIQUERY file. The information categories are either written as separate records (CONCATENATE(N)) or combined and written as a single record (CONCATENATE(Y)).

The following rules apply when requesting information categories.

1. An IMAGE will not be produced unless the TRANSACTION is also requested.
2. IMAGE categories (R, F, and K) are always written as separate records even when CONCATENATE(Y) has been requested.
3. SENDACKDATA will not be produced unless you request the GROUP record and if you want Transaction Ack Data (824 transaction), you must request the TRANSACTION record.
4. SENDACKDATA (E, G, and T records) will always be concatenated to its corresponding GROUP or TRANSACTION record even when CONCATENATE(N) has been requested.
5. RECEIVEACKDATA will not be produced unless GROUP or TRANSACTION has been requested.
6. RECEIVEACKDATA (E, G, and T records) will always be concatenated to its corresponding GROUP or TRANSACTION record even when CONCATENATE(N) has been requested.

Transaction Store Data Extract Common Key

All records created by Transaction Store Data Extract command have a common 141-byte key field that begins with a 3-character record ID. Portions of the key that do not apply to a particular record are initialized with blanks. Table 2-59 on page 2-81 shows which portions of the key apply to which record types.

Table 2-59. Format of the Common Key

Label	Location	Type	Length	Used In	Description
Record ID	1	Character	3	E G T A R K	Identifies the record
Nickname	4	Character	16	E G T A R K	Trading partner nickname
Direction	20	Character	1	E G T A R K	Direction (S/R)
Control Number	21	Character	14	E G T A R K	Interchange control number
Receiver ID	35	Character	35	E G T A R K	Interchange receiver ID
Control Number	70	Character	14	G T A R K	Group control number
Control Number	84	Character	14	T A R K	Transaction control number
Controlling handle	98	Character	20	T A R K	Handle value of controlling transaction YYYYMMDDHHMMSSnnnnnn
Transaction handle	118	Character	20	T A R K	Handle value of transaction YYYYMMDDHHMMSSnnnnnn
Sequence Number	138	Character	4	A	Sequence number

Transaction Store Data Extract Record Formats

The following tables show the formats for the records created by Transaction Store Data Extract command. All tables, except images, are padded to a length of 1024 bytes to leave room for expansion. If concatenation is requested in the DataInterchange Utility control statements, then the full records as described are concatenated into a single record. Images are written separately and use the full logical record length of the EDIQUERY dataset.

Interchange Data Extract Record Layout

Table 2-60 shows the interchange data extract record layout.

Table 2-60 (Page 1 of 3). Interchange Data Extract Record Layout

Label	Location	Type	Length	Description
Record ID	1-3	Character	3	Record ID = E Note: A record ID of E1 indicates an interchange record for functional acknowledgment detail data. A record ID of E2 indicates an interchange record for transaction acknowledgment detail data.
Nickname	4-19	Character	16	Trading partner nickname

Table 2-60 (Page 2 of 3). Interchange Data Extract Record Layout

Label	Location	Type	Length	Description
Direction	20	Character	1	Direction (S/R)
Control Number	21-34	Character	14	Interchange control number
Receiver ID	35-69	Character	35	Interchange receiver ID
Filler	70-141	Character	72	Always contains blanks
Sender ID	142-176	Character	35	Interchange sender ID
Fake flag	177	Character	1	Interchange did not have an interchange header (Y/N)
Sequence error flag	178	Character	1	Y if this interchange is out of sequence. This field only gets set to a Y when a PERFORM ENVELOPE DATA EXTRACT is performed.
Usage indicator	179	Character	1	Indicates usage
Duplicate interchange flag	180	Character	1	Duplicate when received (Y/N)
Envelope date	181-194	Character	14	Date/time envelope created YYYYMMDDHHMMSS
Send date	195-208	Character	14	Date/time envelope sent YYYYMMDDHHMMSS
TA1 acknowledgment	209	Character	1	TA1 acknowledgment received
TA1 date	210-223	Character	14	Date/time TA1 received YYYYMMDDHHMMSS
Network status code	224-225	Character	2	Coded value for the network status
Network status text	226-245	Character	20	Network status code in a text format
Acknowledgment expected	246	Character	1	Network acknowledgment expected
Acknowledgment received	247	Character	1	Network acknowledgment received
Acknowledged date	248-261	Character	14	Date/time envelope of network acknowledgment YYYYMMDDHHMMSS
Message user class	262-269	Character	8	Message user class assigned when sent
Message name	270-277	Character	8	Message name assigned when sent
Sequence number	278-282	Character	5	Sequence number assigned when sent
Message ID	283-290	Character	8	Message ID assigned when sent
Physical data set name	291-346	Character	56	Physical data set name to which data was queued
Group count	347-357	Character	11	Number of groups in interchange
Transaction count	358-368	Character	11	Number of transactions in interchange
Segment count	369-379	Character	11	Number of segments in interchange
Interchange size	380-390	Character	11	Number of bytes in interchange
Interchange header	391-640	Character	250	Image of the interchange header
Interchange trailer	641-670	Character	30	Image of the interchange trailer

Table 2-60 (Page 3 of 3). Interchange Data Extract Record Layout

Label	Location	Type	Length	Description
Filler	671-1024	Character	354	Filler for expansion

Group Data Extract Record Layout

Table 2-61 shows the group data extract record layout.

Table 2-61. Group Data Extract Record Layout

Label	Location	Type	Length	Description
Record ID	1-3	Character	3	Record ID = G Note: A record ID of G1 indicates a group record for functional acknowledgment detail data. A record ID of G2 indicates a group record for transaction acknowledgment detail data.
Nickname	4-19	Character	16	Trading partner nickname
Direction	20	Character	1	Direction (S/R)
Control Number	21-34	Character	14	Interchange control number
Receiver ID	35-69	Character	35	Interchange receiver ID
Control Number	70-83	Character	14	Group control number
Filler	84-141	Character	58	Always contains blanks
Fake flag	142	Character	1	Interchange did not have groups (Y/N)
Sender ID	143-177	Character	35	Application sender ID
Receiver ID	178-212	Character	35	Application receiver ID
Acknowledgment expected	213	Character	1	Functional acknowledgment expected (Y/N)
Acknowledgment received	214	Character	1	Functional acknowledgment received (Y/N) Note: If an acknowledgment is not expected then this field is blank.
Acknowledgment date	215-228	Character	14	Date/time group acknowledgment received YYYYMMDDHHMMSS
Acknowledgment handle	229-248	Character	20	The handle for functional acknowledgment transaction with a format of YYYYMMDDHHMMSSnnnnnn
Transaction count	249-259	Character	11	Number of transactions in group
Segment count	260-270	Character	11	Number of segments in group
Group size	271-281	Character	11	Number of bytes in group
Group header	282-434	Character	153	Image of the group header
Group trailer	435-460	Character	26	Image of the group trailer
Filler	461-1024	Character	564	Filler for expansion

Transaction Data Extract Record Layout

Table 2-62 on page 2-84 shows the transaction data extract record layout.

Table 2-62 (Page 1 of 3). Transaction Data Extract Record Layout

Label	Location	Type	Length	Description
Record ID	1-3	Character	3	Record ID = T Note: A record ID of T1 indicates a transaction record for functional acknowledgment detail data. A record ID of T2 indicates a transaction record for transaction acknowledgment detail data.
Nickname	4-19	Character	16	Trading partner nickname
Direction	20	Character	1	Direction (S/R)
Control Number	21-34	Character	14	Interchange control number
Receiver ID	35-69	Character	35	Interchange receiver ID
Control Number	70-83	Character	14	Group control number
Control Number	84-97	Character	14	Transaction control number
Controlling handle	98-117	Character	20	Handle value of controlling transaction YYYYMMDDHHMMSSnnnnnn
Transaction handle	118-137	Character	20	Handle value of transaction YYYYMMDDHHMMSSnnnnnn
Filler	138-141	Character	4	Always contains blanks
Enveloped date	142-155	Character	14	Date/time transaction put into envelope YYYYMMDDHHMMSS
Creation date	156-169	Character	14	Date/time transaction put into the store
Transaction status code	170-171	Character	2	Current transaction status
Transaction status text	172-191	Character	20	Transaction status in a text format
Acknowledgment received	192	Character	1	Group acknowledgment received
Acknowledgment received text	193-212	Character	20	Group acknowledgment received in a text format
Acknowledgment date	213-226	Character	14	Date/time group acknowledgment received YYYYMMDDHHMMSS
Trx Acknowledgment expected	227	Character	1	Transaction acknowledgment expected
Trx Acknowledgment received	228	Character	1	Transaction acknowledgment received
Trx Acknowledgment received text	229-248	Character	20	Trx acknowledgment received in a text format
Trx Acknowledgment date	249-262	Character	14	Date/time transaction acknowledgment received YYYYMMDDHHMMSS
Acknowledgment handle	263-282	Character	20	The handle for transaction acknowledgment transaction with a format of YYYYMMDDHHMMSSnnnnnn

Table 2-62 (Page 2 of 3). Transaction Data Extract Record Layout

Label	Location	Type	Length	Description
Segment count	283-293	Character	11	Number of segments in transaction
Transaction size	294-304	Character	11	Number of bytes in transaction
Enveloped segment count	305-315	Character	11	Number of segments in transaction when enveloped
Enveloped transaction size	316-326	Character	11	Number of bytes in transaction when enveloped
Format ID	327-342	Character	16	Last Application Data Format ID
Transaction ID	343-350	Character	8	Standard Transaction
Group ID	351-356	Character	6	Function Group ID associated with the transaction
Envelope member	357-364	Character	8	Member name used for enveloping
Envelope type	365	Character	1	Type of envelope associated with transaction
Network ID	366-373	Character	8	Network associated with trading partner
Standard ID	374-381	Character	8	Standard ID of the transaction
Standard Version	382-383	Character	2	Standard Version
Standard Level	384-385	Character	2	Standard Level
Internal Trading Partner ID	386-420	Character	35	Last internal trading partner ID
Application control field	421-455	Character	35	Last application control field
Delivery date	456-469	Character	14	Last date given/got from application YYYYMMDDHHMMSS
Earliest envelope date	470-477	Character	8	Earliest date transaction will be enveloped YYYYMMDD
Earliest purge date	478-485	Character	8	Earliest date transaction will be automatically purged YYYYMMDD
Error level	486	Character	1	Translation error level
Store status	487-488	Character	2	Store status of the transaction
Store status text	489-508	Character	20	Store status in a text format
Override flag	509	Character	1	Envelope overrides provided for transaction
Held flag	510	Character	1	Transaction held flag
Test flag	511	Character	1	Test transaction flag
Duplicate flag	512	Character	1	Transaction part of duplicate envelope
Purge flag	513	Character	1	Purge flag
Translate flag	514	Character	1	Translated flag
Detached flag	515	Character	1	Transaction detached flag
Override handle	516-535	Character	20	Handle for enveloping overrides YYYYMMDDHHMMSSnnnnnn
Security profile name	536-543	Character	8	Security profile name for GROUP LEVEL encryption

Table 2-62 (Page 3 of 3). Transaction Data Extract Record Layout

Label	Location	Type	Length	Description
Encryption key	544-559	Character	16	Encryption key name for GROUP LEVEL encryption
Authentication key	560-575	Character	16	Authentication key name for GROUP LEVEL encryption
Application assigned control number	576-589	Character	14	Transaction control number if assigned by the application
Data element delimiter	590	Character	1	Data element delimiter used when transaction was translated
Sub element-delimiter	591	Character	1	Subelement delimiter used when transaction was translated
Segment terminator	592	Character	1	Segment terminator used when transaction was translated
Decimal notation	593	Character	1	Decimal notation used when transaction was translated
Release character	594	Character	1	Release character used when transaction was translated
Segment ID separator	595	Character	1	Segment ID separator used when transaction was translated
Transaction header	596-680	Character	85	Image of the transaction header
Transaction trailer	681-706	Character	26	Image of the transaction trailer
Last Mapid	707-722	Character	16	Last map used
Filler	723-1024	Character	302	Filler for expansion

Application Data Extract Record Layout

Table 2-63 shows the application data extract record layout.

Table 2-63 (Page 1 of 2). Application Data Extract Record Layout

Label	Location	Type	Length	Description
Record ID	1-3	Character	3	Record ID = A
Nickname	4-20	Character	16	Trading partner nickname
Direction	20	Character	1	Direction (S/R)
Control Number	21-34	Character	14	Interchange control number
Receiver ID	35-69	Character	35	Interchange receiver ID
Control Number	70-83	Character	14	Group control number
Control Number	84-97	Character	14	Transaction control number
Controlling handle	98-117	Character	20	Handle value of controlling transaction YYYYMMDDHHMMSSnnnnnn
Transaction handle	118-137	Character	20	Handle value of transaction YYYYMMDDHHMMSSnnnnnn
Sequence Number	138-141	Character	4	Sequence number
Format ID	142-157	Character	16	Application Data Format ID
Application ID	158-165	Character	8	Application ID

Table 2-63 (Page 2 of 2). Application Data Extract Record Layout

Label	Location	Type	Length	Description
BATCH ID	166-173	Character	8	Batch ID
Delivery Date	174-187	Character	14	Date/time delivered to application YYYYMMDDHHMMSS
Error level	188	Character	1	Translation error level
Acceptable error level	189	Character	1	Acceptable error level
Error count	190-200	Character	11	Number of errors found
Filler	201-1024	Character	824	Filler for expansion

Transaction/Acknowledgment Image Data Extract Record Layout

Table 2-64 shows the image data record layout.

Table 2-64. Image Record Layout

Label	Location	Type	Length	Description	
Record ID	1-3	Character	3	Record ID. Valid values are:	
				Value	Description
				RX	Transaction image continued
				RZ	Final transaction image record
				FX	Functional acknowledgment image continued
				FZ	Final functional acknowledgment image record
				KX	Transaction acknowledgment image continued
KZ	Final transaction acknowledgment image record				
Nickname	4-19	Character	16	Trading partner nickname	
Direction	20	Character	1	Direction (S/R)	
Control Number	21-34	Character	14	Interchange control number	
Receiver ID	35-69	Character	35	Interchange receiver ID	
Control Number	70-83	Character	14	Group control number	
Control Number	84-97	Character	14	Transaction control number	
Controlling handle	98-117	Character	20	Handle value of controlling transaction YYYYMMDDHHMMSSnnnnnn	
Transaction handle	118-137	Character	20	Handle value of transaction YYYYMMDDHHMMSSnnnnnn	
FILLER	138-141	Character	4	Always contains blanks	
Total size	142-152	Character	11	Total size of the image	
Record size	153-163	Character	11	Size of image data in this record	
Image	164-end	Character	Variable	Transaction or Acknowledgment Image	

Chapter 3. Using the DataInterchange Application Program Interface

API Languages	3-1
API Link Edit	3-2
DataInterchange for MVS and DB2 Attachment	3-2
EDITSIN Examples	3-3
DataInterchange for CICS Abend Return Codes	3-3
Calls Using COBOL	3-5
SNB-COBOL	3-5
CCB-COBOL	3-5
FCB-COBOL	3-6
INIT-COBOL	3-6
Calls Using PL/I	3-6
SNB-PL/I	3-6
CCB-PL/I	3-7
FCB-PL/I	3-7
INIT-PL/I	3-7
Calls Using C	3-8
SNB-C	3-8
CCB-C	3-8
FCB-C	3-9
INIT-C	3-9
Calls Using Assembler	3-9
SNB-Assembler	3-10
CCB-Assembler	3-10
FCB-Assembler	3-11
USER-DSECT for Initialization Sample	3-11
INIT-Assembler	3-11
API Business Tasks	3-12
API Function Overview	3-13
Environmental Services Overview	3-14
Translation Services Overview	3-14
Enveloping Services Overview	3-14
Extraction Services Overview	3-15
Communications Services Overview	3-15
Status Update Services Overview	3-16
SYNCPOINT Services Overview	3-16
Environmental Services	3-17
API - Initialization	3-17
API - Utility Services	3-18
API - Termination	3-19
Translation Services	3-20
Translate to Standard	3-22
Enveloping and Sending	3-22
Test Translate to Standard	3-25
Translate to Standard Data Modes	3-25
API - Translate to Standard	3-27
First Call of Session (TS)	3-28
First Call for Transaction (TS)	3-30
Subsequent Calls	3-36
Last Call for Transaction (TS)	3-36

Last Call for Session (TS)	3-39
Special Considerations (TS)	3-40
Outbound Incremental Translation	3-44
Pageable Translation	3-45
Translate to Application	3-45
Receiving and Deenveloping	3-46
Test Translate to Application	3-49
API - Translate to Application	3-49
API - Translate Specific	3-49
First Call of Session (TA)	3-50
First Call of Transaction (TA)	3-52
Subsequent Calls (TF/TA)	3-56
Last Call of Transaction (TA)	3-57
Last Call of Session (TA)	3-57
API - Translate File	3-57
First Call of Session (TF)	3-58
First Call of Transaction (TF)	3-61
Subsequent Calls (TF/TA)	3-67
Last Call of Transaction (TF)	3-68
Last Call of Session (TF)	3-68
Special Considerations (TA)	3-68
Partial Structures (TA)	3-68
Single Unit of Work (TA)	3-68
Clustered Transactions (TA)	3-69
Enveloping Services	3-69
Interchange Layer	3-71
Group Layer	3-71
Transaction Layer	3-72
Enveloping Service	3-72
Envelope Function	3-72
API - Envelope	3-73
Initializing for Envelope API	3-74
Envelope Transaction	3-76
Last Call of Session (EV)	3-81
Special Considerations	3-81
Envelope Versus Reenvelope	3-81
Clustered Transactions (EV)	3-82
Sending Transaction Data	3-82
API - Close and Queue Interchange	3-84
API - End Translation/Enveloping	3-85
Deenvelope Function	3-86
API - Deenvelope	3-86
Initializing for Deenvelope API	3-87
Deenvelope Transaction	3-90
Last Call of Session (DE)	3-96
Special Considerations	3-96
Locate EDI and Application Trading Partners	3-96
Locate Receive Map	3-97
API - Issue Commit	3-98
API - Retrieve Interchange Header	3-99
API - Retrieve Group Header	3-100
API - Retrieve Transaction Header	3-100
Data Extraction Services	3-101
Initialization for Data Extract	3-101

API - Retrieve Detailed Data	3-102
API - Retrieve Transaction Image	3-104
API - Retrieve Transaction Acknowledgment Image	3-105
API - Retrieve FA Image	3-105
Communication Services	3-105
Trading Partner Profile Data Block	3-107
Common CMCB Output Fields	3-108
Return Codes from COMM	3-109
API - Send Transactions and Restart Send Transactions	3-109
CMCB Initialization	3-110
Send Network Operation	3-112
Send Transactions Return Information	3-113
Default Message User Class	3-114
Default Message Name	3-114
API - Send Files	3-115
CMCB Initialization	3-115
Send Files Return Information	3-116
API - Receive and Restart Receive	3-117
CMCB Initialization	3-118
Receive Network Operation	3-119
Receive Return Information	3-119
API - Cancel	3-121
CMCB Initialization	3-122
Cancel Return Information	3-122
API - Return Filename	3-123
CMCB Initialization	3-124
Return Filename Return Information	3-124
Internal Calls	3-124
API - Queue Standard Data	3-124
API - Process Network Acknowledgments	3-125
CMCB Initialization	3-125
Status Update Services	3-126
API - Status Update	3-126
Status Update Return Codes	3-128
Full Envelope Key	3-128
Trading Partner Nickname	3-128
Qualifier and Interchange ID	3-129
Account Number and User ID	3-129
Transaction Handle	3-129
Alternate Key	3-130
Alternate Key 1 - Account Number and User ID	3-130
Alternate Key 2 - Interchange Qualifier and ID	3-130
Status Update Data Block	3-131
SYNCPOINT Services	3-132
DB/2 TIMEOUT / DEADLOCK Processing	3-132
API - Initialize SYNC	3-134
Initialize SYNC Return Codes	3-134
COMMIT Work	3-135
ROLLBACK Work	3-135
API - Get Envelope Service	3-136
API - Put Envelope Service	3-136

Chapter 3. Using the DataInterchange Application Program Interface

This chapter describes the DataInterchange services that can be requested directly from an application program and defines the application program interface (API) that should be used to request each of those services. The following are the types of services for which an API has been defined:

- Environmental services
- Translation services
- Enveloping services
- Communication services
- Data extraction services
- Status update services
- SYNCPOINT services

The API for each of these services is described in detail later in this chapter. However, the information common to all the APIs is described next.

The following control blocks are common to all service requests:

- Common control block (CCB)

DataInterchange uses the common control block to maintain status across all services requested by the application. The other primary function of the common control block is to let the application know if the service just requested was successful. The return code and extended return code fields in the CCB indicate the degree of success. The CCB is initialized by the initialization function of environmental services. After initialization, the CCB should not be altered by the application program. One CCB must be used on all service requests. See “Common Control Block (CCB)” on page A-2 for a complete description of the CCB.

- Service name block (SNB)

The service name block indicates the service being requested. As each of the APIs are described, the logical name associated with the API is provided. This logical name should be placed into the service name block before a request is made. The service name block is also used to indicate the number of parameters provided to the service. An application using the API should have an SNB for each service requested. See “Service Name Block (SNB)” on page A-1 for a complete description of the SNB.

- Function control block (FCB)

The function control block indicates the particular function within a service that is being requested. As each of the APIs are described, the function value that needs to be placed in this block is provided. See “Function Control Block (FCB)” on page A-4 for a complete description of the FCB.

API Languages

Access to DataInterchange services is accomplished using a simple call statement to a DataInterchange-provided stub program. DataInterchange provides four stub programs, one for each language directly supported by DataInterchange as an application programming language. These stub programs and associated languages are:

- FXXZCBL for COBOL programs
- FXXZPLI for PL/I programs

- FXXZC for IBM C/370 programs (MVS only)
- FXXZASM for assembler programs

Although DataInterchange directly supports only these languages, any language that can create an operation-system (OS)-style parameter list and uses OS linkage and register conventions can request DataInterchange services by using the FXXZASM stub program.

API Link Edit

The load library distributed with DataInterchange contains a load module for each of the stub programs already described. These programs are linked with the application program requesting the services. These stub programs are small pieces of code that load and transfer control to DataInterchange when a request is made. DataInterchange is not physically part of the application load module. You use the stub programs to access DataInterchange.

When linking an application program, the DataInterchange distribution load library should be part of the //SYSLIB concatenation so the linkage editor can resolve references to the stub programs. An alternative to //SYSLIB concatenation is a separate DD statement for the DataInterchange load library, and a specific INCLUDE statement within the linkage editor control cards to pull in the language stub that is used.

There are no special residency or addressing requirements for an application program requesting DataInterchange services. The application program can be above or below the 16M line, and all the parameters passed to DataInterchange through the stub program can be above or below the 16M line. Figure 3-1 on page 3-4 shows the relationship between the application load module, the stub programs, and DataInterchange.

DataInterchange for MVS and DB2 Attachment

For the most part, the DB2 processing mode that DataInterchange uses is determined by the existence of a file named EDITSIN. If EDITSIN does not exist, the DB2 processing mode will be DSN. Otherwise, the DB2 processing mode will be CAF. There is one exception to this rule. If EDITSIN does exist and the EDITSIN OPEN keyword value is not N and the EDITSIN CAF keyword value is not Y and DataInterchange detects while trying to make the DB2 attachment that DB2 is already attached, then the DB2 processing mode becomes DSN. This exception is true of the DataInterchange facility CLIST.

If EDITSIN does not exist, the DB2 processing mode is DSN and DataInterchange does not attempt to attach or detach DB2 (an attachment is assumed to have taken place prior to DataInterchange initialization). If EDITSIN does exist, the DB2 processing mode is CAF. If OPEN is not N and if values are supplied for SYSTEM and PLAN, DataInterchange initialization will attempt to attach DB2. If it is found that DB2 is already attached and CAF is not Y, the DB2 processing mode becomes DSN. If CLOSE is not N and if values are supplied for SYSTEM and PLAN, DataInterchange termination will detach DB2.

Table 3-1 (Page 1 of 2). EDITSIN Keywords

Keyword	Description
SYSTEM	The name of the DB2 subsystem (or group, if Data Sharing). This value may be up to four characters long and is required if EDITSIN exists.
PLAN	The DB2 plan name. This value may be up to eight characters long and is required if EDITSIN exists.
OPEN	A value of N tells DataInterchange not to try to attach to DB2 during DataInterchange initialization. The default is Y.

Table 3-1 (Page 2 of 2). EDITSIN Keywords

Keyword	Description
CLOSE	A value of N tells DataInterchange not to detach from DB2 during DataInterchange termination. The default is Y.
CAF	If DataInterchange initialization detects while trying to make a DB2 attachment that DB2 is already attached, the DataInterchange DB2 processing mode becomes DSN. A value of Y for this keyword acts as an override and tells DataInterchange to keep the DB2 processing mode CAF. The default is N.

EDITSIN Examples

Example 1: If your DB2 subsystem is named DB93 and your DB2 plan is named EDIENU31 and if you would like DataInterchange to handle the DB2 attachment and detachment, you would include the following keywords and values in EDITSIN:

```
SYSTEM(DB93) PLAN(EDIENU31)
```

Example 2: If your DB2 subsystem is named DB93 and your DB2 plan is named EDIENU31 and if DataInterchange should not attach or detach DB2, you would include the following keywords and values in EDITSIN:

```
SYSTEM(DB93) PLAN(EDIENU31) OPEN(N) CLOSE(N)
```

Example 3: If your DB2 subsystem is named DB93 and your DB2 plan is named EDIENU31 and DB2 may or may not be attached, but either way you do not want DataInterchange to detach upon termination, and you want DataInterchange to process in CAF mode regardless of whether DB2 was previously attached or not, you would include the following keywords and values in EDITSIN:

```
SYSTEM(DB93) PLAN(EDIENU31) CLOSE(N) CAF(Y)
```

DataInterchange for CICS Abend Return Codes

When an abend is detected, DataInterchange returns -8 in ZCCBRC and the EBCDIC representation of the CICS abend code is ZCCBERC. This return code combination is global and is not mentioned in upcoming tables.

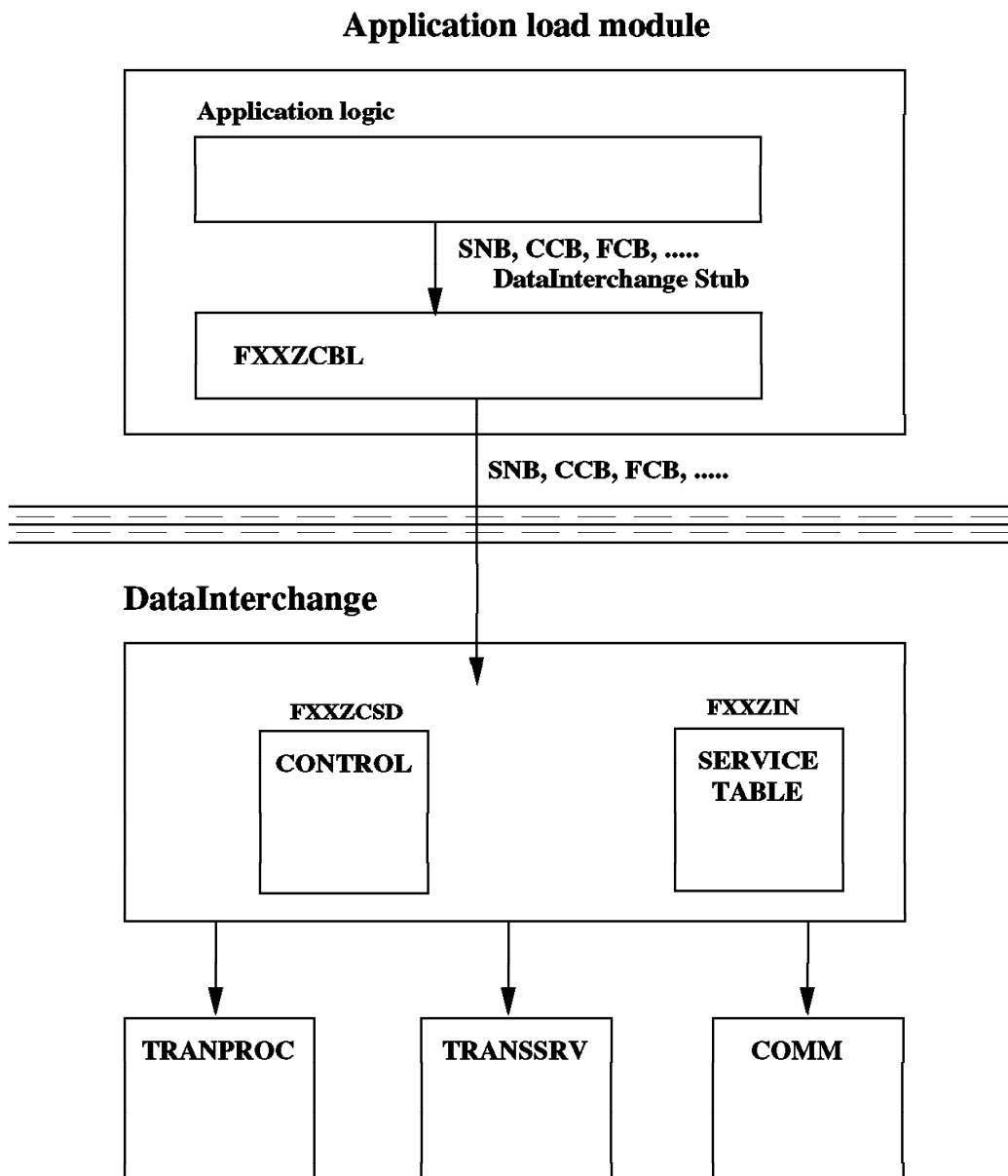


Figure 3-1. Load Module Relationships

Calls Using COBOL

When using the COBOL language, you invoke DataInterchange as a normal call to an external function. The call is made to the **FXXZCBL** routine and is coded as follows:

```
CALL FXXZCBL USING SNB CCB FCB [other parms]
```

DataInterchange assumes that all the parameters are pointers to control blocks. This is the normal way COBOL passes parameters. A code fragment to initialize DataInterchange might look like the following:

SNB-COBOL

```
| 01 SNB-DATA.  
|     03 ZSNBLL          PIC S9(4) COMP-4.  
|     03 ZSNBID          PIC S9(4) COMP-4.  
|     03 ZSNBEYE         PIC X(8).  
|     03 ZSNBNAME        PIC X(8).  
|     03 ZSNBNDX         PIC S9(9) COMP-4.  
|     03 ZSNBPC          PIC S9(4) COMP-4.  
|     03 ZSNBFLG0        PIC X(1).  
|     03 ZSNBFLG1        PIC X(1).  
|     03 ZSNBFANC        PIC S9(9) COMP-4.
```

CCB-COBOL

```
| 01 CCB-DATA.  
|     03 ZCCBLL          PIC S9(4) COMP-4.  
|     03 ZCCBID          PIC S9(4) COMP-4.  
|     03 ZCCBEYE         PIC X(8).  
|     03 ZCCBRC          PIC S9(9) COMP-4.  
|     03 ZCCBERC         PIC S9(9) COMP-4.  
|     03 ZCCBSID         PIC X(8).  
|     03 ZCCBUID         PIC X(8).  
|     03 ZCCBAID         PIC X(8).  
|     03 ZCCBCID         PIC X(8).  
|     03 ZCCBXFID        PIC S9(4) COMP-4.  
|     03 ZCCBLPID        PIC X(6).  
|     03 ZCCBCPID        PIC S9(9) COMP-4.  
|     03 ZCCBRSV         PIC S9(9) COMP-4.  
|     03 ZCCBCCXP        PIC S9(9) COMP-4.  
|     03 ZCCBCABP        PIC S9(9) COMP-4.  
|     03 ZCCBDBID        PIC X(4).  
|     03 ZCCBDBPL        PIC X(8).  
|     03 ZCCBDBUI        PIC X(8).  
|     03 ZCCBDBPW        PIC X(18).  
|     03 ZCCBDSV1        PIC X(26).  
|     03 ZCCBRV2         PIC S9(9) COMP-4 OCCURS 117 TIMES.
```

FCB-COBOL

```
| 01 FCB-DATA.  
|     03 ZFCBLL          PIC S9(4) COMP-4.  
|     03 ZFCBFUNC        PIC S9(4) COMP-4.
```

INIT-COBOL

```
01  APPLID          PIC X(08).  
01  SYSID           PIC X(08).  
  
| MOVE LOW-VALUES TO SNB-DATA CCB-DATA FCB-DATA.  
| MOVE 'ENVSERV ' TO ZSNBNAME.  
| MOVE 5           TO ZSNBPC.  
| MOVE 'ENU '      TO ZCCBLPID.  
| MOVE 1           TO ZFCBFUNC.  
| MOVE 'MYAPPL '   TO APPLID.  
| MOVE 'SYSTEM '   TO SYSID.  
| CALL FXXZCBL USING SNB-DATA CCB-DATA FCB-DATA APPLID SYSID.  
| IF ZCCBRC EQUAL ZERO  
| * initialization worked  
| ELSE  
| * initialization failed  
| END-IF.
```

Calls Using PL/I

When using the PL/I language, you invoke DataInterchange as a normal call to an external function. The call is made to the **FXXZPLI** routine and is coded as follows:

```
CALL FXXZPLI(SNB,CCB,FCB,...other parms...)
```

DataInterchange assumes that all parameters are pointers to control blocks. This is not what PL/I would normally pass on to an external routine. PL/I, unless told otherwise, assumes that the call is being made to another program written in PL/I and passes special PL/I parameter information along with the parameters. Because DataInterchange does not understand this information, the compiler must be told that FXXZPLI is an assembler program with the following statement:

```
DCL FXXZPLI EXTERNAL ENTRY OPTIONS (ASSEMBLER,INTER)
```

A code fragment to initialize DataInterchange might look like the following:

SNB-PL/I

```
| DCL 1 SNB,  
|     3 ZSNBLL      FIXED BIN(15), /* SNB block length      */  
|     3 ZSNBID      FIXED BIN(15), /* Reserved for future use */  
|     3 ZSNBEYE     CHAR(8),       /* SNB block name        */  
|     3 ZSNBNAME    CHAR(8),       /* Service name          */  
|     3 ZSNBNDX     FIXED BIN(31), /* Service index         */  
|     3 ZSNBPC      FIXED BIN(15), /* Service parameter count */  
|     3 ZSNBFLG0    CHAR(1),       /* First flag byte       */  
|     3 ZSNBFLG1    CHAR(1),       /* Second flag byte      */  
|     3 ZSNBFANC    POINTER;        /* First anchor pointer   */
```

CCB-PL/I

```
| DCL 1 CCB,  
|     3 ZCCBLL      FIXED BIN(15), /* CCB block length      */  
|     3 ZCCBID      FIXED BIN(15), /* Reserved for future use */  
|     3 ZCCBEYE     CHAR(8),        /* CCB block name        */  
|     3 ZCCBRC      FIXED BIN(31), /* Return code           */  
|     3 ZCCBERC     FIXED BIN(31), /* Extended return code  */  
|     3 ZCCBSID     CHAR(8),        /* System ID             */  
|     3 ZCCBUID     CHAR(8),        /* User ID               */  
|     3 ZCCBAID     CHAR(8),        /* Application ID        */  
|     3 ZCCBCID     CHAR(8),        /* Error module ID       */  
|     3 ZCCBXFID    FIXED BIN(15), /* Component function ID */  
|     3 ZCCBLPID    CHAR(6),        /* Language profile ID   */  
|     3 ZCCBCPID    FIXED BIN(31), /* Code page ID          */  
|     3 ZCCBRSV     FIXED BIN(31), /* Reserved for future use */  
|     3 ZCCBCCXP    POINTER,        /* CCB extension pointer */  
|     3 ZCCBCABP    POINTER,        /* Common area block pointer */  
|     3 ZCCBDBID    CHAR(4),        /* MVS DB2 subsystem ID  */  
|     3 ZCCBDBPL    CHAR(8),        /* MVS DB2 plan / AIX alias */  
|     3 ZCCBDBUI    CHAR(8),        /* AIX DB2 user ID       */  
|     3 ZCCBDBPW    CHAR(18),       /* AIX DB2 password      */  
|     3 ZCCBRSV1    CHAR(26),       /* DI internal use only   */  
|     3 ZCCBRSV2(117) FIXED BIN(31); /* DI internal use only   */
```

FCB-PL/I

```
| DCL 1 FCB,  
|     3 ZFCBLL      FIXED BIN(15), /* FCB block length      */  
|     3 ZFCBFUNC    FIXED BIN(15); /* Service function code  */
```

INIT-PL/I

```
DCL APPLID CHAR(8) INIT('MYAPPL ');  
DCL SYSID CHAR(8) INIT('SYSTEM ');  
  
SNB = ";CCB = "; FCB = "  
SNB.ZSNBNAME = 'ENVSERV '  
SNB.ZSNBPC = 5;  
CCB.ZCCBLPID = 'ENU '  
FCB.ZFCBFUNC = 1;  
CALL FXXZPLI(SNB,CCB,FCB,APPLID,SYSID);  
IF CCB.ZCCBRC = 0 THEN DO;  
/* initialization worked */  
END;  
ELSE DO;  
/* initialization failed */  
END;
```

Calls Using C

The **fxzxc** stub program can only be used with the IBM C/370 compiler and is only supported in the MVS environment. When using the C language, you invoke DataInterchange as a normal function call. The function invoked is **fxzxc** and is coded as follows:

```
fxzxc(&mysnb,&myccb,&myfcb, ...other parms....)
```

DataInterchange assumes that all parameters are pointers to control blocks. This is why the **fxzxc** call uses the control block address rather than a control block reference (&myccb versus myccb). A code fragment to initialize DataInterchange might look like the following:

SNB-C

```
| typedef struct SNB snb;          /* Provide "snb" data type */
| struct SNB {
|     short zsnbl1;               /* SNB block length      */
|     short zsnbid;               /* Reserved for future use */
|     char  zsnbeye[8];           /* SNB block name        */
|     char  zsnbname[8];          /* Service name           */
|     long  zsnbndx;               /* Service index          */
|     short zsnbpc;               /* Service parameter count */
|     char  zsnbflg0;             /* First flag byte        */
|     char  zsnbflg1;             /* Second flag byte       */
|     void  *zsnbfanc;            /* First anchor pointer   */
| };
```

CCB-C

```
| typedef struct CCB ccb;          /* Provide "ccb" data type */
| struct CCB {
|     short zccb11;               /* CCB block length      */
|     short zccb1d;               /* Reserved for future use */
|     char  zccbeye[8];           /* CCB block name        */
|     long  zccbrc;               /* Return code            */
|     long  zccberc;              /* Extended return code   */
|     char  zccbsid[8];           /* System ID              */
|     char  zccbuid[8];           /* User ID                 */
|     char  zccbaid[8];           /* Application ID          */
|     char  zccbcid[8];           /* Error module ID        */
|     short zccbxfid;             /* Component function ID   */
|     char  zccb1pid[6];          /* Language profile ID     */
|     long  zccbcpid;             /* Code page ID            */
|     long  zccbrsv;              /* Reserved for future use */
|     void  *zccbccxp;            /* CCB extension pointer   */
|     void  *zccbcabp;            /* Common area block pointer */
|     char  zccbdbid[4];          /* MVS DB2 subsystem ID   */
|     char  zccbdbpl[8];          /* MVS DB2 plan / AIX alias */
|     char  zccbdbui[8];          /* AIX DB2 user ID        */
|     char  zccbdbpw[18];         /* AIX DB2 password       */
|     char  zccbrsv1[26];         /* DI internal use only    */
|     long  zccbrsv2[117];        /* DI internal use only    */
| };
```

FCB-C

```
| typedef struct FCB fcb;           /* Provide "fcb" data type */
| struct FCB {
|     short zfcbl1;                /* FCB block length */
|     short zfcbfunc;              /* Service function code */
| };
```

INIT-C

```
| snb mysnb;
| ccb myccb;
| fcb myfcb;
| memset(&mysnb,'\0',sizeof(mysnb));
| memset(&myccb,'\0',sizeof(myccb));
| memset(&myfcb,'\0',sizeof(myfcb));
| memcpy(mysnb.zsnbname,"ENVSERV",8); /* service wanted */
| mysnb.zsnbpc = 5; /* parms on fxxzc call */
| memcpy(myccb.zccb1pid,"ENU ",6); /* language to be used */
| myfcb.zfcbfunc = 1; /* INITIALIZE function */
| fxxzc(&mysnb,&myccb,&myfcb,"MYAPPL ","SYSTEM ");
| if (!myccb.zccbrc) {
| /* initialization worked */
| } else {
| /* initialization failed */
| }
```

Calls Using Assembler

When using the assembler language, you invoke DataInterchange as a normal call to an external function. The call is made to **FXXZASM** using standard OS parameter lists, register conventions, and linkage conventions. The register conventions are:

Register	Description
R0	Not defined
R1	Address of a parameter list
R2	Not defined
R3	Not defined
R4	Not defined
R5	Not defined
R6	Not defined
R7	Not defined
R8	Not defined
R9	Not defined
R10	Not defined
R11	Not defined
R12	Not defined
R13	Address of 72-byte area used by called program to save the registers of the calling program
R14	Return address to program making the call
R15	Entry point for routine being called

The parameter list pointed to by register 1 consists of a series of pointers to the parameters, which are coded as follows:

```

R1-----> +0: Address of the SNB
            +4: Address of the CCB
            +8: Address of the FCB
            +C: Address of service parm 1
            +10: Address of service parm 2
            etc. for as many parms as the service calls for

```

A code fragment to initialize DataInterchange might look like the following:

SNB-Assembler

```

| SNB      DSECT ,
|          DS    0D
| ZSNBLL   DS    H           SNB block length
| ZSNBID   DS    H           Reserved for future use
| ZSNBEYE  DS    CL8         SNB block name
| ZSNBNAME DS    CL8         Service name
| ZSNBNDX  DS    F           Service index
| ZSNBPC   DS    H           Service parameter count
| ZSNBFLG0 DS    CL1         First flag byte
| ZSNBFLG1 DS    CL1         Second flag byte
| ZSNBFANC DS    F           First anchor pointer
| ZSNBLEN  EQU   *-ZSNBLL    Length of SNB

```

CCB-Assembler

```

| CCB      DSECT ,
|          DS    0D
| ZCCBLL   DS    H           CCB block length
| ZCCBID   DS    H           Reserved for future use
| ZCCBEYE  DS    CL8         CCB block name
| ZCCBRC   DS    F           Return code
| ZCCBERC  DS    F           Extended return code
| ZCCBSID  DS    CL8         System ID
| ZCCBUID  DS    CL8         User ID
| ZCCBAID  DS    CL8         Application ID
| ZCCBCID  DS    CL8         Error module ID
| ZCCBXFID DS    H           Component function ID
| ZCCBLPID DS    CL6         Language profile ID
| ZCCBCPID DS    F           Code page ID
| ZCCBRSV  DS    F           Reserved for future use
| ZCCBCCXP DS    F           CCB extension pointer
| ZCCBCABP DS    F           Common area block pointer
| ZCCBDBID DS    CL4         MVS DB2 subsystem ID
| ZCCBDBPL DS    CL8         MVS DB2 plan / AIX alias
| ZCCBDBUI DS    CL8         AIX DB2 user ID
| ZCCBDBPW DS    CL18        AIX DB2 password
| ZCCBRSV1 DS    CL26        DI internal use only
| ZCCBRSV2 DS    117F        DI internal use only
| ZCCBLEN  EQU   *-ZCCBLL    Length of CCB

```


FCB-Assembler

```
| FCB      DSECT ,
|          DS    0D
| ZFCBLL   DS    H          FCB block length
| ZFCBFUNC DS    H          Service function code
| ZFCBLEN  EQU   *-ZFCBLL   Length of FCB
```

USER-DSECT for Initialization Sample

```
USERDS    DSECT ,
SAVAREA   DS    18F
SNBADDR   DS     A
CCBADDR   DS     A
FCBADDR   DS     A
APPADDR   DS     A
SYSADDR   DS     A
SNBAREA   DS    CL(SNBLEN)
CCBAREA   DS    CL(CCBLEN)
FCBAREA   DS    CL(FCBLEN)
```

INIT-Assembler

```
*****
* It is assumed storage for the USERDS DSECT      *
* has been allocated (initialized with binary zeros) *
* and is addressable.                             *
*****
```

```
APPLID    DC      CL8'MYAPPL '
SYSID     DC      CL8'SYSTEM '
LA        13,SAVAREA
LA        6,SNBAREA
LA        7,CCBAREA
LA        8,FCBAREA
USING     SNB,6
USING     CCB,7
USING     FCB,8
MVC       ZSNBNAME(8),=CL8'ENVSERV '
MVC       ZSNBPC(2),=X'0005'
MVC       ZCCBLPID(6),=CL6'ENU '
MVC       ZFCBFUNC(2),=X'0001'
ST        6,SNBADDR
ST        7,CCBADDR
ST        8,FCBADDR
LA        9,APPLID
ST        9,APPADDR
LA        9,SYSID
ST        9,SYSADDR
LA        1,SNBADDR
L         15,=V(FXXZASM)
BALR      14,15
ICM       9,15,ZCCBRC
BNZ       INITERR
```

```
*****
*   DataInterchange initialization successful   *
*****
```

```
INITERR EQU *
```

```
*****
*   DataInterchange initialization unsuccessful *
*****
```

API Business Tasks

There is no individual API service that performs a business function. It requires a combination of services called in a specific order to accomplish the task. For example, if you want to write a program that reads an application file, translates the file into a standard format, and sends the data to all trading partners, the following sequence of API service calls is required:

1. Environmental service to initialize DataInterchange (see “API - Initialization” on page 3-17)
2. Repeated calls to the translation service to translate data and envelope at the same time (see “API - Translate to Standard” on page 3-27)
3. Translation service to signal the end of translation (see “API - End Translation/Enveloping” on page 3-85)
4. Communication service to send the transactions (see “API - Send Transactions and Restart Send Transactions” on page 3-109)
5. Environmental service to terminate DataInterchange (see “API - Termination” on page 3-19)

All of these calls can be accomplished using the DataInterchange Utility with a `PERFORM TRANSLATE AND SEND` command. The utility issues the same API requests that you would issue within an application program.

The decision to use the API or a DataInterchange Utility request (`PERFORM`) should be based on control and availability of data rather than the availability of DataInterchange Utility functions. The DataInterchange Utility provides most of the functions available in the API. However, the DataInterchange Utility requires that the data be in specific formats; for example, C and D records or raw data accessed using specific access methods, such as QSAM in MVS. This method only provides overall result status (JCL condition codes). The following are a few reasons you might consider writing an API program:

- You have more direct control when writing an API program of your own. At each step, the system provides detailed information about the status of each API request. The DataInterchange Utility provides a condition code indicating the most severe error and an audit trail of all errors that occurred. If you want to automatically process these errors, a program to parse the AUDIT file could become complex. You should consider writing a program to request API services directly.
- You might want to synchronize the updating of your data with the results of DataInterchange processing. For example, if you are translating, you might want to update your application record to indicate whether the translation was successful, and you want the change of your data to be synchronized with the DataInterchange databases to verify that the business transaction has been processed.

Note: In CICS, it is possible to synchronize application data with DataInterchange data without the need to write an API program. For more information, see Chapter 5, “Using DataInterchange in the CICS Environment” on page 5-1.

- The application data might not meet the RAWDATA requirements of DataInterchange or be formatted in C and D records. As a result, only your application understands the data. When you understand the data, you can tell DataInterchange about it and perform translation.
- The application data might not meet the access type requirements of DataInterchange (QSAM in MVS; VSAM ESDS, temporary storage queue or transient data queue in CICS). For example, if the application data is stored in DB2 tables, the DataInterchange utilities cannot be used without first moving the data to one of the storage mechanisms mentioned above.
- The sequencing of transactions in interchanges and groups that is performed by the DataInterchange Utility might not meet your requirements.

As the API functions are described in the following sections, the equivalent PERFORM commands are given. After reading the descriptions of the PERFORM commands, determine whether an API program is required or if you need the DataInterchange Utility. For more information, see Chapter 1, “Using the DataInterchange Utility.”

You might need a combination of application-written API programs and DataInterchange Utility services. For example, you might want control of the translation process. However, control might not be necessary for enveloping and sending the data. In this situation, your business process might be as follows:

1. An API program that issues the following API requests:
 - a. Environmental service to initialize DataInterchange
 - b. Translation service to translate data without enveloping
 - c. Translation service to signal the end of translation
 - d. Environmental service to terminate DataInterchange
2. A DataInterchange Utility step to use the PERFORM ENVELOPE AND SEND command, or a two-step DataInterchange Utility process to use the PERFORM ENVELOPE command followed by the PERFORM SEND command.

API Function Overview

This section contains an overview of the functions provided by each API service with a CALL example that shows the parameters expected by each of the functions. This overview is followed by detailed descriptions of each service and each function within the service. Table 3-2 contains the abbreviations for control blocks used for the remainder of this publication.

Table 3-2 (Page 1 of 2). Control Block Abbreviations

Control Block	Description	Page
CCB	Common control block	A-2
CMCB	Communications control block	A-34
DATABLK	Communications data block	A-52
FCB	Function control block	A-4
NPDB	Network profile data block	A-53
REQDB	Requestor profile data block	A-56
SNB	Service name block	A-1
TPPDB	Trading partner profile data block	A-42
TRCB	Translator control block	A-6
TRIDB	Translator input data block	A-29
TRODB	Translator output data block	A-31

Table 3-2 (Page 2 of 2). Control Block Abbreviations

Control Block	Description	Page
USDB	Update status data block	3-131
USKB	Update status key block	3-126
UTILCB	Utility control block	5-20

Environmental Services Overview

Table 3-3 is an overview of the functions provided by the environmental services. The logical name for the environmental service is **ENVSERV**. This service is described in “Environmental Services” on page 3-17.

Table 3-3. Overview of Environmental Services

Function Code Value	Function Provided	Sample Call Statement for Function	Page
1	DataInterchange initialization	FXXZccc(SNB,CCB,FCB,'applid','sysid')	3-17
2	DataInterchange termination	FXXZccc(SNB,CCB,FCB)	3-19

Translation Services Overview

Table 3-4 is overview of the functions provided by the translation services. The logical name for the translation service is **TRANPROC**. This service is described in “Translation Services” on page 3-20.

Table 3-4. Overview of Translation Services

Function Code Value	Function Provided	Sample Call Statement for Function	Page
111	Test translate to standard	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-27
131	Production translate to standard	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-27
211	Test deenvelope and translate to application	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-57
212	Production deenvelope and translate to application	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-57
213	Translate a specific transaction to application	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-49
1000	End translation	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-85

Enveloping Services Overview

Table 3-5 on page 3-15 is an overview of the functions provided by the enveloping services. The logical name for the enveloping service is **TRANPROC**. This service is described in “Enveloping Services” on page 3-69.

Table 3-5. Overview of Enveloping Services

Function Code Value	Function Provided	Sample Call Statement for Function	Page
1	Retrieve interchange header	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-99
2	Retrieve group header	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-100
3	Retrieve transaction header	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-100
214	Deenvelope transactions	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-86
215	Envelope transaction	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-72
990	Close and QUEUE the current interchange	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-84
991	Issue database COMMIT	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-98

Extraction Services Overview

Table 3-6 is an overview of the functions provided by the data extraction services. The logical name for the data extraction service is **TRANPROC**. This service is described in “Data Extraction Services” on page 3-101.

Table 3-6. Overview of Data Extraction Services

Function Code Value	Function Provided	Sample Call Statement for Function	Page
216	Retrieve detailed transaction data	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-102
217	Retrieve transaction image	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-104
218	Retrieve functional acknowledgment image	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-105
219	Retrieve transaction acknowledgment image	FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)	3-105

Communications Services Overview

Table 3-7 is an overview of the functions provided by the communications services. The logical name for the communications service is **COMMbbb**. This service is described in “Communication Services” on page 3-105.

Table 3-7 (Page 1 of 2). Overview of Communications Services

Function Code Value	Function Provided	Sample Call Statement for Function	Page
110	Queue standard data	FXXZccc(SNB,CCB,FCB,CMCB,TPPDB,DATABLK)	3-124
211	Send transactions	FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)	3-109
221	Send files	FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)	3-115
232	Receive	FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)	3-117
233	Cancel previously sent data	FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)	3-121
300	Return file name	FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)	3-123

Table 3-7 (Page 2 of 2). Overview of Communications Services

Function Code Value	Function Provided	Sample Call Statement for Function	Page
252	Process network acknowledgments	FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)	3-125

Status Update Services Overview

Table 3-8 is an overview of the functions provided by the status update services. The logical name for the update status service is **TRANSSRV**. A description of this service is in “Status Update Services” on page 3-126.

Table 3-8. Overview of Status Update Services

Function Code Value	Function Provided	Sample Call Statement for Function	Page
210	Update status using the envelope full key with trading partner identified by the account number and user ID.	FXXZC(SNB,CCB,FCB,USKB210,status[,USDB[,REQID]])	3-129
211	Update status using the transaction handle value for one of the transactions within the interchange.	FXXZC(SNB,CCB,FCB,USKB211,status[,USDB[,REQID]])	3-129
212	Update status using an alternate key with the trading partner identified by the account number and user ID.	FXXZC(SNB,CCB,FCB,USKB212,status[,USDB[,REQID]])	3-130
213	Update status using the full envelope key with the trading partner identified by the interchange qualifier and interchange ID.	FXXZC(SNB,CCB,FCB,USKB213,status[,USDB[,REQID]])	3-129
214	Update status using an alternate key with the trading partner identified by the interchange qualifier and interchange ID.	FXXZC(SNB,CCB,FCB,USKB214,status[,USDB[,REQID]])	3-130
215	Update status using the full envelope key with the trading partner identified by the trading partner nickname.	FXXZC(SNB,CCB,FCB,USKB215,status[,USDB[,REQID]])	3-128

SYNCPOINT Services Overview

Table 3-9 on page 3-17 is an overview of the functions provided by the SYNCPOINT services. The logical name for the SYNCPOINT service is **SYNCSERV**. A description of this service is in “SYNCPOINT Services” on page 3-132.

Table 3-9. Overview of SYNCPOINT Services

Function Code Value	Function Provided	Sample Call Statement for Function	Page
1	Initialize SYNCPOINT services	FXXZccc(SNB,CCB,FCB,syncval)	3-134
2	COMMIT work	FXXZccc(SNB,CCB,FCB)	3-135
3	ROLLBACK work	FXXZccc(SNB,CCB,FCB)	3-135

Environmental Services

The environmental services establish the DataInterchange environment and remove the DataInterchange environment. Your application program should first make an initialization function request. After the initialization is complete, other services can be requested. If another service is requested before an initialization is requested, a return code of -1 is posted in the common control block or the program will ABEND.

The final function your application program should request is a termination function request. The services requested between initialization and termination (such as translation or communication services) internally acquire storage, open files, and obtain control over resources. The termination function is necessary to enable DataInterchange to release all resources that have been obtained.

API - Initialization

You must request the initialization function to enable your program to use the DataInterchange environment. During request processing, DataInterchange confirms that all necessary resources for a DataInterchange environment are available, and that you are authorized to access the system (identified by the **sysid** parameter). DataInterchange also verifies that the language profile ID (saved in the **ZCCBLPID** field of the common control block before the call) identifies a member in the language profile (LANGPROF).

The **applid** parameter should identify a member in the application definition (APPDEFS) profile or a member in the activity log (ACTLOGS) profile. If an APPDEFS member does not exist, the following assumptions are made:

- The **applid** parameter identifies an ACTLOGS member (if it does not and logging is necessary for other service requests, errors occur).
- The management reporting functions should be enabled for the current DataInterchange session.

The following is the syntax for the function request:

```
FXXZccc(SNB,CCB,FCB,applid,sysid)
```

The parameters for this function request are:

Parameter	Description						
SNB	Is the service name block identifying environmental services.						
	<table> <tr> <th>Field</th><th>Value</th></tr> <tr> <td>ZSNBNAME</td><td>ENVSERVb to identify environmental services</td></tr> <tr> <td>ZSNBPC</td><td>5 (number of parameters in the call)</td></tr> </table>	Field	Value	ZSNBNAME	ENVSERVb to identify environmental services	ZSNBPC	5 (number of parameters in the call)
Field	Value						
ZSNBNAME	ENVSERVb to identify environmental services						
ZSNBPC	5 (number of parameters in the call)						

CCB Is the common control block.

Field	Value
ZCCBLPID	The language profile member that should be used for this DataInterchange session.

FCB Is the function control block with a **ZFCBFUNC** value of 1.

applid Is the application ID that should match either a member in the application definition profile or the activity log profile. The application ID should be 8 bytes in length, left-justified, and padded with blanks.

sysid Is the system ID that must match a resource name the security administrator assigned to DataInterchange. The system ID must be 8 bytes in length, left-justified, and padded with blanks. The system ID is an optional parameter (if not provided, the **ZSNBPC** field should be 4) that has a default value of DIENU.

Note: The sysid parameter does not apply in a CICS environment.

The results of the initialization request are posted in the return code and extended return code fields of the common control block. The following values are possible:

RC	Explanation
----	-------------

0	Initialization was successful.
---	--------------------------------

4	Initialization failed. Extended return codes are:
---	---

ERC	Explanation
-----	-------------

4	No service table defined (failure to locate FXXZIN load module).
---	--

8	Insufficient virtual storage to load programs and tables.
---	---

12	Failure initializing the EDIT service. This may be caused by an incorrect ZCCBLPID value (one that cannot be matched with a LANGPROF member). In VSAM installations, this error may occur because of the inability to access PROFDEF, PROFDAT, EDITPXRF, TABLDEF, or TABLDAT VSAM files. In DB2 installations, this error may occur because of the inability to access the profile or translation/validation tables (in many cases, this could be a DB2 timestamp error, meaning that the timestamp in load module EDIPSMD is different from its corresponding DBRM timestamp). If a DB2 timestamp error is suspected (SQL code -818), the DB2 plan must be rebound. Besides EDIPSMD, there are two other DataInterchange DB2 load modules: EDIRPML and (for CICS users) EDICRIN, which could generate DB2 timestamp problems if out of sync.
----	--

16	A DataInterchange session for the same user is already active.
----	--

1024	Access to the system is denied.
------	---------------------------------

API - Utility Services

The utility service application program interface (API) is used when implementing HOT-DI.

The utility service API is:

FXXZccc(SNB,CCB,FCB,UTILCB)

The parameters for this function request are:

Parameter	Description						
SNB	Is the service name block identifying the utility service.						
	<table> <tr> <th>Field</th><th>Value</th></tr> <tr> <td>ZSNBNAME</td><td>UTILSRV</td></tr> <tr> <td>ZSNBPC</td><td>4 (number of parameters in the call)</td></tr> </table>	Field	Value	ZSNBNAME	UTILSRV	ZSNBPC	4 (number of parameters in the call)
Field	Value						
ZSNBNAME	UTILSRV						
ZSNBPC	4 (number of parameters in the call)						
CCB	Is the common control block that was used to initialize DataInterchange.						
FCB	Is the function control block						
	<table> <tr> <th>Field</th><th>Value</th></tr> <tr> <td>ZFCBFUNC</td><td>1 - Normal execution 2 - HOT DataInterchange mode</td></tr> </table>	Field	Value	ZFCBFUNC	1 - Normal execution 2 - HOT DataInterchange mode		
Field	Value						
ZFCBFUNC	1 - Normal execution 2 - HOT DataInterchange mode						
UTILCB	The utility service control block.						

API - Termination

You must request the termination function for DataInterchange to perform housekeeping activities for cleanup of all system resources acquired when processing the API service requests. Now, DataInterchange performs such functions as releasing virtual storage, closing files, and releasing locks on resources.

If an error occurs during termination, DataInterchange returns a value other than zero in the return code field (**ZCCBRC**). If this occurs, your application program should request the termination function again and continue to do this until the function completes successfully. A failure in termination indicates that a file could not be closed, or storage could not be released. When this happens, the DataInterchange component for which the error occurred attempts to log the error. After the error occurs, DataInterchange returns control to your program so that your program can request termination again to complete the task.

DataInterchange does not attempt to call the program where the error occurred again, but continues to call other programs that are used during cleanup. The following is an example of the syntax for the function request:

```
FXXZccc(SNB,CCB,FCB)
```

The parameters for this function request are:

Parameter	Description						
SNB	Is the service name block for environmental services.						
	<table> <tr> <th>Field</th><th>Value</th></tr> <tr> <td>ZSNBNAME</td><td>ENVSRVb to identify environmental services</td></tr> <tr> <td>ZSNBPC</td><td>3 (number of parameters in the call)</td></tr> </table>	Field	Value	ZSNBNAME	ENVSRVb to identify environmental services	ZSNBPC	3 (number of parameters in the call)
Field	Value						
ZSNBNAME	ENVSRVb to identify environmental services						
ZSNBPC	3 (number of parameters in the call)						
CCB	Is the common control block used for the initialization function call.						
FCB	Is the function control block with a ZFCBFUNC value of 2.						

The results of the termination request are posted in the return code and extended return code fields of the common control block. The following values are possible:

RC	Explanation
0	Termination was successful.
>0	Termination failed, try again.
<0	Not a valid common control block address or an incorrect name in the ZSNBNAME field.

Translation Services

Translation services, enveloping services, and data extraction services are closely related. They share the same application program interface (API) implemented by the same logical service (TRANPROC). These services also share the same database component within DataInterchange, called the Transaction Store. It is important to understand the Transaction Store to understand and use the API functions.

A number of section headings in the pages that follow include a two-letter abbreviation within parentheses. These are used to make the section headings unique. The two-character abbreviations you will see and their meanings are:

- TS - Translate to standard
- TA - Translate to application
- TF - Translate file
- EV - Envelope

The exchange of information with a trading partner is a sequence of interruptible events whose status must be maintained and tracked. The processes used to exchange data fall into two categories: a batch-oriented process, characteristic of the MVS environment, and a time-critical process, characteristic of the CICS environment.

In the batch-oriented process, application data is stored within files owned by the applications until it is time for a batch job to run, which will translate the application data into a standard format. This job, or a different job run later, can request that all the standard data should be gathered (enveloped) and sent to the trading partner. The trading partners receive the data, process the data, and return a response.

In the time-critical process, application data is usually translated, enveloped, and sent as soon as the application data is available. Although some batching of data may occur, it is typically a much smaller batch than in the batch-oriented process. The trading partner is expected to process the data and generate a response immediately.

The Transaction Store database saves all standard data sent or received and tracks the status of the data as it progresses from translated, to send, to received, to acknowledged. The Transaction Store is updated by all translation and enveloping operations, and is maintained and reported on by the Transaction Store Facility or the DataInterchange Utility. The updating of the Transaction Store database is optional and may be controlled using the APPDEFS profile. For more information, see the APPDEFS profile in the *DataInterchange Administrator's Guide*. The Transaction Store database consists of seven DB2 tables (or seven VSAM files if the VSAM version of DataInterchange is being used). The following list describes these tables:

Table	Description				
EDIVTSTH	<p>The transaction handle table. This is the primary table within the Transaction Store from which all other tables derive meaning. An entry is created any time a translation occurs between application formats and standard formats and any time a transaction is deenveloped. As transactions are added to the Transaction Store, they are assigned a unique key value called a transaction handle (THANDLE). The format of the transaction handle is:</p> <p>YYYYMMDDHHMMSSxxnnnn</p> <p>where:</p> <table><tr><th>Parameter</th><th>Meaning</th></tr><tr><td>YYYYMMDD</td><td>Is the date of the first transaction for the translation session.</td></tr></table>	Parameter	Meaning	YYYYMMDD	Is the date of the first transaction for the translation session.
Parameter	Meaning				
YYYYMMDD	Is the date of the first transaction for the translation session.				

HHMMSS	Is the time of the first transaction for the translation session.
xx	Is a random number assigned by DataInterchange with a range of values from 00 to 99.
nnnn	Is a sequential number ranging from 0 to 9999. If more than 9999 transactions are translated in the same session, DataInterchange receives a new date, new time, and new random number, and starts the sequence at 0 again.

This key value is communicated between application programs and DataInterchange API services through the **TSKEY** and **TSKEYU** fields in the translator control block (TRCB).

EDIVTSTI	The standard transaction image table. It contains the standard transaction image and is created any time a translation occurs between application formats and standard formats, or any time a deenvelope takes place.
EDIVTSTO	The transaction override table. This table contains override values for fields in the service segments. A transaction override table is created only when translation occurs from application format to standard format, when overrides are supplied for service segment fields. These values are captured at translation time and used when the transaction is enveloped.
EDIVTSEV	This table contains information about an entire interchange, such as the status on the network, the date and time created, and the date and time sent. A table entry is created during enveloping or deenveloping for each interchange header created or deenveloped.
EDIVTSGP	This table contains information about a group of transactions within an interchange, such as the functional acknowledgment status of the group. A table entry is created during the enveloping or deenveloping process. At least one table entry is created, even when the interchange does not contain a functional group.
EDIVTSTU	This table contains information about the use of a transaction within an interchange or group, such as the date and time the transaction was enveloped, and the acknowledgment status of the transaction. A table entry is created during the enveloping or deenveloping process. A transaction that has been enveloped more than once, (REENVELOPED) will have entries in the EDIVTSTU, EDIVTSGP and EDIVTSEV tables.
EDIVTSAU	The application usage table. This table contains information about the processing of a transaction by an application. A table entry is created any time a translation for a transaction is attempted. A transaction that has been translated more than once, (RETRANSLATE), will have multiple entries in this table. A transaction that has never been translated (DEENVELOPED only) has no entries in this table.

The two primary functions of translation services are:

- Translate to standard

The translate-to-standard service transforms data in the application defined format (defined using the *Application data format* selection on the Administrator's Menu (MP01)) into the standard defined format (defined using the *EDI standards* selection on the Administrator's Menu (MP01)) as defined by a map (which is defined using the *Trading partner transactions* selection on the Administrator's Menu (MP01)). See "API - Translate to Standard" on page 3-27 for more information.

- Translate to application

The translate-to-application service transforms data in the standard defined format into the application defined format as defined by a map. See "API - Translate File" on page 3-57 for more information.

The two primary functions have support from the following minor functions:

- End translation

This function signals that there is no more data to process. See “API - End Translation/Enveloping” on page 3-85 for more information.

- Retrieve interchange header

This function retrieves the standard image of the current interchange, and is used by the DataInterchange Utility to create the E optional record. See “API - Retrieve Interchange Header” on page 3-99 for more information.

- Retrieve group header

This function retrieves the standard image of the current group, and is used by the DataInterchange Utility to create the G optional record. See “API - Retrieve Group Header” on page 3-100 for more information.

- Retrieve transaction

This function retrieves the standard image of the current transaction and is used by the DataInterchange Utility to create the T optional record. See “API - Retrieve Transaction Header” on page 3-100 for more information.

- Close and queue interchange

This function forces an interchange to be completed and written to a file associated with the network. See “API - Retrieve Transaction Header” on page 3-100 for more information.

Translate to Standard

This section describes the translate to standard API request. The translate to standard service transforms data in the application defined format (defined using the *Application data format* selection on the Administrator’s Menu (MP01)) into the standard defined format (defined using the *EDI standards* selection on the Administrator’s Menu (MP01)) as defined by a map (which is defined using the *Trading partner transactions* selection on the Administrator’s Menu (MP01)).

The DataInterchange Utility uses this API internally when you issue any of the following commands:

- PERFORM TRANSLATE TO STANDARD
- PERFORM TRANSLATE AND ENVELOPE
- PERFORM TRANSLATE AND SEND

There are enveloping, sending, and testing considerations when using the translate to standard API.

Enveloping and Sending

The following three distinct API functions allow you to translate, envelope, and send a file to a trading partner:

- Translation services with a function code of 131 translates the data. See “API - Translate to Standard” on page 3-27 for more information.
- Enveloping services with a function code of 215 envelopes the data. See “Envelope Function” on page 3-72 for more information.
- Communication services with a function code of 211 sends the data. See “API - Send Transactions and Restart Send Transactions” on page 3-109 for more information.

Using these functions, the translation service translates the data and updates the Transaction Store database with the transaction information and an image of the standard data produced.

The enveloping service builds the proper service segments and retrieves the transaction image from the Transaction Store. It invokes communication services with a function code of 110, which writes the interchange to a file associated with the network.

When the communication service is invoked with a 211 function code, it builds the proper commands for the network and invokes the network to send the file to your trading partner.

This sequence can be shortened, because the translate to standard function of the translation service (function code 131) optionally allows a transaction to be enveloped when the transaction is translated. This is what happens when you use the DataInterchange Utility's PERFORM TRANSLATE AND ENVELOPE command, which is illustrated in Figure 3-2 on page 3-24.

Automatic enveloping at the time of translation is controlled by the **ENVLDELAY** flag in the translator control block. Delayed enveloping may not be used if the translated transactions are not stored in the Transaction Store database. For more information, see the APPDEFS profile in the DataInterchange Administrator's Guide. The **ENVLDELAY** and other fields involved with translation are described in detail in "Translator Control Block (TRCB)" on page A-6. Consider the following points when performing automatic enveloping:

- All data involved in a translation, including the image of the standard data produced, is saved in the Transaction Store. This occurs whether enveloping is part of the translation or is delayed. A standard image is saved even when the translation is not successful, because the image might prove helpful in determining why the translation failed.
- If the enveloping function is separate from translation, there must be a way to tell the enveloping service which transaction should be enveloped. This is done through the unique key value assigned to the transaction when it is added to the Transaction Store.

The key is stored in the database as a 10-byte packed value and is returned in packed format in the **TSKEY** field of the translator control block. It is also returned as a 20-byte character value in the **TSKEYU** field.

When an envelope function is requested, either the TSKEY or TSKEYU value must be accessible to the enveloping service. The program that issues the translate to standard function can save the TSKEY values in a file. The program that issues the envelope calls can read the file to get the TSKEY values. You should store the TSKEY values in a file with other data that can be sorted to organize the data the way you want it enveloped.

If only the TSKEY values are necessary, you can use the DataInterchange Utility's PERFORM QUERY command to extract the TSKEY values that you want enveloped. The PERFORM QUERY command writes the TSKEY values in TSKEY sequence into the file identified by the EDIQUERY ddname.

- The enveloping service, invoked separately, or automatically by setting ENVLDELAY to N, invokes communications services to write the standard data to a file associated with the network. Your program does not need to do this.
- No single API function is the equivalent of the DataInterchange Utility's PERFORM TRANSLATE AND SEND command. Using the API, translating and enveloping is one step, and sending the data is another.

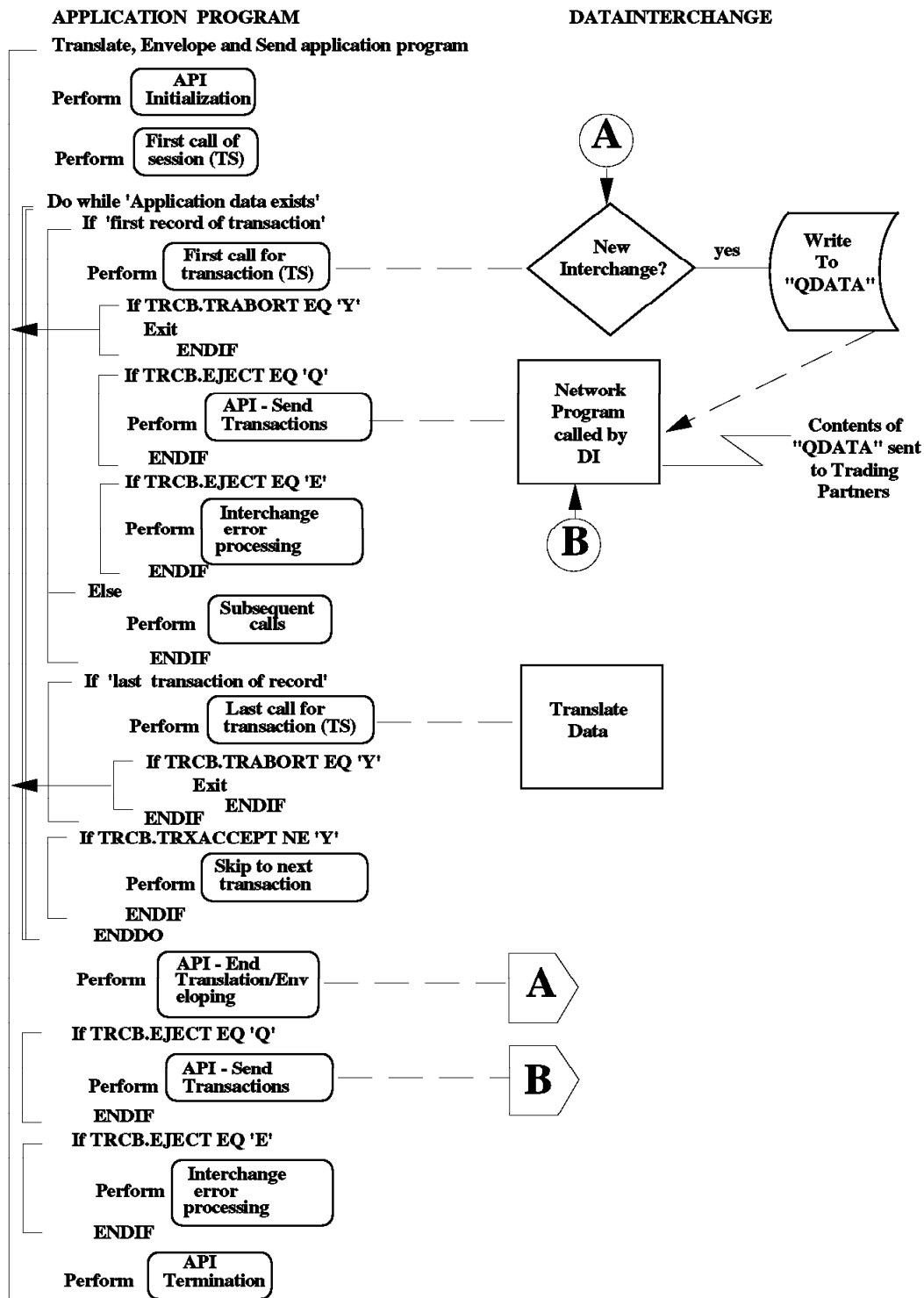


Figure 3-2. Translate, Envelope, and Send Process

Test Translate to Standard

The following describes the two methods for testing your transactions:

1. You can use the test function code 111 to translate to standard format in test mode. In test mode, your program follows the same steps as for normal work. Translation occurs and the interchange created is written to the file associated with the network, but no information is posted to the Transaction Store. If you require or want the Transaction Store services, you must use the test method outlined in option 2. You can use the results of the translation for receiving in test mode.

Note: This form of testing is available only through the application interface.

If you are using the test function code 111, DataInterchange assumes that you require a test usage, if one is available, and automatically sets the **TEST** field in the TRCB to T. Because you are running in test mode, it is also assumed that you want enveloping to occur so that the standard transaction image created will be written to a file that you can inspect. This should be a file that is in no danger of being accidentally sent, should your test translation be followed by a communication service call to send the data. The ddname that writes the standard data produced is EDITEST. Communications ignores any request to send data when the ddname of the file to be sent is EDITEST.

You can verify the results of your program's efforts by examining the standard data produced and the information recorded in the event log. To view or print the event log, choose EVENT LOGGING from the Administrator's Menu.

Note: EDITEST is a temporary storage queue in the CICS environment.

2. You can use the test usage indicator to inform trading partners that a transaction you are sending is for test purposes only. When you use the test indicator, translation and communications occur normally (including the updating of the Transaction Store database), except that the envelope carries an indication that the data is for testing purposes. You may prefer this method because the entire path from you to your trading partner, including any returned network or functional acknowledgments, can be tested. To use this method, set the test usage field to T on the Send usage panel. When using the DataInterchange Utility, you must do one of the following:

- Set the test indicator flag in the C record to T or U to ensure that this is a test usage.
- Set the RAWUSAGE keyword on the PERFORM command to T to indicate that test mode is being used with RAWDATA.

When you use the API, set the **TEST** field in the TRCB to T or U.

Note: Only the X12 (ISA) and EDIFACT (UNB) envelope types have a test or production flag defined. DataInterchange does not mix test transactions with production transactions in the same interchange; however, only ISA and UNB indicate the type of data to the receiver.

Translate to Standard Data Modes

Multiple calls to the translator are usually required to provide the translator with the data you want translated. A final call indicates that all the data has been provided and the translation should begin.

The three modes of operation, based on the type of data being provided for the translator, are:

- Single unit of work mode

In single unit of work mode, all application data needed for a translation is provided in a single call to the translator. This happens when the application data format consists of a single structure definition, or when there are multiple structures, all of them have **passed separately** flags of 'N'. It is then possible to pass all the data to the translator and have the translation take place with a single request. You do this by setting the **EJECT** flag to Y, and either not providing a structure ID in the **ATSID** field or setting the **ATSID** field equal to the base structure of the application data definition.

If you want to design your application so that translation is always a two-step process, where the first step provides the data and the second step translates the data, set the **EJECT** flag to W. The translator waits for a call with the **EJECT** flag set to Y before doing translation, even when single unit of work mode is in effect. This two-step process facilitates the application design. As you read the details of the API request, you see that considerable activity takes place on the first and last calls for a transaction. If you do not set the **EJECT** field to W, a single call replaces the first and last calls. In error situations with only a single call, it can be difficult to determine the type of error that occurred.

- Multiple unit of work mode.

In multiple unit of work mode, more than one application record contributes to the translation. For example, if you are creating a purchase order, there may be a header record, many detail records, and a trailer record. With each call to the translator, you provide a single record that corresponds to a structure in the application data definition, and all subordinate structures that have not been passed separately.

The **ATSID** field is used to tell the translator the name of the structure provided in the TRIDB buffer. When all the data has been provided, one more call to the translator should be made using an **EJECT** flag of Y. This tells the translator that the translation should begin.

Note: No data is provided when EJECT is set to Y. This setting indicates that all data has been provided.

Until now, the translator has been buffering all data provided in the previous calls. The translator must have all the data before any translation can take place.

- RAWDATA mode

In the single unit of work and multiple unit of work modes, the application knows the data that it is processing and communicates that information to the translator through the **ATSID** field. Use the RAWDATA mode only if the application does not know the data it is processing.

For example, if you are writing a general-purpose utility program, your program probably is not aware of all the different data formats you are being asked to process. The RAWDATA mode reverses the roles of the application program and the translator. The translator returns the structure name to the application, and tells the application it is time to translate using the **TRNSTAT** flag.

For more information about RAWDATA processing, see “Special Considerations (TS)” on page 3-40.

Many calls can be made to the translator during a translate to standard session, but the following require special attention:

- The first call to the translator after a DataInterchange initialization or after a translation service termination. This is the call that establishes default values that apply from the first call until a translator termination call is received.
- The first call to the translator for a transaction. This is when key information concerning the transaction is provided, and the translator determines the relationship that this transaction has with the previous transaction.
- The final call to the translator for a transaction. This is when translation from the application format to the standard format actually takes place.
- The final call to the translator to terminate the session. This is when cleanup and final processing occur.

All other calls merely involve the movement of data from the application buffers into the translator buffers. The following sections describe these four critical calls in detail.

API - Translate to Standard

You can use the two following function codes to request a translate to standard function:

- Function code 131 requests a translation in production mode.
- Function code 111 requests a translation in test mode.

Before Release 4 of DataInterchange, there were considerable differences in your program's operation using function codes 131 and 111. Function code 111 allowed you to test your mapping, but did not help in testing your application program. The switch from function 111 to 131 required program changes.

Using this release of DataInterchange, you do not have to change your program to switch from test to production mode. You only need to change the function code used to invoke the services. Differences between the test mode and production mode are:

- DataInterchange forces the TEST flag in the TRCB to a value of T. If a test mapping exists, it uses the test mapping when testing your program. If a test mapping does not exist, it uses the production mapping.
- DataInterchange forces the ENVLDELAY flag in the TRCB to a value of N. This writes the standard data produced to a file.
- DataInterchange forces the **FILEID** field in the TRCB to EDITEST. If a communications service request to send transaction data is received and the file name used is EDITEST, DataInterchange ignores the request.
- Control numbers are not taken from the trading partner profile member. Instead, they are assigned values of Tnnnnn, where nnnnn is a sequential value starting with 00001.
- DataInterchange automatically logs the standard image produced and the application data that is provided.
- DataInterchange does not write any data to the Transaction Store, to prevent cluttering up the store with test data.
- Statistics maintained by the Management Reporting component of DataInterchange are not updated.

The following is the basic format of the API request:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter	Description
-----------	-------------

SNB	The service name block identifying translation services. Valid values are:
-----	--

Field	Value
ZSNBLL	32
ZSNBNAME	TRANPROC (identifies translation services)
ZSNBPC	6 (number of parameters in the call)

CCB	The common control block used to initialize DataInterchange.
-----	--

FCB	The function control block with a ZFCBFUNC value of 131 (indicates a production program) or 111 (indicates a test program).
-----	---

TRCB	The translator control block. See "Translator Control Block (TRCB)" on page A-6 for details about this block.
------	---

TRIDB	Is the input data block. This block contains the application data to be translated. The format of the data in this block must match the format of the data defined in the Application Data Format.
-------	--

TRODB Is the output data block. DataInterchange uses the output data block as a work buffer. The buffer must be at least 32000 bytes in length. Its maximum size is the size of the maximum standard segment that is produced, excluding the BIN segment.

In the remainder of this section, the phrase *calling the translator* means *Invoking the DataInterchange API with a request to translate to standard*. When reference to a field within the TRCB, the field is uppercase and highlighted (for example, **EJECT**).

First Call of Session (TS)

There are numerous fields in the control blocks defined for the translate to standard API function whose values only need to be established once. These values can be established prior to the first call. They do not have to be refreshed or changed before any other call. Each of the control blocks, the fields within the control blocks, and the initialization considerations are described in the following tables.

Table 3-10. Service Name Block (SNB) Initialization for Translate to Standard

SNB	Initialization Value
ZSNBLL	32
ZSNBNAME	TRANPROC (the service name for translation services)
ZSNBPC	6 (all calls for translation services have six parameters)

Table 3-11. Function Code Block (FCB) Initialization for Translate to Standard

FCB	Initialization Value
ZFCBLL	4
ZFCBFUNC	131 (for a production program) 111 (for a test program)

Table 3-12 (Page 1 of 3). Translator Control Block (TRCB) Initialization for Translate to Standard

TRCB	Initialization Value
BLKLEN	1536 (the length of the translator control block in Version 2)
BLKNME	EDITRCB
BLKTYPE	There are two possible formats for the TRIDB and TRODB buffer. One format limits the buffer size to 32768. The other format has no limit. A value of H indicates that the unlimited format is used. Any other value indicates that the format with a limit of 32768 is being used.
XPANDED	Y
ENVLDELAY	This field indicates whether enveloping should occur at the same time as translation. A value of Y indicates that enveloping should not occur. Any other value indicates that enveloping should occur.
SCOPE	A value of E requests interchange level recovery. DataInterchange does not issue a database COMMIT until an interchange is complete. A SCOPE value of E is only honored if enveloping occurs. Any other value indicates that the recovery scope should be at the transaction level, and DataInterchange issues a database COMMIT at the end of every successful transaction. Note: This field contains values that affect the recovery scope during the session. See "Send Recovery Scope" on page 3-42 for additional information.

Table 3-12 (Page 2 of 3). Translator Control Block (TRCB) Initialization for Translate to Standard

TRCB	Initialization Value
INMEMTRANS	<p>This parameter applies only if the value of SCOPE is E and indicates the maximum number of transactions that should be maintained in virtual storage before any database updates are attempted. Database updating occurs when INMEMTRANS or the end of the interchange is reached. This value is important in an environment where multiple application programs are requesting translation services concurrently. The higher the value for INMEMTRANS, the more concurrency can be achieved. See the description on INMEMTRANS on page A-23 for more information on this field.</p> <p>Note: This field contains values that affect the recovery scope during the session. See “Send Recovery Scope” on page 3-42 for more information.</p>
FILEID	<p>If enveloping occurs, FILEID provides the ddname of the file where the transaction data is written when an interchange is complete. This field is optional. If it is not provided, the ddname specified in the <i>Trans data queue</i> field of the network profile (NETPROF) is used instead. This value does not have to be a constant for the entire session, but if it is used, it must be set before an interchange is complete.</p>
ITPBREAK	<p>Flag that indicates whether a change in the internal trading partner ID (vendor number) causes a new interchange. A value of Y indicates that each time the internal trading partner ID changes, the current interchange should be completed and written and a new interchange started. Any other value indicates that a new interchange should be started only when the trading partner nickname changes in value.</p> <p>For example, if an application has many different vendor numbers for the same trading partner, and you want each interchange sent to the trading partner to contain data associated with one of those vendors, set the ITPBREAK value to Y. If you want all the data, regardless of the vendor number, to be put into one interchange for the trading partner, set the ITPBREAK value to N. N is suggested, but any value other than Y is treated as an N.</p>
BATCHID	<p>The batch ID is a value that can be associated with a transaction and used later to retrieve the transaction quickly. The database has an alternate key value using the batch ID. Besides using the transaction handle itself, this is the quickest way to retrieve a transaction from the database during the selection process. If you do not provide a value for BATCHID, the field takes on the default value of DDHHMMSS, where DD is the current day of the month, and HHMMSS is the current time. For more information, see “Batches and Bundles” on page 3-43.</p> <p>Note: This field is checked with each call to the translator and if a value is not supplied, the default value is used. The value that exists when the “last call for transaction (TS)” is made will be the value that is associated with the transaction.</p>
TRXLIFE	<p>The amount of time the transaction remains in the Transaction Store before it is eligible to be purged. If you do not provide a value, it takes on the default value of 30 days.</p> <p>Note: This field is checked with each call to the translator and if a value is not supplied, the default value is used. The value that exists when the “last call for transaction (TS)” is made will be the value that is associated with the transaction.</p>
IMGLIFE	<p>The amount of time the transaction’s image (that is, the standard data produced) remains in the Transaction Store. If you do not provide a value, it takes on the default value of 30 days.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. This field is checked with each call to the translator and if a value is not supplied, the default value is used. The value that exists when the “last call for transaction (TS)” is made will be the value that is associated with the transaction. 2. This field is not currently supported.

Table 3-12 (Page 3 of 3). Translator Control Block (TRCB) Initialization for Translate to Standard

TRCB	Initialization Value
ENVLDATE	The earliest date that the transaction is considered eligible for enveloping. This parameter is honored only when delayed enveloping is being used (that is, ENVLDELAY is set to Y). If you do not provide a value, the transaction is considered eligible for enveloping immediately. Note: This field is checked with each call to the translator and if a value is not supplied, the default value is used. The value that exists when the "last call for transaction (TS)" is made will be the value that is associated with the transaction.
ERRFILTER	Values that indicate which errors should be filtered out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more information, see "Error Filtering" on page 1-4.
RAWDATAOUT	A value of 'Y' indicates that RAWDATA output is desired for Fixed-to-Fixed mappings. Any other value indicates that 'C' and 'D' record output is wanted.
FFILEID	The ddname of the file where the translated data for Fixed-to-Fixed translation is written. This field is optional. If this field is not provided, the ddname formed from the concatenation of <i>Application file name</i> from the target application data format and the <i>File suffix</i> from the trading partner profile will be used. This value does not have to be a constant for the entire session. However, if you want to use this field, it must be set before an interchange is completed.

Table 3-13. Translator Output Data Block (TRODB) Initialization for Translate to Standard

TRODB	Initialization Value
BLKLEN	Set to the size of the data block including the BLKLEN field. The minimum value for this field is 32000 bytes.
RESERVED	Initialize this field to zeros.

Because the first call for a session also qualifies as the first call for a transaction, you should also follow the instructions in "First Call for Transaction (TS)."

First Call for Transaction (TS)

On the first call to the translator for a transaction, the information required to locate the **map** that directs the transformation from application data to standard data must be provided. The EDI administrator creates the map for a specific application data format definition. The **ATFID** field contains the definition name for which the map was defined and for which data is being provided. However, many different users can use a map, and the EDI administrator also defines these users by adding trading partner send usage entries to the map.

The fields for the trading partner send usage are:

Field	Description
INTPID	The internal trading partner ID. This is the name of the trading partner known to your application program. It is sometimes called a vendor number.

TEST The usage indicator. Valid values are:

Value	Description
-------	-------------

I	Specifies that this is an information transaction and that an information usage should be used, if one exists. If an information usage does not exist, a production usage will be used instead. Even when a production usage is used, the transaction is flagged as an information transaction.
P	Specifies that this is a production transaction and that only a production usage should be used. This is the default.
T	Specifies that this is a test transaction and that a test usage should be used, if one exists. If a test usage does not exist, a production usage will be used instead. Even when a production usage is used, the transaction is flagged as a test transaction.
U	Specifies that the translator should determine if the transaction is test, information, or production based on the usage found. If a test usage exists, the transaction is a test transaction, and a value of T is returned. If an information usage exists, the transaction is an information transaction, and a value of I is returned. If only a production usage exists, the transaction is a production transaction, and a value of P is returned.

DataInterchange requires these fields to determine what translation is being requested. Table 3-14 describes the TRCB fields that must be set by the application on the first call for translation:

Table 3-14 (Page 1 of 4). Initialization - First Call for Transaction - TRCB

TRCB	Initialization Value						
ATFID	The application data format ID for which data is being provided.						
INTPID	The internal trading partner ID (vendor number) to which the standard data produced is destined. For more information on the RAWDATA interface, see the "Send RAWDATA" on page 3-40.						
TEST	A value of T for a test transaction, P for a production transaction, I for an information transaction, or U for the transaction status to be determined by the usage that exists.						
RAWDATA	A value of N indicates that the RAWDATA interface is not being used. Any value other than Y indicates that RAWDATA is not being used. If RAWDATA is being used, see "Send RAWDATA" on page 3-40 for a description of that interface.						
EJECT	<p>The EJECT flag serves the following purposes:</p> <table border="1"> <thead> <tr> <th>Flag</th><th>Description</th></tr> </thead> <tbody> <tr> <td>X</td><td>Indicates that a partial structure is provided. For more information on partial structures, see "Providing a Partial Structure" on page 3-41.</td></tr> <tr> <td>Y</td><td>Indicates that all the application data has been provided and translation should begin. For more information on this flag, see "Translate to Standard Data Modes" on page 3-25. For an explanation of what happens when the EJECT field is set to Y, see "Last Call for Transaction (TS)" on page 3-36.</td></tr> </tbody> </table> <p>Note: If this is a single unit of work, and you decide to use an EJECT value of Y with the first request (provide the data and translate it at the same time), see "Last Call for Transaction (TS)" on page 3-36.</p>	Flag	Description	X	Indicates that a partial structure is provided. For more information on partial structures, see "Providing a Partial Structure" on page 3-41.	Y	Indicates that all the application data has been provided and translation should begin. For more information on this flag, see "Translate to Standard Data Modes" on page 3-25. For an explanation of what happens when the EJECT field is set to Y, see "Last Call for Transaction (TS)" on page 3-36.
Flag	Description						
X	Indicates that a partial structure is provided. For more information on partial structures, see "Providing a Partial Structure" on page 3-41.						
Y	Indicates that all the application data has been provided and translation should begin. For more information on this flag, see "Translate to Standard Data Modes" on page 3-25. For an explanation of what happens when the EJECT field is set to Y, see "Last Call for Transaction (TS)" on page 3-36.						
ATSID	The name of the structure for which data is being provided in the TRIDB.						

Table 3-14 (Page 2 of 4). Initialization - First Call for Transaction - TRCB

TRCB	Initialization Value
BNDLFLAG	<p>The bundle flag is a way to associate transactions. Usually DataInterchange considers each transaction independently of any other transaction. However, transaction associations can be made using the BNDLFLAG.</p> <p>A BNDLFLAG value of Y indicates that this transaction starts a new bundle, or starts an association. DataInterchange refers to the associated transactions as a cluster of transactions. The first transaction of a cluster is called the controlling transaction. This is the transaction that is displayed when you use the Transaction Store Facility.</p> <p>The transaction handle (THANDLE) value of the controlling transaction is the value used by DataInterchange to associate all of the transactions in a cluster. THANDLE is returned in the TSKEY and TSKEYU fields.</p> <p>A BNDLFLAG value of N indicates that this transaction is the last transaction in a cluster. A blank BNDLFLAG value indicates there is no change in bundle status. If a cluster is active, this transaction is considered part of the cluster. For more information, see “Batches and Bundles” on page 3-43.</p>
HOLDFLAG	<p>Use a value of Y if this transaction is to be placed in a hold status when added to the Transaction Store. A transaction in hold status is not available for any other activity, such as enveloping, until it is released. Use a value of N if the transaction is not to be held. Any value other than Y is treated like N.</p>
ROUTCODE	<p>A three-character generic routing code provided by the application and used by DataInterchange to select a generic send usage. A blank specifies a default generic send usage.</p>

Note: When this transaction gets enveloped (either now or at some future date), the envelope function builds the service segments that surround a transaction. These service segments consist of an interchange header that identifies the sender and the receiver, a group header that gathers all transactions of similar characteristics and can be used to identify the application sender and receiver, and a transaction header that provides the name of the transaction being sent. Most of the values for fields in these segments are established by the EDI Administrator, but your program can override certain fields by providing values that you want associated with a transaction. These values are provided with the transaction at the time of translation and DataInterchange uses them when the transaction is enveloped. The following fields in the TRCB may be used to provide overrides. **If you do not use overrides, you should make sure these fields are blank before making the first request for a transaction. These fields are also used as return fields and, therefore, your program must make sure they have the desired values (either blank or an override value) before making each first request.** For more information on service segments, see “Interchange Layer” on page 3-71.

ISYNTAXID	<p>Overrides the interchange syntax ID for E and T envelopes.</p> <p>Note: This field is an override for the interchange header.</p>
ISYNTAXVER	<p>Overrides the interchange syntax version for E and T envelopes.</p> <p>Note: This field is an override for the interchange header.</p>
ISIDQUAL	<p>Overrides the interchange sender ID qualifier for E, I, and X envelopes.</p> <p>Note: This field is an override for the interchange header.</p>
ISID	<p>Overrides the interchange sender ID (interchange header field with IS data type).</p> <p>Note: This field is an override for the interchange header.</p>
ISENDNAME	<p>Overrides the interchange sender name for U and T envelopes.</p> <p>Note: This field is an override for the interchange header.</p>
IREVROUT	<p>Overrides the interchange reverse routing for E envelopes.</p> <p>Note: This field is an override for the interchange header.</p>

Table 3-14 (Page 3 of 4). Initialization - First Call for Transaction - TRCB

TRCB	Initialization Value
IRIDQUAL	Overrides the interchange receiver ID qualifier for E, I, and X envelopes. Note: This field is an override for the interchange header.
IRID	Overrides the interchange receiver ID (interchange header field with IR data type). Note: This field is an override for the interchange header.
RECVNAME	Overrides the interchange receiver name for U and T envelopes. Note: This field is an override for the interchange header.
IROUTEADDR	Overrides the interchange routing address for E envelopes. Note: This field is an override for the interchange header.
IVERREL	Overrides the interchange version and release (interchange header field with VR or LV data type). Note: This field is an override for the interchange header.
ISPW	Overrides the interchange password (interchange header field with PW data type). Note: This field is an override for the interchange header.
IAPREF	Overrides the application reference (interchange header field with AP data type). Note: This field is an override for the interchange header.
ISTDID	Overrides the interchange standard ID for I and X envelopes. Note: This field is an override for the interchange header.
IPRIOR	Overrides the interchange priority code for E and T envelopes. Note: This field is an override for the interchange header.
ICOMMAGREE	Overrides the interchange communication agreement E and T envelopes. Note: This field is an override for the interchange header.
GSIDQUAL	Overrides the group sender ID qualifier for E envelope groups. Note: This field is an override for the group header.
GSID	Overrides the group sender ID (group header field with AS data type). Note: This field is an override for the group header.
GRID	Overrides the group receiver ID (group header field with AR data type). Note: This field is an override for the group header.
GRIDQUAL	Overrides the group receiver ID qualifier for E envelope groups. Note: This field is an override for the group header.
GAPW	Overrides the group password (group header field with PW data type). Note: This field is an override for the group header.
GVER	Overrides the group version (group header field with VR data type). Note: This field is an override for the group header.
GREL	Overrides the group release (group header field with LV data type). Note: This field is an override for the group header.
GRESAGENCY	Overrides the group responsible agency code for E, U, and X envelopes. Note: This field is an override for the group header.
TVER	Overrides the transaction version (transaction header field with VR data type). Note: This field is an override for the transaction header.

Table 3-14 (Page 4 of 4). Initialization - First Call for Transaction - TRCB

TRCB	Initialization Value
TREL	Overrides the transaction release (transaction header field with LV data type). Note: This field is an override for the transaction header.

Table 3-15. Initialization - First Call for Transaction - TRODB

TRIDB	Initialization Value
BLKLEN	The length of the TRIDB, including this field. If you are using the same TRIDB for all calls, this field needs to be initialized only once.
RESERVED	Set this field to zeros. If you are using the same TRIDB for all calls, this field needs to be initialized only once.
DATALEN	Set this field equal to the number of bytes (characters) present in the DATA field. This number should match the number of bytes that were defined for the structure when the application data format was defined.
DATA	This is the actual application data. The format of the data must match the format for the structure (provided in ATSID) as defined in the application data format definition.

When initialization is complete, the translator is invoked with the following API request:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

When the translator receives this request, it verifies that everything it needs to translate the transaction is available. The **ATFID**, **INTPID**, and **TEST** fields are used to locate the trading partner usage and map. The control string generated from the map and the standard and transaction descriptions are retrieved.

Numerous errors can occur at this point. See "Translation Return Codes" on page B-19 for a description of the errors from the translator. If an error occurs, the return code and extended return code for the error are posted in the CCB's **ZCCBRC** and **ZCCBERC** fields.

The two fields within the TRCB that indicate the status of the transaction and the status of the translator are:

Field	Description
TRXACCEPT	This field has a value of Y if the transaction is acceptable and can be translated. A value other than Y indicates that something needed to process this transaction is missing and, therefore, translation cannot occur. At this point, your program has many options. The option taken by the DataInterchange Utility is to copy all the data for this transaction to an exception file. When the error is analyzed and corrected, the data in the exception file can be reprocessed. If the value is not Y, and your program continues, the next call to the translator is considered to be the first call for the next transaction.
TRABORT	This field has a value of Y if the error is so severe that the translator does not continue processing. If the value is Y, and your program continues after such an error, the next call you make to the translator is considered the first call for the session.

Table 3-16 on page 3-35 describes the TRCB fields that are returned on the first call for a transaction. They identify the key attributes of the transaction.

Table 3-16. Fields in TRCB Returned on First Call for a Transaction

TRCB	Description												
TRNID	The standard transaction or message ID that is generated from this application data.												
TEST	If you used a TEST value of U, TEST becomes an output field that indicates whether a production, test, or information usage is being used.												
ENVTYPE	Indicates the envelope type that is used when this transaction is enveloped. Possible values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>E</td><td>UNB/UNZ</td></tr> <tr> <td>I</td><td>ICS/ICE</td></tr> <tr> <td>T</td><td>STX/END</td></tr> <tr> <td>U</td><td>BG/EG</td></tr> <tr> <td>X</td><td>ISA/IEA</td></tr> </table>	Value	Description	E	UNB/UNZ	I	ICS/ICE	T	STX/END	U	BG/EG	X	ISA/IEA
Value	Description												
E	UNB/UNZ												
I	ICS/ICE												
T	STX/END												
U	BG/EG												
X	ISA/IEA												
MAPKEY	Identifies the trading partner transaction used to translate the application data.												
TPNICK	The trading partner nickname associated with the transaction.												

Enveloping During Translation: If enveloping is occurring at the same time as translation (**ENVLDELAY** value of N), the translator checks the first call to see if the transaction just provided should be put into the current interchange, or if it should cause a new interchange to be started. See “Fields Causing New Interchange” on page 3-83 for a description of the fields that cause a new interchange to be started.

If a new interchange should be started, the current interchange is completed and written to the file associated with the network, or to the file identified by the **FILEID** field. Table 3-17 lists the fields in the TRCB returned when an interchange is written. The **EJECT** field indicates whether an interchange was written. The other fields provide information about the interchange, the network, and the file receiving the interchange.

Table 3-17 (Page 1 of 2). Fields in TRCB Returned When an Interchange is Written

TRCB	Description						
EJECT	If an interchange was written to the file associated with the network, the EJECT field has one of the following values: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Q</td><td>Indicates the interchange was written successfully</td></tr> <tr> <td>E</td><td>Indicates the interchange was not written successfully</td></tr> </table> <p>Note: When the EJECT field has a value of E indicating that an error occurred, the QRC and QERC fields contain values indicating the error. The specific error that occurred is logged by DataInterchange communications services. The most likely error is that the ddname could not be opened.</p>	Value	Description	Q	Indicates the interchange was written successfully	E	Indicates the interchange was not written successfully
Value	Description						
Q	Indicates the interchange was written successfully						
E	Indicates the interchange was not written successfully						
QRC	The return code associated with the error.						
QERC	The extended return code associated with the error.						
DSNAME	The physical dataset name to which the interchange was written. In CICS, the DSNAME has the same value as QDDNAME . This is the name of the temporary storage queue containing the interchange.						
QNETID	The name of the network associated with the trading partner.						
QPTTOPT	Y if the network identified by QNETID is a point-to-point network.						

Table 3-17 (Page 2 of 2). Fields in TRCB Returned When an Interchange is Written

TRCB	Description
QSRPGM	Y if the network identified by QNETID has a network send/receive program associated with it. If the network does not have an associated send/receive program, there is no need to send the interchange. A network might exist without a send/receive program for creating interchanges that are sent to a trading partner using some other means.
QDDNAME	The ddname to which the interchange is written. This field does not have a value if a point-to-point network is used. In CICS, this is the name of the temporary storage queue to which the interchange was written.
QTPNICK	The trading partner nickname for which the interchange was built.
QSIZE	The total number of bytes in the interchange. QSIZE is stored as a 4-byte binary value.
QBT	Character representation of QSIZE .

See “Sending Transaction Data” on page 3-82 for an explanation of items to be considered if your application program is using the communications services, send API, for sending the data that was just written to the file. See “API - Send Transactions and Restart Send Transactions” on page 3-109 for more information on API.

Subsequent Calls

Calls to the translator between the first and last calls transfer transaction data between the application program and the translator. The application data is placed in the TRIDB. The structure name describing the data is placed in the **ATSID** field. When the call is made, the translator copies the data from the TRIDB into the translator's internal buffers. With each call, the **TRXACCEPT** and **TRABORT** flags should be checked.

When all the data has been provided, a final call is necessary to tell the translator to translate the data into the standard format. See “Last Call for Transaction (TS)” for more details on the final call.

Last Call for Transaction (TS)

The last call for a transaction is indicated by an **EJECT** field set to Y. At this point, you can also set the other fields if you do not want the values established explicitly or by default in the first call. The following are fields you might choose to provide with values:

- BATCHID
- TRXLIFE
- IMGLIFE
- ENVLDATE

Note: See “Send Recovery Scope” on page 3-42 for details about the use of the **NOCOMIT** flag.

When the fields are initialized, the following API request should be issued:

```
FXXZccc(CCB,SNB,FCB,TRCB,TRIDB,TRODB)
```

Note: No data is supplied on this request, but the TRIDB and TRODB parameters are still required.

When the translator receives this request, it translates data from the application format into the standard format in accordance with the mapping instructions. Numerous errors can occur at this point. They are listed in “Translation Return Codes” on page B-19.

If errors occur, the error codes associated with the most severe error are posted in the CCB's **ZCCBRC** and **ZCCBERC** fields. An acceptable transaction can have errors. An acceptable error level for the transaction is established when a usage is created by the EDI Administrator.

The two fields within the TRCB that must be checked to determine the severity of an error are:

Field	Description
TRXACCEPT	This field has a value of Y if the transaction is acceptable and can be translated. A value other than Y indicates that something needed to process this transaction is not available and translation cannot occur.
TRABORT	<p>This field has a value of Y if the error is so severe that the translator does not continue.</p> <p>Your program can continue in many ways. The DataInterchange Utility would copy the data for this transaction to the exception file where it could be reprocessed later. If the value is not Y, the next call to the translator is considered the first call for the next translation.</p> <p>If the value is Y, and your program continues after such an error, the next call you make to the translator is considered the first call for the session.</p>

Numerous fields within the TRCB are provided as output at this time.

Translation - TRCB Fields: Table 3-18 shows all the TRCB fields that are updated, based on translation.

Table 3-18. Fields in TRCB Returned on Translation

TRCB	Description
APPCTLNUM	See XACFIELD .
XACFIELD	The application control value. The application control value is defined by the application field with an AC data type, or by the fields that were assigned when the trading partner transaction map was created. It is the value that uniquely identifies the transaction to the application.
TRXACCEPT	Y if the transaction had an acceptable translation.
TRABORT	Y if an error occurred that was so severe that the translator did not continue. If this flag is set, the translator has terminated of its own accord.
TSKEY	The key value assigned to the transaction within the Transaction Store. This is called the transaction handle, and has a format of YYYYMMDDHHMMSSxxnnnn. It is returned as a 10-byte packed value in this field, which is how it is stored in the database.
TSKEYU	The key value assigned to the transaction within the Transaction Store. The same value as that in the TSKEY field, except that TSKEYU is an unpacked (character) 20-byte value.
ERRNUM	The total number of errors flagged while translating the data.
ERRCDES	An array of the first 10 different errors flagged in translating the data. Each possible error detected by the translator has a unique error code. See Table A-5 on page A-28 for a list of values.

Enveloping - TRCB Fields: Numerous fields within the TRCB are related to the current status of an interchange that is being created. Table 3-19, Table 3-20, and Table 3-21 on page 3-39 describe the fields relating to the following items:

- Interchange header and trailer
- Group header and trailer
- Transaction header and trailer

Table 3-19. Fields in TRCB Related to Interchange Header and Trailer

TRCB	Description
NEWENV	Y if the transaction caused a new interchange to be started. If a copy of the interchange header is needed, a function code value of 1 can be used to obtain an exact image of the interchange header. For more information, see “API - Retrieve Interchange Header” on page 3-99.
IHCTL	See IHXCTL .
IHXCTL	The interchange control number assigned to the current interchange. The value is returned in this field as a right-justified value with leading zeros.
GRPNUM	The total number of groups within the interchange at the current time, stored as a 4-byte binary value. See IGT .
TRNNUM	The total number of transactions within the current interchange at the current time, stored as a 4-byte binary value. See ITT .
SEGNUM	The total number of segments within the current interchange at the current time, stored as a 4-byte binary value. See IST .
ESIZE	The total number of bytes within the current interchange at the current time, stored as a 4-byte binary value. See IBT .
ISID	The interchange sender ID (interchange header field with IS data type).
IRID	The interchange receiver ID (interchange header field with IR data type).
IDATE	The interchange date (interchange header field with DT data type).
ITIME	The interchange time (interchange header field with TM data type).
IVERREL	The interchange version and release (interchange header field with VR or LV data type).
IGT	Character representation of GRPNUM .
ITT	Character representation of TRNNUM .
IST	Character representation of SEGNUM .
IBT	Character representation of ESIZE .
ISPW	The interchange password (interchange header field with PW data type).
IAPREF	The application reference (interchange header field with AP data type).

Table 3-20 (Page 1 of 2). Fields in Translator Control Block (TRCB) Related to Group Header and Trailer

TRCB	Description
NEWGRP	Y, if the transaction caused a new group to be started. If a copy of the group header is needed, a function code value of 2 can be used to obtain an exact image of the group header. For more information, see “API - Retrieve Group Header” on page 3-100.
GHCTL	See GHXCTL .
GHXCTL	The group control number assigned to the current group. The value is returned in this field as a right-justified value with leading zeros.

Table 3-20 (Page 2 of 2). Fields in Translator Control Block (TRCB) Related to Group Header and Trailer

TRCB	Description
TRNGRP	The total number of transactions within the current group at the current time, stored as a 4-byte binary value. See GTT .
GSID	The group sender ID (group header field with AS data type).
GRID	The group receiver ID (group header field with AR data type).
GDATE	The group date (group header field with DT data type).
GTIME	The group time (group header field with TM data type).
GAPW	The group password (group header field with PW data type).
GVER	The group version (group header field with VR data type).
GREL	The group release (group header field with LV data type).
GTT	Character representation of TRNGRP .

Table 3-21. Fields in the Translator Control Block (TRCB) Related to Transaction Header and Trailer

TRCB	Description
NEWTRN	Y, if the transaction caused a new transaction to be started. If a copy of the transaction header is needed, a function code value of 3 can be used to obtain an exact image of the transaction header. For more information, see “API - Retrieve Transaction Header” on page 3-100.
THCTL	See THXCTL .
THXCTL	The transaction control number assigned to the current transaction. The value is returned in this field as a right-justified value with leading zeros.
SEGTRN	The total number of segments within the current transaction stored as a 4-byte binary value. See TST .
TTC	The current transaction or message ID value. See also TRNID .
TVER	The transaction version (transaction header field with VR data type).
TREL	The transaction release (transaction header field with LV data type).
TST	Character representation of SEGTRN .

Last Call for Session (TS)

If the translator is called at least once and the **TRABORT** flag does not have a value of Y, the translator should be invoked one last time with a termination function code.

The termination function code signals the translator that the application is finished making translation requests. On receiving the request, the translator releases any resources that have been acquired. If an interchange is active, the translator completes and writes the current interchange.

Special Considerations (TS)

The following sections list special considerations.

Send RAWDATA: In usual processing, it is assumed that the application has detailed knowledge of the data being processed and communicates this information to the translator by setting the appropriate fields in the TRCB. The specific values the application must communicate to the translator are:

- The internal trading partner ID associated with this transaction in the **INTPID** field
- The structure name for the data being provided in the TRIDB in the **ATSID** field
- The end of the transaction, indicated by using a Y in the **EJECT** field

If you are writing a general-purpose application program that is capable of handling numerous application files, similar to the DataInterchange Utility, your application may not be aware of all the details for all possible files.

The translator has a RAWDATA mode that can be used in these situations. The RAWDATA mode reverses the roles of the application and the translator in the following ways:

1. Rather than the application telling the translator which structure is provided, the translator tells the application.
2. Rather than the application telling the translator the internal trading partner ID, the translator returns the ID value to the application.
3. Rather than the application telling the translator the transaction is complete, the translator tells the application it is time to translate the data.

RAWDATA mode is signaled on the first call to the translator for a transaction by setting the **RAWDATA** field to Y and setting the **ATFID** to the application data format ID that is being used for this transaction. If your application knows the internal trading partner ID, that value can be put into the **INTPID** field.

If your application does not know the internal trading partner ID, initialize the **INTPID** field with blanks, and the translator can extract the internal trading partner ID from the application data.

Note: When you use the DataInterchange Utility, the internal trading partner ID must be part of the application data and is extracted by the translator when the appropriate structure is provided. However, an API program can explicitly set the internal trading partner ID value in the **INTPID** field, removing the requirement that a field in the application data contain this value.

For RAWDATA mode to succeed, the EDI Administrator must have provided the raw data specifications when the application data format (**ATFID** value) was defined. The raw data specifications include:

- The name of the field that contains the internal trading partner ID
- Either the name of a structure that starts a transaction, or the name of a structure that ends a transaction, or both

If possible, provide the name of a structure that ends a translation. The translator can then detect the end of the current transaction before receiving the record that begins the next transaction.

- The offset into each structure in which a value exists that uniquely identifies the structure
- A unique value that identifies each structure defined in the application data format

When using generic send usages, the application has the capability of selecting a specific generic usage by generic routing code. The routing code is provided by the application in the TRCB, C record, or in an application field (raw data only).

After each request, the translator posts information into the TRCB that specifies the status of the transaction. The following are the TRCB fields that are updated:

Field	Description												
ATSID	Contains the name of the structure that was received, based on the raw data specifications.												
INTPID	<p>Contains the name of the internal trading partner ID. The translator returns the internal trading partner ID value in this field when the structure containing the value is received. At this point, the processing in the first call for a transaction takes place.</p> <p>If the internal trading partner ID is not in the first structure, first call processing is delayed until the internal trading partner is received. This is critical to your program, because if the transaction is not acceptable (because of missing items in the environment, such as a mapping), and previous structures have been received, the translator must be told to discard all the previous data. This is done by making one more request with an EJECT value of F.</p> <p>For example, if the internal trading partner ID is provided in the third structure, and an error is returned on this third structure because a map could not be found, the translator must be called once with an EJECT value of F. This causes the previously accepted structures to be discarded.</p> <p>If a field containing the internal trading partner ID is not defined in the application data format, or if the field is defined but it contains all blanks, then the value of INTPID that was provided on the first call for the transaction will be the default value that is used.</p>												
TRNSTAT	<p>Contains a value that indicates the status of the transaction and, therefore, determines what your program should do next. The following are possible values in this field, and the implications to your program:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>I</td><td>The data that you supplied has been ignored. It might be ignored because the translator could not determine the structure based on the raw data specifications, or the raw data specifications indicated that a specific structure started each transaction and that structure has not yet been supplied.</td></tr> <tr> <td>S</td><td>The data you supplied was identified and has started a transaction.</td></tr> <tr> <td>C</td><td>The data you supplied was identified and continued the transaction.</td></tr> <tr> <td>R</td><td>The data you supplied is identified to start a transaction, but a current transaction is already in progress. Follow the instructions for the last call for a transaction to terminate the previous transaction, and call the translator again with the same data.</td></tr> <tr> <td>Y</td><td>The data you supplied is identified to end a transaction. To force the translation of the current data, follow the steps for the last call for a transaction.</td></tr> </table>	Value	Description	I	The data that you supplied has been ignored. It might be ignored because the translator could not determine the structure based on the raw data specifications, or the raw data specifications indicated that a specific structure started each transaction and that structure has not yet been supplied.	S	The data you supplied was identified and has started a transaction.	C	The data you supplied was identified and continued the transaction.	R	The data you supplied is identified to start a transaction, but a current transaction is already in progress. Follow the instructions for the last call for a transaction to terminate the previous transaction, and call the translator again with the same data.	Y	The data you supplied is identified to end a transaction. To force the translation of the current data, follow the steps for the last call for a transaction.
Value	Description												
I	The data that you supplied has been ignored. It might be ignored because the translator could not determine the structure based on the raw data specifications, or the raw data specifications indicated that a specific structure started each transaction and that structure has not yet been supplied.												
S	The data you supplied was identified and has started a transaction.												
C	The data you supplied was identified and continued the transaction.												
R	The data you supplied is identified to start a transaction, but a current transaction is already in progress. Follow the instructions for the last call for a transaction to terminate the previous transaction, and call the translator again with the same data.												
Y	The data you supplied is identified to end a transaction. To force the translation of the current data, follow the steps for the last call for a transaction.												

Providing a Partial Structure: Usually, all data for a given structure is provided to the translator in a single request by moving the complete structure into the translator input data block (TRIDB). However, if it is not possible for all the data to be provided in a single request, you can signal that a partial structure is being provided by setting the **EJECT** field to X.

The translator usually ignores the **DATALEN** field of the TRIDB, because the amount of data provided is defined by the structure name (**ATSID**) and the definition of structure in the application data format definition. If the **EJECT** field is set to X, **DATALEN** should be the amount of data provided with this request.

Continue to provide data with an EJECT value of X until you are ready to provide the final data for the structure. The request to provide the final data for the structure should have the **EJECT** flag set to blank.

Note: If RAWDATA is being used and partial data is being provided for the structure that contains the internal trading partner ID field, the internal trading partner ID field must be present in the first partial record provided for the structure.

Send Recovery Scope: DataInterchange allows you to specify how much processing should take place before DataInterchange makes a request to the underlying system to commit all resources. When a COMMIT request is issued, all database changes become permanent.

If a system or program failure occurs, changes made to the database since the last COMMIT are removed by the file system. The guarantee of database integrity is only available for the DB2 version of DataInterchange or the VSAM version when running within the CICS environment with the Transaction Store files defined as recoverable.

The amount of processing before a COMMIT request is issued is called the recovery scope. The translator allows a transaction level recovery scope or an interchange level recovery scope. The level desired is indicated in the **SCOPE** field of the TRCB. The following lists valid values for the **SCOPE** field:

Value	Description
-------	-------------

E	This value indicates interchange level recovery is needed. DataInterchange does not issue a database COMMIT request until an interchange is complete and has been written to the file associated with the network.
---	--

This value is valid only when enveloping operations are taking place. When an interchange recovery scope is used, use the **INMEMTRANS** field to increase concurrency. The **INMEMTRANS** field specifies the maximum number of transactions that should be maintained in virtual storage before any database updates are attempted. Database updating occurs when **INMEMTRANS** or the end of the interchange is reached. This value is important in an environment where multiple application programs are requesting translation services concurrently. The higher the value for **INMEMTRANS**, the more concurrency achieved. See "Translator Control Block (TRCB)" on page A-6 for the description of **INMEMTRANS** and virtual storage considerations.

Do not use an interchange recovery scope if an interchange is to contain a large number of transactions. If the number of transactions exceeds INMEMTRANS, other processes are locked out until the entire interchange is complete. Also, DB2 has a limit on the number of locks that a process can hold. A large number of transactions could cause this value to be exceeded.

T	This value indicates the recovery scope should be at the transaction level. DataInterchange issues a database COMMIT request at the end of every successful transaction.
---	--

T is a suggested value, but any value other than E is interpreted like a T.

Regardless of the value of the SCOPE, when the translator determines it is time for a COMMIT request to be issued, the COMMIT is issued before the translator returns control to the application. If the application requires that the database changes that it makes be synchronized with database changes that DataInterchange makes, you can tell the translator not to issue a COMMIT by setting the **NOCOMIT** flag to Y. The application can then make database updates based on the results of the translation and call the translator with the issue-database-commit function code (991). For more information, see "API - Issue Commit" on page 3-98.

The translator issues the necessary call to commit all resources that have been updated. It is important that you use the 991 function code rather than the application program requesting the COMMIT request on its own (either directly to DB2 and/or CICS or by using the SYNCPOINT service). The translator needs to know that the COMMIT has taken place. A COMMIT request issued by the application without the knowledge of the translator could result in a deadlock situation.

Note: Setting the **NOCOMMIT** flag to Y prevents the translator from issuing commits. However, NOCOMMIT does not prevent DataInterchange termination from issuing a commit. A syncpoint service call with syncpoint interval set to -1 made prior to the DataInterchange termination call will prevent this terminate commit. For more information, see “SYNCPOINT Services” on page 3-132.

Forced Interchange Termination: If enveloping occurs at the same time as translation, the translator attempts to build the largest interchange possible. The current interchange is completed and a new one started only when certain fields change in value. For more information and a complete list of items that cause a new interchange, see the “Fields Causing New Interchange” on page 3-83.

If your application wants to control the size or content of an interchange, it is possible to use the close-and-queue- current-interchange function code 990. This function request should be issued between the last call for the current transaction and the first call for the next transaction. For more information, see “API - Close and Queue Interchange” on page 3-84.

The **ESIZE** field contains the current size of the interchange. This value can be compared to a threshold value that, once exceeded, signals your program to issue the request to close the current interchange. When an interchange is closed, a group trailer (if groups are being used) and an interchange trailer segment are added to the end of the interchange. Your threshold value should take the expected sizes of these segments into account. You can control the size of the interchange only when using the application programming interface.

If you are using C and D records, you can tell the DataInterchange Utility to force the termination of an interchange by using the Z1 record. However, you cannot limit the size of an interchange to a specific value using the DataInterchange Utility or facility.

Batches and Bundles: You can associate transactions by using the **BATCHID** field or by using the **BNDLFLAG** field. There is no relationship between these two fields, and the transaction associations formed by each are different.

BATCHID is an informal association provided as a convenience. You can use it as a fast way to select transactions from the Transaction Store. There is an alternate index on the **BATCHID** value and is, therefore, the best field on which to base a selection if you do not use the **THANDLE**. You can use this field to isolate all the transactions from a particular program execution.

If a problem occurs with the execution, the **BATCHID** can be used to place all the transactions from that execution on HOLD to prevent them from being enveloped until the problem is resolved. **BATCHID** can also be used to reenvelope transactions from a certain program execution if the file containing the interchanges was corrupted.

Whatever the use, the **BATCHID** association between transactions is informal and differs little from the set of transactions created using any common value, such as trading partner nickname or standard ID.

BNDLFLAG creates a formal relationship between transactions. The transactions associated with **BNDLFLAG** are clustered and, usually, are treated as a single unit.

In UNTDI transactions, bundles are necessary. With all other standards, transactions are independent of each other and can be handled independently. For example, X12 defines a header segment, detail segments, and trailer segment within a transaction, and UNTDI defines a header transaction, detail transaction, and a trailer transaction. These transactions are not independent. You cannot send the header transaction in one interchange, and the details and trailer transactions in another interchange. They must all be sent together in the same interchange and should not include any additional transactions in that interchange. By bundling, you tell DataInterchange which header belongs with which detail and trailer transactions. DataInterchange can then envelope the appropriate transaction separately from the

other transactions. Therefore, multiple transactions can be treated as a single unit. Bundling is required for UNTDI but is optional for the other standards. Your application can use bundling if you want to isolate a group of transactions and have them processed as a single unit rather than as individual transactions.

Any action on any member of the cluster is done to all members of the cluster. During enveloping operations, clustered transactions are not placed in the same interchange as transactions that are not clustered. One cluster is not put in the same interchange as another cluster. Members in a cluster can have different **BATCHID** values, and the same **BATCHID** value can be used in more than one cluster.

Outbound Incremental Translation

Typically, during outbound translation, all the application data being translated is read into memory before translation begins. This can be a disadvantage if application files are large. With incremental translation, application data is passed into DataInterchange in small chunks, and these chunks can then be translated and released from memory. This minimizes application data memory usage.

Incremental translation requires application data to be grouped as header, detail, and trailer. These three groups are rigid and data from one cannot be mapped into another. Incremental translation is only suitable where there is iterative detail data; for example, line items. Conceptually, this would mean: one header, many line items, and one trailer. During incremental translation, header data would be passed into DataInterchange first and then translated. Following this, one or more line items would be passed into DataInterchange and then translated. This step of passing in line items and then translating them can be repeated as many times as necessary. Finally, the trailer would be passed into DataInterchange and translated.

The first step in getting ready to incrementally translate data is to make sure the application data is grouped as described above. The next step is to make sure the DataInterchange map being used maintains this grouping so that data from one group is not mapped into another. Then an &BOUNDARY special literal must be placed in the map. The &BOUNDARY must be mapped on the data element just prior to the main detail loop mapping and should not be mapped with an application field name. If the data element is already mapped to an application field name, simply repeat the mapping with &BOUNDARY. After this is done, make sure the control string is re-generated.

The final step in getting ready to incrementally translate data is to enable the API program to process the data as described next. See the Translation Services section of this chapter for complete details on how to translate using an API. What follows is simply an addendum with special considerations for EJECT and BOUNDARY in order to facilitate incremental translation. With incremental translation, the header records are first passed into DataInterchange, each with EJECT space and BOUNDARY X. After the last header record is passed in, a subsequent call should be made with EJECT Y and BOUNDARY H. This tells DataInterchange to translate the header and then remove the header application data from memory. This last call should be made without data.

This next step is iterative. One or more detail records may be passed into DataInterchange, each with EJECT space and BOUNDARY X. When translation of these detail records is desired, a subsequent call should be made with EJECT Y and BOUNDARY D. This tells DataInterchange to translate the detail records passed in and then remove those application records from memory. This last call should be made without data. This process of passing in detail records and then translating them can occur as many times as necessary.

Finally, after all the detail records have been passed in and translated, all the trailer records should be passed into DataInterchange, each with EJECT space and BOUNDARY X. After the last trailer record is passed in, a subsequent call should be made with EJECT Y and BOUNDARY T. This tells DataInterchange to translate the trailer and then remove the trailer application data from memory. At this point, translation is complete and if enveloping is not delayed, an envelope will be created and queued.

The general rule for mapping &BOUNDARY is as follows: if there are loops within the header, &BOUNDARY can be mapped anywhere after the beginning of the last level-one header loop and before the beginning of the detail loop. Otherwise, if there are no loops within the header, &BOUNDARY can be mapped anywhere in the header.

&BOUNDARY acts as a switch that tells the control string generator to place a special end-of-header marker at the beginning of the next level-one loop. &BOUNDARY should never be mapped except in maps used for incremental translation. Nor should incremental translation ever be attempted with maps that do not contain a properly mapped &BOUNDARY.

Pageable Translation

- | Pageable Translation is designed to better utilize memory during translation. It does this by paging EDI and application data buffers to DASD once the allotted number of internal buffers has been exhausted.
- | The buffer size is 28,632 bytes, and the allotted number of internal buffers before Pageable Translation begins is 1,000. This means that approximately 28 MB of virtual storage can be used to hold EDI and application data before Pageable Translation is triggered.
- | To use Pageable Translation in MVS, a temporary work file must be defined with ddname EDIVAX. The amount of space allocated to this file is dependent on the maximum amount of data to be translated.
- | The amount of storage required to translate an envelope without Pageable Translation can be calculated by adding the following components:
 - | Number of bytes of largest interchange +
 - | Number of bytes of largest application transaction image +
 - | 4 MB overhead +
 - | Number of structures in largest interchange x 120 bytes
- | Pageable Translation deals with the first two components, and ensures that the amount of virtual storage required for them does not exceed 28 MB. The other components are not addressed by Pageable Translation.
- | The maximum amount of data that DataInterchange can page with Pageable Translation is approximately one gigabyte (specifically, 32,000 multiplied by 28,632 bytes).
- | Pageable Translation in CICS uses temporary storage queues with names that begin with EDI. Therefore, the size of DFHTEMP may have to be considered if Pageable Translation is desired in CICS.
- | To enable Pageable Translation via the API, set the VAXFLAG field in the TRCB to X. To enable Pageable Translation via the DataInterchange Utility, use the PAGE(Y) option on TRANSLATE commands.
- | A sample of how EDIVAX might be defined follows:

```
//EDIFAX DD DISP=(NEW,DELETE,DELETE),UNIT=SYSDA,SPACE=(CYL,25)
```

Translate to Application

This section describes the details of the translate-to-application API request. Translate to application is the act of taking data in the standard defined format (defined by using *EDI standards* from the Administrator's Menu (MP01)) and transforming that data into the application defined format (defined by using the *Application data format* for the Administrator's Menu (MP01)) as defined by a map (which is defined using *Trading partner transactions* from the Administrator's Menu MP01)).

This is the same API that the DataInterchange Utility uses internally when you issue any of the following commands:

- PERFORM TRANSLATE TO APPLICATION
- PERFORM DEENVELOPE AND TRANSLATE
- PERFORM RECEIVE AND TRANSLATE
- PERFORM RETRANSLATE TO APPLICATION

Receiving and Deenveloping

You can use the following API functions to receive a file, deenvelope the interchanges within the file received, and translate the standard data into an application format for processing by various application programs:

- Communication services with a function code of 232 receives the data. See “API - Receive and Restart Receive” on page 3-117.
- Enveloping/Deenveloping services with a function code of 214 deenvelopes the data. See “Deenvelope Function” on page 3-86.
- Transaction services with a function code of 213 translates the data. See “API - Translate Specific” on page 3-49.

The communication service invokes the network and receives the data from the network into the specified file. The deenveloping function performs the following:

- Parses the interchanges within the file.
- Extracts each transaction.
- Places the details of each transaction along with the transaction image into the Transaction Store.
- Generates a functional acknowledgment, if one was requested.
- Reconciles any received acknowledgments with the original transactions.

The translation service retrieves the transaction image from the Transaction Store and translates the data into the application format. You can combine the deenvelope and translate functions into one step using a call to transaction services with a function code of 212. For more information, see “API - Translate File” on page 3-57.

The shortened sequence occurs when the DataInterchange Utility PERFORM DEENVELOPE AND TRANSLATE command is used. This sequence is illustrated in Figure 3-3 on page 3-48.

The following issues are involved with these functions:

- The communications receive function invokes a network program to receive data into a file. The data placed in this file is under the control of the network program. DataInterchange counts the bytes in the interchanges within the file so that the management reporting component of DataInterchange can be updated. No information from the interchanges is recorded in the Transaction Store.
- The deenvelope function generates the functional acknowledgment for the data received and reconciles the received functional acknowledgments with the original transactions. This is true whether the deenvelope function works through code 214 (deenvelope only) or 212 (deenvelope and translate).
- Because deenveloping and translation can be separate functions, there must be a way to tell the translator which transaction from the Transaction Store should be processed. You do this through the unique key value assigned to the transaction when it is added to the store. The program that is deenveloping the data must remember the key value, called a transaction handle (THANDLE).

Otherwise, a list of key values must be retrieved from the Transaction Store using the `PERFORM QUERY` command.

- No single API function is the equivalent of the DataInterchange Utility `PERFORM RECEIVE AND TRANSLATE` command. Using the API, receiving the data, and deenveloping and translating the data are two separate steps.

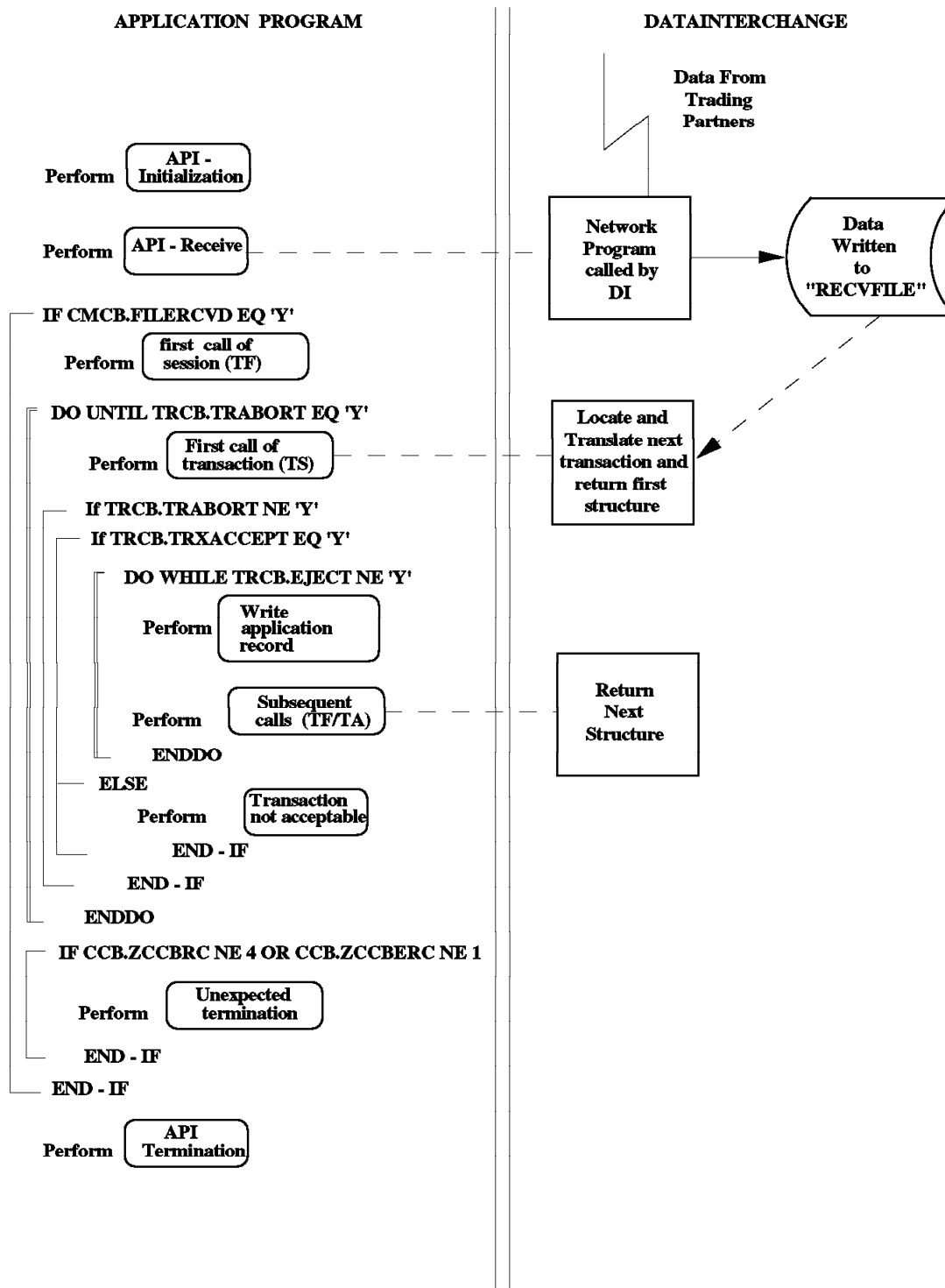


Figure 3-3. Receive, Develope, and Translate

Test Translate to Application

There are two methods to test your transactions:

1. The test function code translates to application format in test mode (function code 211). In test mode, your program follows the same steps as for normal work. Interchanges are read from a file and parsed. Transactions are translated and presented to your program one transaction at a time.

However, no information is written to the Transaction Store, and functional acknowledgments are neither generated nor reconciled. If you want Transaction Store services, or if you want a functional acknowledgment returned to your trading partner, you must use the test method outlined in the second option.

If you are using the test function code, DataInterchange assumes that you are interested in using a test usage (if one is available) and, therefore, automatically forces test mode. You can verify the results of your program's efforts by examining the application data produced, as well as the information that was recorded in the event log. To view or print the event log, choose event logging from the Administrator's Menu.

2. The trading partner sends the data with the test indicator turned on in the interchange header segment if the envelope type being used has a test indicator. When you use the test indicator, translation and functional acknowledgment processing occurs as usual (including the updating of the Transaction Store database).

The interchange that contains the functional acknowledgment carries an indication that the data is for testing purposes. You may prefer this method, rather than option 1, because the entire path (from your trading partner to your program, and back to your trading partner) with functional acknowledgments can be tested.

API - Translate to Application

The following describes the functions of the APIs provided for the translate-to-application function:

API	Description
Translate Specific	The API translates a specific transaction into application format. You can use this API when a transaction already exists within the Transaction Store and you want to translate it for the first time or translate it again. The transaction is placed in the store through a deenvelope operation, or a deenvelope and translate operation already performed.
Translate File	The API deenvelopes and translates all the transactions within a file. Using this API, all the interchanges within a file are processed and all transactions within the interchanges are deenveloped, translated, and added to the Transaction Store.

These APIs are described in detail in the following sections.

API - Translate Specific

The basic format of the API request to translate a specific transaction from standard format to application format is:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter	Description						
SNB	The service name block identifying translation services.						
	<table> <tr> <th>Field</th><th>Value</th></tr> <tr> <td>ZSNBNAME</td><td>TRANPROC (identifies translation services)</td></tr> <tr> <td>ZSNBPC</td><td>6 (number of parameters in the call)</td></tr> </table>	Field	Value	ZSNBNAME	TRANPROC (identifies translation services)	ZSNBPC	6 (number of parameters in the call)
Field	Value						
ZSNBNAME	TRANPROC (identifies translation services)						
ZSNBPC	6 (number of parameters in the call)						
CCB	The common control block used to initialize DataInterchange.						
FCB	The function control block with a ZFCBFUNC value of 213.						
TRCB	The translator control block. See “Translator Control Block (TRCB)” on page A-6 for details about this block.						
TRIDB	The input data block. The input data block is used by DataInterchange as a work buffer. This buffer must be at least 32000 bytes in length. Its maximum size should be the size of the maximum standard segment (excluding the BIN segment) that is received. See “Translator Input Data Block (TRIDB)” on page A-29 for a general description of this block.						
TRODB	The output data block. This block contains the application data produced. The format of the DATA field in this block matches the format of the data defined in the application data format. The block has a minimum size of 32000. See “Translator Output Data Block (TRODB)” on page A-31 for a general description of this block.						

In the remainder of this section, the phrase *calling the translator* means *Invoking the DataInterchange API with a request to translate a specific transaction to application format*. When reference to a field within the TRCB, the field is uppercase and highlighted (for example, **EJECT**).

First Call of Session (TA)

The values for many of the fields in the control blocks defined for the translate to application API function need only be established once. These values can be established before the first call. It is not necessary to refresh or change them before another call. The following tables illustrate the control blocks, the fields within the control blocks, and the initialization considerations.

Table 3-22. Service Name Block (SNB) Initialization for Translate-to-Application

SNB	Initialization Value
ZSNBLL	32
ZSNBNAME	TRANPROC (the service name for translation services)
ZSNBPC	6 (all calls for translation services have six parameters)

Table 3-23. Function Code Block (FCB) Initialization for Translate to Application

FCB	Initialization Value
ZFCBLL	4
ZFCBFUNC	213

Table 3-24 (Page 1 of 2). Translator Control Block (TRCB) Initialization for Translate-to-Application

TRCB	Initialization Value
BLKLEN	1536 (the length of the translator control block in Version 2)
BLKNME	EDITRCB

Table 3-24 (Page 2 of 2). Translator Control Block (TRCB) Initialization for Translate-to-Application

TRCB	Initialization Value
BLKTYPE	There are two possible formats for the TRIDB and TRODB buffer. One has a buffer size limited to 32768. The other has no limit. A value of H indicates you are using the unlimited format. Any other value indicates you are using the format with a limit of 32768.
XPANDED	Y
BATCHID	<p>A value that can be associated with a transaction and used later to retrieve the transaction quickly. The database has an alternate key value using the batch ID. Besides using the transaction handle itself, this is the quickest way to retrieve a transaction from the database during the selection process. If you do not provide a value for the BATCHID field, the default value DDHHMMSS is used, where DD is the current day of the month, and HHMMSSL is the current time.</p> <p>Note: This field is checked with each call to translator and if a value is not supplied, the default value is used. The value that exists when the first call of transaction (TA) is made will be the value that is associated with the transaction.</p>
ERRFILTER	Values that indicate which errors should be filtered out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more detailed information, see "Error Filtering" on page 1-4.
MAPCHAIN	This flag indicates whether mapchaining is in effect. A value of Y indicates that the CURRENT transaction will be translated again (using the value in the variable DIMAPCHAIN to select the map) rather than the NEXT transaction being translated. Any other value causes the NEXT transaction to be translated.
FORCETEST	Indicates whether the translate process is to be forced to select only a test usage. If FORCETEST(Y) was used on the Deenvelope process, it also must be used on the Translate to Application to select those transactions stored by the Deenvelope.

Table 3-25. Translator Input Data Block (TRIDB) Initialization for Translate-to-Application

TRIDB	Initialization Value
BLKLEN	Set to the size of the data block, including the BLKLEN field. The minimum value for this field is 32000 bytes.
RESERVED	Binary zeros

Table 3-26. Translator Output Data Block (TRODB) Initialization for Translate-to-Application

TRODB	Initialization Value
BLKLEN	Set to the size of the data block, including the BLKLEN field. The minimum value for this field is 32000 bytes.
RESERVED	Binary zeros

First Call of Transaction (TA)

Table 3-27 describes the first call for transaction initialization values for the TRCB fields. These fields should be set before the first call for a transaction is made:

Table 3-27. Initialization - First Call for Transaction

TRCB	Initialization Value
TSKEY	The transaction handle for the transaction you want to translate. The format of the handle is YYYYMMDDHHMSSxxnnnn formatted as a packed field within 10 bytes. If your program does not handle packed values, initialize this field to all blanks or all binary zeros and use the TSKEYU field.
TSKEYU	The transaction handle for the transaction you want to translate. The format of the handle is YYYYMMDDHHMSSxxnnnn formatted as a 20-byte character field. Use this field only if the TSKEY field does not have a value.
RAWDATA	Use a value of Y if you want the translator to automatically fill in the values for the internal trading partner ID and the record ID before returning the structure to your application.

Table 3-28 describes the first call for transaction initialization values for the TRODB fields. These fields should be set before the first call for a transaction is made:

Table 3-28. Initialization - First Call for Transaction

TRODB	Initialization Value
BLKLEN	The length of the TRODB including this field. If you are using the same TRODB for all calls, this field needs to be initialized only once. It has a minimum size of 32000.
RESERVED	Set this field to zeros. If you are using the same TRODB for all calls, this field needs to be initialized only once.

When initialization is complete, the translator is invoked with the following API request:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

On the first call to the translator, the transaction identified by the **TSKEY** or **TSKEYU** field is retrieved from the Transaction Store along with the interchange, group, and transaction header images. The interchange sender ID and sender qualifier are extracted from the interchange header. These values are used to determine the trading partner who sent the data. See “Locate EDI and Application Trading Partners” on page 3-96 for information used to determine the trading partner.

The transaction/message ID value is extracted from the transaction header. At this point, the translator attempts to locate a trading partner transaction usage and map that provides the instructions for translating the transaction from the standard format into an application format. The mapping and the usages must have been established by the EDI Administrator before the API requests are made. See “Locate Receive Map” on page 3-97 for a description of the fields used and the search to locate the mapping.

Numerous errors can occur at this point. See “Translation Return Codes” on page B-19 for a list of all possible errors from the translator. If an error occurs, the return code and extended return code for that error are posted in the **ZCCBRC** and **ZCCBERC** fields of the CCB. The following fields within the TRCB reflect the status of the transaction and the status of the translator:

Field	Description
TRXACCEPT	<p>This field has a value of Y if the transaction had an acceptable translation. A value other than Y indicates that something needed to process this transaction is missing and, therefore, translation cannot occur, or that there were errors in translation that were not acceptable according to the error level established by the EDI Administrator in the usage.</p> <p>If the value is not Y and your program continues, the next call to the translator is considered the first call for the next transaction.</p>
TRABORT	<p>This field has a value of Y if the error is so severe that the translator does not continue processing.</p> <p>If the value is Y and your program continues after such an error, the next call you make to the translator is considered the first call for the session.</p>

Transaction TRCB Fields (TA): The tables in this section show the fields returned on the first call for a transaction. Such a large quantity of information is returned on the first call for a translate to application request that multiple tables are used to show the TRCB fields returned. See the note at the beginning of each table for a description of the data in the table.

Table 3-29 shows fields that provide some basic attributes of the transaction just processed, and provide information about the transaction side of the processing rather than the application side.

Table 3-29. Transaction Attribute Fields in the Translator Control Block (TRCB)

TRCB	Description
TRNID	The standard transaction or message ID for the transaction/message received.
TEST	A value of T if the interchange header indicates test. A value of P if the interchange header indicates production. A value of I if the interchange header indicates information.
MAPKEY	Identifies the trading partner transaction used to translate the standard data.
TPNICK	The trading partner nickname associated with the transaction.
DUPTRAN	Y if the transaction is a duplicate. This indicates that the interchange being received has been received before.
TRXACCEPT	Y if the transaction had an acceptable translation and indicates that application data has been returned in the TRODB, unless the EJECT field has a value of Y.
TRABORT	Y if an error occurred that was so severe that the translator did not continue. If this flag is set, the translator has terminated of its own accord.
TSKEY	The key value assigned to the transaction within the Transaction Store. This is called the transaction handle and has a format of YYYYMMDDHHMMSSxxnnnn. It is returned as a 10-byte packed value in this field, which is how it is stored in the database.
TSKEYU	The key value assigned to the transaction within the Transaction Store. It has the same value as the TSKEY field, except TSKEYU is an unpacked (character) 20-byte value.
EJECT	Y if the end of the transaction has been reached. No data is returned with an EJECT value of Y. It indicates that the end of the transaction has been reached and the next call is treated as the first call for a transaction. A value other than Y indicates that more data remains for the current transaction and that the next call will return the next structure or provide an EJECT value of Y. For more information, see the “Partial Structures (TA)” on page 3-68.
ERRNUM	The total number of errors flagged in translating the data.
ERRCDES	An array of the first 10 different errors flagged in translating the data. Each possible error detected by the translator has a unique error code. See Table A-5 on page A-28 for more information.

Table 3-30 on page 3-54 shows the fields that are application-related and provides information about the application side of the processing rather than the transaction side.

Table 3-30. Application Related Fields in Translator Control Block (TRCB)

TRCB	Description																
ATFID	The application data format definition ID associated with this transaction.																
ATSID	The name of the structure being returned on this call. The data related to the structure is in the TRODB.																
APPPFILE	The ddname for the application file to which the application data should be written. This field is taken from the application data format definition, but can be overridden in the trading partner transaction receive usage record.																
APTYPE	<p>The type of file represented by APPPFILE. This field is for CICS only (except MQ, which is supported in both MVS and CICS). Valid file types are:</p> <table> <tr> <th>Code</th><th>Description</th></tr> <tr> <td>MQ</td><td>DataInterchange MQSeries Queue profile member name</td></tr> <tr> <td>TD</td><td>Transient data queue name identified by the first four characters of APPPFILE</td></tr> <tr> <td>TM</td><td>Temporary storage queue (main) identified by APPPFILE</td></tr> <tr> <td>TS</td><td>Temporary storage queue (auxiliary) identified by APPPFILE</td></tr> <tr> <td>VS</td><td>ESDS VSAM file identified by APPPFILE</td></tr> <tr> <td>TX</td><td>CICS transaction code identified by the first four characters of APPPFILE</td></tr> <tr> <td>PG</td><td>Program identified by APPPFILE</td></tr> </table> <p>If the keyword is not provided for MVS, this field is ignored; instead, the default for the corresponding file keyword is the ddname of a sequential file. For CICS, the default is TS.</p>	Code	Description	MQ	DataInterchange MQSeries Queue profile member name	TD	Transient data queue name identified by the first four characters of APPPFILE	TM	Temporary storage queue (main) identified by APPPFILE	TS	Temporary storage queue (auxiliary) identified by APPPFILE	VS	ESDS VSAM file identified by APPPFILE	TX	CICS transaction code identified by the first four characters of APPPFILE	PG	Program identified by APPPFILE
Code	Description																
MQ	DataInterchange MQSeries Queue profile member name																
TD	Transient data queue name identified by the first four characters of APPPFILE																
TM	Temporary storage queue (main) identified by APPPFILE																
TS	Temporary storage queue (auxiliary) identified by APPPFILE																
VS	ESDS VSAM file identified by APPPFILE																
TX	CICS transaction code identified by the first four characters of APPPFILE																
PG	Program identified by APPPFILE																
INTPID	The internal trading partner ID (vendor number) taken from the trading partner transaction receive usage.																
APPCTLNUM	See XACFIELD .																
XACFIELD	The application control value. The application control value is defined by the application field with an AC data type or by the fields that were assigned when the trading partner transaction (map) was created. This is the value that uniquely identifies the transaction to the application.																
RAWDATA	If raw data processing was requested by setting the RAWDATA field to Y on the first request, RAWDATA is set on output to indicate if raw data processing occurred. If raw data processing occurred, RAWDATA has a value of Y. If raw data processing did not occur because the application data format did not have raw data specifications, RAWDATA has a value of N. Raw data processing consists of the translator setting the record ID values and the internal trading partner ID field value. The format of data written to the application file is under the control of the application program. All the necessary data for C and D records is available, but need not be used if RAWDATA output is requested.																

The translator output data block contains the application data, and for each call to the translator, a single structure (record) of application data is returned. The structure returned is indicated by the **ATSID** field of the TRCB. Data is returned only if the transaction was acceptable (**TRXACCEPT** field value of Y) and this is not the end of the transaction (EJECT value is not Y). See “Partial Structures (TA)” on page 3-68 for special considerations.

Table 3-31. Application Data

TRODB	Description
DATALEN	The number of characters of data being returned in the DATA field.
DATA	The application data with a format defined by the application data format definition for the structure (ATSID field).

Table 3-32 contains fields that are concerned with the interchange to which the current transaction belongs.

Table 3-32. Interchange Header/Trailer Fields in the Translator Control Block (TRCB)

TRCB	Description												
QTPNICK	The trading partner nickname for the last transaction processed.												
ENVTYPE	Indicates the envelope type of the interchange to which the transaction belongs. The following are the possible values: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>E</td><td>UNB/UNZ</td></tr> <tr> <td>I</td><td>ICS/ICE</td></tr> <tr> <td>T</td><td>STX/END</td></tr> <tr> <td>U</td><td>BG/EG</td></tr> <tr> <td>X</td><td>ISA/IEA</td></tr> </table>	Value	Description	E	UNB/UNZ	I	ICS/ICE	T	STX/END	U	BG/EG	X	ISA/IEA
Value	Description												
E	UNB/UNZ												
I	ICS/ICE												
T	STX/END												
U	BG/EG												
X	ISA/IEA												
IHCTL	See IHXCTL .												
IHXCTL	The interchange control number assigned to the current interchange. The value is returned in this field as a right-justified value with leading zeros.												
ISYNTAXID	The interchange syntax ID for E and T envelopes.												
ISYNTAXVER	The interchange syntax version for E and T envelopes.												
ISIDQUAL	The interchange sender ID qualifier for E, I, and X envelopes.												
ISID	The interchange sender ID (interchange header field with an IS data type).												
ISENDNAME	The interchange sender name for U and T envelopes.												
IREVROUT	The interchange reverse routing for E envelopes.												
IRIDQUAL	The interchange receiver ID qualifier for E, I, and X envelopes.												
IRID	The interchange receiver ID (interchange header field with an IR data type).												
RECVNAME	The interchange receiver name for U and T envelopes.												
IRouteADDR	The interchange routing address for E envelopes.												
IDATE	The interchange date (interchange header field with a DT data type).												
ITIME	The interchange time (interchange header field with a TM data type).												
IVERREL	The interchange version and release (interchange header field with a VR or LV data type).												
ISPW	The interchange password (interchange header field with a PW data type).												
IAPREF	The application reference (interchange header field with a AP data type).												
ISTDID	The interchange standard ID for I and X envelopes.												
IPRIOR	The interchange priority code for E and T envelopes.												
ICOMMAGREE	The interchange communication agreement E and T envelopes.												

Table 3-33 contains fields involved with the group to which the current transaction belongs.

Table 3-33 (Page 1 of 2). Group Header/Trailer Fields in the Translator Control Block (TRCB)

TRCB	Description
GHCTL	See GHXCTL .
GHXCTL	The group control number assigned to the current group. The value is returned in this field as a right-justified value with leading zeros.
GSIDQUAL	The group sender ID qualifier for E envelope groups.

Table 3-33 (Page 2 of 2). Group Header/Trailer Fields in the Translator Control Block (TRCB)

TRCB	Description
GSID	The group sender ID (group header field with a data type of AS).
GRID	The group receiver ID (group header field with a data type of AR).
GRIDQUAL	The group receiver ID qualifier for E envelope groups.
GDATE	The group date (group header field with a data type of DT).
GTIME	The group time (group header field with a data type of TM).
GAPW	The group password (group header field with a data type of PW).
GVER	The group version (group header field with a data type of VR).
GREL	The group release (group header field with a data type of LV).
GRESAGENCY	The group responsible agency code for E, U, and X envelopes.

Table 3-34 contains fields that are involved with the transaction header for the current transaction.

Table 3-34. Transaction Header/Trailer Fields in the Translator Control Block (TRCB)

TRCB	Description
NEWTRN	Y if translation was acceptable. If a copy of the transaction header is wanted, a function code value of 3 can be used to obtain an exact image of the transaction header.
THCTL	See THXCTL .
THXCTL	The transaction control number assigned to the current transaction. The value is returned in this field as a right-justified value with leading zeros.
TTC	The current transaction or message ID value. See also TRNID .
TVER	The transaction version (transaction header field with a data type of VR).
TREL	The transaction release (transaction header field with a data type of LV).

Subsequent Calls (TF/TA)

The items listed below summarize all of the activity that took place on the first call of a transaction (TA).

1. The next transaction in the file is located or the specific transaction is retrieved from the Transaction Store.
2. The map associated with the transaction is found.
3. The translation from standard format into application structures takes place.
4. The application structures are sorted into the order defined by the application data format definition.
5. The first application structure is returned to the calling program within the TRODB.

If the first call of a transaction was successful, the **TRXACCEPT** field has a value of Y, and the application continues to call the translator with the following call (the parameters are the same as those on the first call of a transaction):

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The application continues to call the translator until the **EJECT** field has a value of Y. For each call made, the next application structure is returned in the TRODB and the name of the structure is placed in the **ATSID** field.

Last Call of Transaction (TA)

The translator signals that the last call for a transaction has just been made by setting the **EJECT** field to a value of Y. No data accompanies this call. It signals the end of the current transaction. At this point, your application might do application-related processing relative to the end of the transaction, including updating the application databases. The next call is interpreted as the first call for the next transaction. The translator issues a COMMIT request with the next call.

Last Call of Session (TA)

When all the specific transactions have been processed, the translator should be called one last time with an end-translation function code. For more information, see “API - End Translation/Enveloping” on page 3-85.

API - Translate File

You can use two function codes to request a translate-file-to-application format function. Function code 212 requests a translation in production mode, and function code 211 is used to request a translation in test mode.

Before Release 4 of DataInterchange, there were considerable differences in your program's operation using function codes 212 and 211. Function code 211 allowed you to test your mapping, but did not help in testing your application program. The switch from function 211 to 212 required program changes.

Using this release of DataInterchange, you do not have to change your program to switch from test to production mode. You only need to change the function code used to invoke the services. The following lists the differences between the test and production modes:

- DataInterchange assumes that the test indicator is set in the interchange, even when it is not present, or when it indicates production data. This forces the use of a test mapping, if one exists.
- DataInterchange automatically logs the standard image received and the application data produced.
- DataInterchange does not write any data to the Transaction Store to prevent cluttering up the store with test data.
- Functional acknowledgments are not generated even if requested in the usage record.
- Statistics maintained by the Management Reporting component of DataInterchange are not updated.

The basic format of the API request to translate to application format is:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter	Description
-----------	-------------

SNB	The service name block identifying translation services.
-----	--

Field	Value
ZSNBNAME	TRANPROC (identifies translation services)
ZSNBPC	6 (number of parameters in the call)

CCB	The common control block used to initialize DataInterchange.
-----	--

FCB	The function control block with a ZFCBFUNC value of 212 to indicate a production program or 211 to indicate a test program.
-----	--

TRCB	The translator control block. For more information, see “Translator Control Block (TRCB)” on page A-6.
TRIDB	The input data block. The input data block is used by DataInterchange as a work buffer. This buffer must be at least 32000 bytes in length. Its maximum size should be the size of the maximum standard segment (excluding the BIN segment) that is received. See “Translator Input Data Block (TRIDB)” on page A-29 for a general description of this block.
TRODB	The output data block. This block contains the application data that has been produced. The format of the data in the DATA field of this block matches the format of the data defined in the Application Data Format. The block has a minimum length of 32000. See “Translator Output Data Block (TRODB)” on page A-31 for a general description of this block.

In the remainder of this section, the phrase *calling the translator* means *Invoking the DataInterchange API with a request to translate file*. When a reference is made to a field within the TRCB, the field is uppercase and highlighted (for example, **EJECT**).

First Call of Session (TF)

The values of numerous fields in the control blocks defined for the translate to application API function need to be established only once. The values can be established before the first call. It is not necessary to refresh or change them before any other call. The following tables illustrate the control blocks, the fields within the control blocks, and the initialization considerations.

Table 3-35. Service Name Block (SNB) Initialization for Translate-to-Application

SNB	Initialization Value
ZSNBLL	32
ZSNBNAME	TRANPROC (the service name for translation services)
ZSNBPC	6 (all calls for translation services have six parameters)

Table 3-36. Function Code Block (FCB) Initialization for Translate-to-Application

FCB	Initialization Value
ZFCBLL	4
ZFCBFUNC	212 (for a production program) 211 (for a test program)

Table 3-37 (Page 1 of 3). Translator Control Block (TRCB) Initialization for Translate-to-Application

TRCB	Initialization Value
BLKLEN	1536 (the length of the translator control block in Version 2)
BLKNME	EDITRCB
BLKTYPE	There are two possible formats for the TRIDB and TRODB buffer. One format has a buffer size limited to 32768 while the other format has no limit. A value of H indicates you are using the unlimited format. Any other value indicates you are using the format with a limit of 32768.
XPANDED	Y
ENVLDELAY	This field indicates if enveloping of functional acknowledgments should occur. A value of Y indicates that enveloping should not occur. Any other value indicates enveloping should occur.

Table 3-37 (Page 2 of 3). Translator Control Block (TRCB) Initialization for Translate-to-Application

TRCB	Initialization Value						
DUPTRAN	<p>Indicates how duplicate interchanges should be processed. DUPTRAN is only an input field on the first call of a session and establishes if duplicate interchanges are errors. On all other requests, DUPTRAN is an output field. The following are valid values:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>N</td><td>Indicates that duplicate interchanges should not be processed but should be considered as errors. If a duplicate interchange is received, the translator will issue message TR0211 and return to the application with an interchange level error (extended return code value of 5).</td></tr> <tr> <td>Y</td><td>Indicates that duplicate interchanges should be processed and transactions should be returned. Transactions are flagged as duplicate transactions. Y is a suggested value, but any value other than N is interpreted as Y.</td></tr> </table>	Value	Description	N	Indicates that duplicate interchanges should not be processed but should be considered as errors. If a duplicate interchange is received, the translator will issue message TR0211 and return to the application with an interchange level error (extended return code value of 5).	Y	Indicates that duplicate interchanges should be processed and transactions should be returned. Transactions are flagged as duplicate transactions. Y is a suggested value, but any value other than N is interpreted as Y.
Value	Description						
N	Indicates that duplicate interchanges should not be processed but should be considered as errors. If a duplicate interchange is received, the translator will issue message TR0211 and return to the application with an interchange level error (extended return code value of 5).						
Y	Indicates that duplicate interchanges should be processed and transactions should be returned. Transactions are flagged as duplicate transactions. Y is a suggested value, but any value other than N is interpreted as Y.						
SCOPE	<p>A value of E indicates that interchange level recovery is required. Using E, DataInterchange does not issue a database COMMIT until an interchange is complete. Any other value indicates that the recovery scope should be at the transaction level. DataInterchange then issues a database COMMIT at the start of every transaction.</p> <p>Note: This field contains values that affect the recovery scope during the session. See "Send Recovery Scope" on page 3-42 for more information about this field.</p>						
INMEMTRANS	<p>This parameter applies only if the value of SCOPE is E. It indicates the maximum number of transactions that should be maintained in virtual storage before any database updates are attempted. Database updating occurs when INMEMTRANS or the end of the envelope is reached.</p> <p>This value is important only in an environment where multiple application programs are requesting translation services concurrently. The higher the value for INMEMTRANS, the greater the concurrency achieved. See "Translator Control Block (TRCB)" on page A-6 for a more detailed description of this field.</p> <p>Note: This field contains values that affect the recovery scope during the session. See "Send Recovery Scope" on page 3-42 for more information about this field.</p>						
FILEID	The ddname of the file that contains the interchanges that should be processed. This is an optional field. If the field value is provided, the REQID is ignored.						
REQID	The requestor ID identifies a member in the requestor profile (REQPROF). The <i>Receive file name</i> field in the requestor profile has the name of the file that contains the interchanges to be processed. This field is required only if FILEID is not provided.						
MRREQID	If the interchanges being deenveloped have not been recorded in the Management Reporting statistics database, the MRREQID identifies the requestor ID for which statistics should be updated. This is necessary only if the interchanges were not received using the communications service receive function.						
FUNACKFLE	If functional acknowledgments are being enveloped, FUNACKFLE is the ddname of the file to which the transaction data is written when the interchange is complete. This field is optional. If not provided, the ddname specified in the <i>Trans data queue</i> field of the network profile (NETPROF) is used.						
BATCHID	<p>The batch ID is a value that can be associated with a transaction and used later to retrieve the transaction quickly. The database has an alternate key value using the batch ID. The database has an alternate key value using the batch ID. Besides using the transaction handle itself, this is the quickest way to retrieve a transaction from the database during the selection process. If you do not provide a value for the BATCHID field, the default value DDHHMMSS is used, where DD is the current day of the month, and HHMMSSL is the current time.</p> <p>Note: This field is checked with each call to the translator and if a value is not supplied, the default value is used. The value that exists when the first call for transaction (TF) is made will be the value that is associated with the transaction.</p>						

Table 3-37 (Page 3 of 3). Translator Control Block (TRCB) Initialization for Translate-to-Application

TRCB	Initialization Value
TRXLIFE	<p>The amount of time the transaction remains in the Transaction Store before it is eligible to be purged. If a value is not supplied, it takes on the default value of 30 days.</p> <p>Note: This field is checked with each call to the translator and if a value is not supplied, the default value is used. The value that exists when the first call for transaction (TF) is made will be the value associated with the transaction.</p>
IMGLIFE	<p>The amount of time the transactions image (that is, the standard data produced) remains in the Transaction Store. If you do not supply a value, it takes on the default value of 30 days.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. This field is checked with each call to the translator and if a value is not supplied, the default value is used. The value that exists when the first call for transaction (TF) is made will be the value associated with the transaction. 2. This field is not currently supported.
ERRFILTER	<p>Values that indicate which errors should be filtered out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more information, see "Error Filtering" on page 1-4.</p>
MAPCHAIN	<p>This flag indicates whether mapchaining is in effect. A value of Y indicates that the CURRENT transaction will be translated again (using the value in the variable DIMAPCHAIN to select the map) rather than the NEXT transaction being translated. Any other value causes the NEXT transaction to be translated.</p>
FORCETEST	<p>Indicates whether the translate process is to be forced to select only a test usage. If FORCETEST(Y) was used on the Deenvelope process, then it also must be used on the Translate to Application to select those transactions stored by the Deenvelope.</p>

Table 3-38. Transaction Input Data Block (TRIDB) Initialization for Translate-to-Application

TRIDB	Initialization Value
BLKLEN	<p>Set to the size of the data block, including the BLKLEN field. The minimum value for this field is 32000 bytes.</p>
RESERVED	<p>Binary zeros</p>

Table 3-39. Translator Output Data Block (TRODB) Initialization for Translate-to-Application

TRODB	Initialization Value
BLKLEN	<p>Set to the size of the data block, including the BLKLEN field. The minimum value for this field is 32000 bytes.</p>
RESERVED	<p>Binary zeros</p>

Because the first call for a session also qualifies as the first call for a transaction, follow the instructions in "First Call of Transaction (TF)" on page 3-61.

First Call of Transaction (TF)

The following tables show the fields you should set before the first call for a transaction:

Table 3-40. Initialization - First Call for Transaction - TRCB

TRCB	Initialization Value
RAWDATA	Use a value of Y if you want the translator to automatically fill in the values for the internal trading partner ID and the record ID before it returns the structure to your application.
HOLDFLAG	Use a value of Y if the transaction is to be placed in a hold status when added to the Transaction Store. A transaction in hold status is not available for any other activity until it is released. A value of N is suggested if the transaction is not to be held, but any value other than Y is treated like N.

Table 3-41. Initialization - First Call for Transaction - TRODB

TRODB	Initialization Value
BLKLEN	The length of the TRODB, including this field. If you are using the same TRODB for all calls, this field needs to be initialized only once.
RESERVED	Set this field to zeros. If you are using the same TRODB for all calls, this field needs to be initialized only once.

When initialization is complete, the translator is invoked with the following API request:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

On the first call to the translator for the session, the file containing the interchanges (identified by the **FILEID** or **REQID** field) is opened and the first interchange within the file is located. The entire interchange is read into virtual storage and validated. The interchange sender ID and sender qualifier are extracted from the interchange header. These values are used to determine the trading partner that sent the data. See “Locate EDI and Application Trading Partners” on page 3-96 for an explanation of the search for a trading partner.

On the first call to the translator for a transaction, the next transaction within the current file is located (which may involve finding the next interchange) and the transaction/message ID value is extracted from the transaction header. At this point, the translator attempts to locate a trading partner transaction usage and map that provides the instructions for translating the transaction from the standard format into an application format. The mapping and the usages must have been established by the EDI Administrator before the API requests are made. See “Locate Receive Map” on page 3-97 for a description of the fields used and the search to locate a mapping.

Numerous errors can occur at this point. See “Translation Return Codes” on page B-19 for a description of all the possible errors from the translator. If an error occurs, the return code and extended return code for that error are posted in the CCB’s **ZCCBRC** and **ZCCBERC** fields.

The following fields within the TRCB should be checked, because they reflect the status of the transaction and the status of the translator:

Field	Description
TRXACCEPT	<p>This field has a value of Y if the transaction had an acceptable translation. A value other than Y indicates that something needed to process this transaction is missing and, therefore, translation cannot occur, or that there were errors in translation that were not acceptable according to the error level established by the EDI Administrator in the usage.</p> <p>If the value is not Y and your program continues, the next call to the translator is considered the first call for the next transaction.</p>
TRABORT	<p>This field has a value of Y, if the error is so severe that the translator does not continue.</p> <p>If the value is Y and your program continues after such an error, the next call you make to the translator is considered the first call for the session. If the FILEID or REQID fields are not changed, continuation of the program can result in an infinite loop, because the same file is processed again from the beginning. This would result in the same error, unless the environment changes between the first and subsequent attempts.</p>

Transaction TRCB Fields (TF): The tables in this section show the fields returned on the first call for a transaction. Such a large quantity of information is returned on the first call for a translate-to-application request that multiple tables are used to show the TRCB fields returned. See the note at the beginning of each table for a description of the data in the table.

Table 3-42 shows the fields that provide information on basic attributes of the transaction. They supply data about the transaction side of the processing rather than the application side.

Table 3-42 (Page 1 of 2). Transaction Attribute Fields in TRCB

TRCB	Description
TRNID	The standard transaction or message ID for the transaction/message received
TEST	A value of T if the interchange header indicates test. A value of P if the interchange header indicates production. A value of I if the interchange header indicates information.
MAPKEY	Identifies the trading partner transaction used to translate the standard data
TPNICK	The trading partner nickname associated with the transaction.
DUPTRAN	Y if this transaction is a duplicate. It indicates that the interchange being received has been received before.
TRXACCEPT	Y if the transaction had an acceptable translation. It indicates that application data has been returned in the TRODB, unless the EJECT field has a value of Y.
TRABORT	Y, if an error occurred that was so severe that the translator did not continue. If this flag is set, the translator terminated of its own accord.
TSKEY	The key value assigned to the transaction within the Transaction Store. This is called the transaction handle and has a format of YYYYMMDDHHMMSSxxnnnn. It is returned as a 10-byte packed value in this field, which is how it is stored in the database.
TSKEYU	The key value assigned to the transaction within the Transaction Store. This is the same value as the TSKEY field, except TSKEYU is an unpacked (character) 20-byte value.
EJECT	<p>Y, if the end of the transaction has been reached. There is no data returned with an EJECT value of Y. It indicates that the end of the transaction has been reached. The next call is treated as the first call for a transaction.</p> <p>A value other than Y indicates that more data remains for the current transaction and that the next call returns the next structure or an EJECT value of Y. See "Partial Structures (TA)" on page 3-68 for related considerations.</p>
ERRNUM	The total number of errors flagged during the translation of the data.

Table 3-42 (Page 2 of 2). Transaction Attribute Fields in TRCB

TRCB	Description
ERRCDES	An array of the first 10 different errors flagged during the translation of the data. Each possible error detected by the translator has a unique error code. For more information, see Table A-5 on page A-28.

Table 3-43 shows the fields that are involved in the application side of the processing rather than the transaction side.

Table 3-43. Application Related Fields in TRCB

TRCB	Description																
ATFID	The application data format definition ID associated with this transaction																
ATSID	The name of the structure being returned on this call. The data related to the structure is in the TRODB.																
APPFIL	The ddname name for the application file to which the application data should be written. This field is taken from the application data format definition but can be overridden in the trading partner transaction receive usage record.																
APPTYPE	<p>The type of file represented by APPFIL. This field is for CICS only (except MQ, which is supported in both MVS and CICS). Valid file types are:</p> <table> <tr> <th>Code</th><th>Description</th></tr> <tr> <td>MQ</td><td>DataInterchange MQSeries Queue profile member name</td></tr> <tr> <td>TD</td><td>Transient data queue name identified by the first four characters of APPFIL</td></tr> <tr> <td>TM</td><td>Temporary storage queue (main) identified by APPFIL</td></tr> <tr> <td>TS</td><td>Temporary storage queue (auxiliary) identified by APPFIL</td></tr> <tr> <td>VS</td><td>ESDS VSAM file identified by APPFIL</td></tr> <tr> <td>TX</td><td>CICS transaction code identified by the first four characters of APPFIL</td></tr> <tr> <td>PG</td><td>The program identified by APPFIL</td></tr> </table> <p>If the keyword is not provided for MVS, this field is ignored; instead, the default for the corresponding file keyword is the ddname of a sequential file. For CICS, the default is TS.</p>	Code	Description	MQ	DataInterchange MQSeries Queue profile member name	TD	Transient data queue name identified by the first four characters of APPFIL	TM	Temporary storage queue (main) identified by APPFIL	TS	Temporary storage queue (auxiliary) identified by APPFIL	VS	ESDS VSAM file identified by APPFIL	TX	CICS transaction code identified by the first four characters of APPFIL	PG	The program identified by APPFIL
Code	Description																
MQ	DataInterchange MQSeries Queue profile member name																
TD	Transient data queue name identified by the first four characters of APPFIL																
TM	Temporary storage queue (main) identified by APPFIL																
TS	Temporary storage queue (auxiliary) identified by APPFIL																
VS	ESDS VSAM file identified by APPFIL																
TX	CICS transaction code identified by the first four characters of APPFIL																
PG	The program identified by APPFIL																
INTPID	The internal trading partner ID (vendor number) taken from the trading partner transaction receive usage																
APPCTNUM	See XACFIELD .																
XACFIELD	The application control value. The application control value is defined by the application field with an AC data type or by the fields that were assigned when the trading partner transaction (map) was created. It is the value that uniquely identifies the transaction to the application.																
RAWDATA	<p>If raw data processing was requested by setting the RAWDATA field to Y on the first request, RAWDATA is set on output to indicate if raw data processing occurred. If raw data processing occurred, RAWDATA has a value of Y. If raw data processing did not occur because the application data format did not have raw data specifications, RAWDATA has a value of N.</p> <p>Raw data processing consists of the translator setting the record ID values and the internal trading partner ID field value. The format of data written to the application file by the application program is under the control of the application program. All the necessary data for C and D records is available, but need not be used if raw data output is requested.</p>																

Table 3-44 on page 3-64 describes the TRODB application data.

The translator output data block contains the application data. For each call to the translator, a single structure (record) of application data is returned. The structure returned is indicated by the **ATSID** field of the TRCB. Data is returned only if the transaction was acceptable (**TRXACCEPT** value of Y) and the

transaction is not complete (EJECT value is equal Y). See “Partial Structures (TA)” on page 3-68 for special considerations.

Table 3-44. Application Data

TRODB	Description
DATALEN	The number of characters of data returned in the DATA field.
DATA	The application data with a format as defined by the application data format definition for the structure (ATSID field).

Table 3-45 describes the TRCB acknowledgment related fields.

Table 3-45. Fields Related to Functional Acknowledgments

TRCB	Description																
FABUILT	<p>This field is set equal to the envelope type for the functional acknowledgment. The possible values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>E</td><td>UNB/UNZ</td></tr> <tr> <td>I</td><td>ICS/ICE</td></tr> <tr> <td>T</td><td>STX/END</td></tr> <tr> <td>U</td><td>BG/EG</td></tr> <tr> <td>X</td><td>ISA/IEA</td></tr> <tr> <td>G</td><td>Indicates the acknowledgment does not have an interchange header</td></tr> <tr> <td>S</td><td>Indicates the acknowledgment was written to the Transaction Store, but was not enveloped</td></tr> </table>	Value	Description	E	UNB/UNZ	I	ICS/ICE	T	STX/END	U	BG/EG	X	ISA/IEA	G	Indicates the acknowledgment does not have an interchange header	S	Indicates the acknowledgment was written to the Transaction Store, but was not enveloped
Value	Description																
E	UNB/UNZ																
I	ICS/ICE																
T	STX/END																
U	BG/EG																
X	ISA/IEA																
G	Indicates the acknowledgment does not have an interchange header																
S	Indicates the acknowledgment was written to the Transaction Store, but was not enveloped																
FARC	The return code from the translator for the functional acknowledgment translation done during deenvelope. This return code can be found in <i>DataInterchange Messages and Codes</i> .																
FAERC	The extended return code from the translator for the functional acknowledgment translation done during deenvelope. This return code can be found in <i>DataInterchange Messages and Codes</i> .																

Table 3-46 shows the fields involved with the interchange to which the current transaction belongs. These fields are established when the first transaction for the interchange is processed. They remain constant for all the transactions within this interchange.

Table 3-46 (Page 1 of 3). Interchange Header/Trailer Fields in TRCB

TRCB	Description
DSNAME	The physical data set name from which transactions are processed.
QTPNICK	The trading partner nickname for which the last transaction was processed.
QSIZE	The total number of bytes that were in the interchange, stored as a 4-byte binary value.
QBT	Character representation of the QSIZE field.
ENVTYPE	Indicates the envelope type received. The following are valid values:
IHCTL	See IHXCTL .
IHXCTL	The interchange control number assigned to the current interchange. This value is returned in this field as a right-justified value with leading zeros.
ISYNTAXID	The interchange syntax ID for E and T envelopes.
ISYNTAXVER	The interchange syntax version for E and T envelopes.
ISIDQUAL	The interchange sender ID qualifier for E, I, and X envelopes.

Table 3-46 (Page 2 of 3). Interchange Header/Trailer Fields in TRCB

TRCB	Description
ISID	The interchange sender ID (interchange header field with IS data type).
ISENDNAME	The interchange sender name for U and T envelopes.
IREVROUT	The interchange reverse routing for E envelopes.
IRIDQUAL	The interchange receiver ID qualifier for E, I, and X envelopes.
IRID	The interchange receiver ID (interchange header field with IR data type).
RECVNAME	The interchange receiver name for U and T envelopes.
IRouteADDR	The interchange routing address for E envelopes.
IDATE	The interchange date (interchange header field with DT data type).
ITIME	The interchange time (interchange header field with TM data type).
IVERREL	The interchange version and release (interchange header field with VR or LV data type).
ISPW	The interchange password (interchange header field with PW data type).
IAPREF	The application reference (interchange header field with AP data type).
ISTDID	The interchange standard ID for I and X envelopes
IPRIOR	The interchange priority code for E and T envelopes.
ICOMMAGREE	The interchange communication agreement E and T envelopes.
NEWENV	Y if the transaction is the first transaction of an interchange. If a copy of the interchange header is required, the function code value of 1 can be used to obtain an exact image of the interchange header. For more information, see “API - Retrieve Interchange Header” on page 3-99.
GRPNUM	<p>The total number of groups processed so far within the current interchange, stored as a 4-byte binary value. See IGT.</p> <p>Note: The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.</p>
TRNNUM	<p>The total number of transactions processed so far within the current interchange, stored as a 4-byte binary value. See ITT.</p> <p>Note: The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.</p>
SEGNUM	<p>The total number of segments processed so far within the current interchange, stored as a 4-byte binary value. See IST.</p> <p>Note: The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.</p>
ESIZE	<p>The total number of bytes processed so far within the current interchange, stored as a 4-byte binary value. See IBT.</p> <p>Note: The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.</p>
IGT	<p>Character representation of GRPNUM.</p> <p>Note: The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.</p>
ITT	<p>Character representation of TRNNUM.</p> <p>Note: The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.</p>

Table 3-46 (Page 3 of 3). Interchange Header/Trailer Fields in TRCB

TRCB	Description
IST	Character representation of SEGNUM . Note: The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.
IBT	Character representation of ESIZE . Note: The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.

Table 3-47 contains fields that are involved in the group to which the current transaction belongs.

Table 3-47 (Page 1 of 2). Group Header/Trailer Fields in the TRCB

TRCB	Description
GHCTL	See GHXCTL .
GHXCTL	The group control number assigned to the current group. The value is returned in this field as a right-justified value with leading zeros. Note: This field value is established when the first transaction for the group is processed.
GSIDQUAL	The group sender ID qualifier for E envelope groups.
GSID	The group sender ID (group header field with AS data type). Note: This field value is established when the first transaction for the group is processed.
GRID	The group receiver ID (group header field with AR data type). Note: This field value is established when the first transaction for the group is processed.
GRIDQUAL	The group receiver ID qualifier for E envelope groups.
GDATE	The group date (group header field with DT data type). Note: This field value is established when the first transaction for the group is processed.
GTIME	The group time (group header field with TM data type). Note: This field value is established when the first transaction for the group is processed.
GAPW	The group password (group header field with PW data type). Note: This field value is established when the first transaction for the group is processed.
GVER	The group version (group header field with VR data type). Note: This field value is established when the first transaction for the group is processed.
GREL	The group release (group header field with LV data type). Note: This field value is established when the first transaction for the group is processed.
GRESAGENCY	The group responsible agency code for E, U, and X envelopes.
NEWGRP	Y if the transaction is the first transaction of an interchange. If a copy of the group header is required, a function code value of 2 can be used to obtain an exact image of the group header. For more information, see "API - Retrieve Group Header" on page 3-100.
TRNGRP	The total number of transactions processed so far within the current group, stored as a 4-byte binary value. See GTT. Note: This field value changes as transactions within the group are processed, indicating how much of the group has been processed.

Table 3-47 (Page 2 of 2). Group Header/Trailer Fields in the TRCB

TRCB	Description
GTT	Character representation of TRNGRP . Note: This field value changes as transactions within the group are processed, indicating how much of the group has been processed.

Table 3-48 shows the fields involved with the transaction header for the current transaction.

Table 3-48. Transaction Header/Trailer Fields in the TRCB

TRCB	Description
NEWTRN	Y if a transaction header is available. If a copy of the transaction header is required, a function code value of 3 can be used to obtain an exact image of the transaction header. For more information, see "API - Retrieve Transaction Header" on page 3-100.
LASTINENV	Y if this transaction is the last transaction within the current interchange.
THCTL	See THXCTL .
THXCTL	The transaction control number assigned to the current transaction. The value is returned in this field as a right-justified value with leading zeros.
SEGTRN	The total number of segments within the current transaction, stored as a 4-byte binary value. See TST .
TTC	The current transaction or message ID value. See TRNID .
TVER	The transaction version (transaction header field with VR data type).
TREL	The transaction release (transaction header field with LV data type).
TST	Character representation of SEGTRN .

Subsequent Calls (TF/TA)

The items listed below summarize all the activity that took place on the first call of a transaction (TF).

1. The next transaction in the file is located or the specific transaction is retrieved from the Transaction Store.
2. The map associated with the transaction is found.
3. The translation from standard format into application structures takes place.
4. The application structures are sorted into the order defined by the application data format definition.
5. The first application structure is returned to the calling program within the TRODB.

If the first call of a transaction was successful, the **TRXACCEPT** field has a value of Y and the application continues to call the translator until the **EJECT** field has a value of Y. The application uses the following call:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The parameters on this call are exactly the same as those on the first call of a transaction. For each call made, the next application structure is returned in the **TRODB** field. The name of the structure is placed in the **ATSID** field.

Last Call of Transaction (TF)

The translator signals that the last call for a transaction has just been made by setting the **EJECT** field to a value of Y. No data accompanies this call. It signals the end of the current transaction. The next call is interpreted as the first call for the next transaction.

At this point, your application might perform application related processing relative to the end of the transaction, including updating of the application databases. If there is a transaction recovery scope (**SCOPE** value other than E), the translator issues a COMMIT request on the next call.

If this transaction is the last transaction for an interchange, this is indicated in the **LASTINENV** field with a value of Y.

Last Call of Session (TF)

The translator signals the last call for a session through the CCB return codes. A **ZCCBRC** value of 4 and a **ZCCBERC** value of 1 indicate that all the data from the current file has been processed and that the translator has terminated. If there is another file that your application can process, cycle back to the first call of the session and set the **REQID** or **FILEID** field to indicate the next file to be processed.

Special Considerations (TA)

The following sections list special considerations.

Partial Structures (TA)

After a transaction has been translated, the application data produced is returned to the application program in the TRODB. The data is returned one structure at a time (that is, one structure for each API request made).

When all the data for a transaction has been returned, the next request results in an **EJECT** field value of Y, which indicates that the transaction is complete. The minimum size of the TRODB is 32000 bytes, but it can be as large as you like. If a structure exceeds the size of the TRODB, as much data as can fit in the TRODB is returned and the **EJECT** field has a value of X.

The **REQSIZE** field contains the total structure size. When the **EJECT** field has a value of X, it should be left as an X. The next call to the translator returns the remainder of the structure. The translator continues to return a value of X in the **EJECT** field until the entire structure has been returned to the application. The **EJECT** field is set to D.

If you do not want the remainder of the structure, set the value of the **EJECT** field to blank. The translator returns data for the next structure, rather than the residual data for the current structure.

Single Unit of Work (TA)

After a transaction has been translated, the application data produced is returned to the application program in the TRODB. In most cases, the data is returned one structure at a time (that is, one structure for each API request made). The end of the data is signaled when the translator sets the **EJECT** field to Y.

As each structure is returned, the name of the structure is returned in the **ATSID** field. However, this changes if the application data has been defined as only one structure, or if multiple structures are defined as part of their parent structure (and are not passed separately). This is called single unit work mode, and means that all the application data is provided in a single structure. With single unit of work, all the data is

returned on the first request and an EJECT value of Y is not provided. Single unit of work mode can be detected because the **ATSID** field contains all blanks and the **EJECT** field has a value of D.

It is possible to have a partial structure returned in single unit of work mode as well. In this case, the **EJECT** field has a value of X until all the data for the structure has been returned.

Clustered Transactions (TA)

During a translate-to-standard operation, the application has control over clustering transactions by setting the **BNDLFLAG** field in the TRCB. Clustering is a way of creating associations between transactions so that any operation that occurs to any member of the cluster occurs to all members of the cluster.

During translate to application or deenvelope requests, there is no control provided for clustering transactions as they are taken from the interchange. DataInterchange automatically clusters all the transactions for a UNTDI interchange (STX/END interchange header/trailer), but transactions from any other interchange type are not clustered.

If the DataInterchange utilities are used to deenvelope and translate a UNTDI interchange, and if any transaction in the interchange fails to translate successfully, no data for any transaction in that interchange is returned to the application. The all or nothing aspect of processing UNTDI interchanges is up to the application requesting the translate file function. The DataInterchange Utility will discard all data in a bundle if there is an error for any transaction within the bundle. However, because data for each transaction in the bundle will be returned by the translator, your application may perform differently.

DataInterchange also forces an interchange recovery scope (**SCOPE** value of E) when a UNTDI interchange is being processed. The interchange recovery scope is forced by the translator and cannot be changed by the application program.

Enveloping Services

Before you can send a transaction to a trading partner, and before a trading partner can send a transaction to you, the transaction must go through an envelope operation. The enveloping is necessary so networks can identify the trading partner who should receive the data and so the translators can identify and verify the type of data being received. The following are the three layers of enveloping:

- Interchange layer
- Group layer
- Transaction layer

Each layer contains a header segment and a trailer segment. The data within these segments:

- Identifies the trading partners involved (both sender and receiver)
- Identifies the applications or departments within the trading partner organizations involved (both sending and receiving)
- Identifies the exact nature of the data that is being exchanged

Each segment has fields that help the receiving translator verify that a complete and consistent interchange has been received. These fields consist of the following:

- A control number in the header and trailer that must match.
- A control count that indicates the number of items that should be present at the next level. For example, a number in the interchange trailer indicates how many groups should be present; a number

in the group trailer indicates how many transactions should be present; and a number in the transaction trailer indicates how many standard segments should be in the transaction.

It is the sender's responsibility to set these values based on the interchange created and the receiver's responsibility to verify that the data received agrees with the information contained in the header and trailer segments. These enveloping header and trailer segments are called *control* or *service* segments. Just as there are standards for transactions, there are standards for the service segments.

DataInterchange supports the following enveloping standards:

- EDIFACT envelope standard - envelope and profile ID is E

Segment ID	Purpose
UNB	Interchange header
UNG	Group header
UNH	Transaction header
UNT	Transaction trailer
UNE	Group trailer
UNZ	Interchange trailer
UNA	Delimiter segment

- ICS envelope standard - envelope and profile ID is I

Segment ID	Purpose
ICS	Interchange header
GS	Group header
ST	Transaction header
SE	Transaction trailer
GE	Group trailer
ICE	Interchange trailer

- UNTDI envelope standard - envelope and profile ID is T

Segment ID	Purpose
STX	Interchange header
BAT	Group header
MHD	Transaction header
MTR	Transaction trailer
EOB	Group trailer
END	Interchange trailer
SCH	Delimiter segment

Note: BAT and EOB are accepted by DataInterchange during receive processing but are never generated by DataInterchange.

- UCS envelope standard - envelope and profile ID is U

Segment ID	Purpose
BG	Interchange header
GS	Group header
ST	Transaction header
SE	Transaction trailer
GE	Group trailer
EG	Interchange trailer

- X12 envelope standard - envelope and profile ID is X

Segment ID	Purpose
ISA	Interchange header
GS	Group header
ST	Transaction header

SE	Transaction trailer
GE	Group trailer
IEA	Interchange trailer

Each layer of enveloping has a specific purpose as described in the following sections.

Interchange Layer

Networks use the interchange header to identify the trading partner that is sending the data, and the trading partner that should receive the data. The network uses the sender ID for routing any error messages or network acknowledgments for the interchange. The interchange header is used by DataInterchange on the receiving end to perform the following:

- Determine the trading partner, which in turn is used to determine the map needed for translation.
- Determine the destination for any functional acknowledgments that are generated.

A control number in the interchange header uniquely identifies this interchange between the sender and receiver. The control number in the interchange trailer must match the control number in the header. A count field in the interchange trailer contains the number of groups and transactions within the interchange.

The interchange layer is optional in DataInterchange when GS/GE segments are used at the group level. If the interchange layer is not present, the GS02 and GS03 values identify the trading partners involved. GS02 and GS03 must contain the trading partner nickname of the sender and receiver respectively.

Group Layer

The group layer is not required for all enveloping standards. It is an optional layer for EDIFACT and UNTDI, but it is a mandatory layer for the X12, UCS, and ICS standards. The layer is optional in some standards and mandatory in other standards because of differences in how functional acknowledgments are defined in the various standards. For the X12, UCS, and ICS standards, the functional acknowledgments (997 or 999 transactions) are defined for the group layer. In these standards it is the GROUP that is being acknowledged. However, for EDIFACT and UNTDI, the acknowledgment (CONTRL message) is defined at the interchange level, and it is the INTERCHANGE that is being acknowledged. As a result, the group layer becomes optional.

The group layer identifies the sending and receiving applications or divisions within the trading partner organization. As a result, all transactions and messages that have a similar purpose or destination can be grouped together, then routed to the proper application based on the values within the group header.

DataInterchange uses the application sender and application receiver ID values from the group header to locate the map used to translate a transaction within the group. For more information, see “Locate Receive Map” on page 3-97. The group header contains the sending and receiving IDs. It also contains a control number that must be unique within the interchange. However, if 999 or 997 functional acknowledgments are being generated, the group control number must be unique for the interchange sender or receiver combination. In this case, the group control number serves the same purpose as the interchange control number. This is one reason interchanges using GS and GE as the functional group header and trailer can be sent without the interchange level of enveloping (this is not supported by all networks, but some do allow omission of interchange level when GS and GE functional groups are being used). The group trailer contains a control number that must match the control number in the group header, and a count field that contains the number of transactions within the group.

Transaction Layer

The transaction layer identifies the transaction that immediately follows. The transaction ID (810, 850, 856, and so on) or message ID (ORDERS, INVOICE, CONTRL, and so on) is in the transaction header. This is the final piece used by DataInterchange to locate a map for translating the transaction. The transaction header also contains a control number that must be unique within the group if groups are being used, and unique within the interchange if groups are not being used. The transaction trailer also contains a control number that must match the control number in the header. The trailer also contains a count field that defines the number of segments within the transaction. This number must match the actual number of segments received.

Enveloping Service

The enveloping service provides the following functions:

- An enveloping function

The enveloping function extracts transactions from the Transaction Store and builds transaction headers and trailers, organizing the transactions into groups, and the groups into interchanges. The enveloping function adds the EDIVTSEV, EDIVTSGP and EDIVTSTU records to the Transaction Store database.

- A deenveloping function

The deenveloping function locates interchanges within a file, parses the interchange into groups and transactions, and adds the transactions to the Transaction Store. The deenveloping function also adds the EDIVTSEV, EDIVTSGP and EDIVTSTU records to the Transaction Store database.

Each of these functions is described in detail in the following sections.

Envelope Function

This section describes the details of the envelope application program interface (API) request. Enveloping is the act of extracting transactions from the Transaction Store and building the transaction, group, and interchange headers and trailers that are necessary to send those transactions to the appropriate trading partners.

Note: Enveloping does not need to be a separate API request. It is possible to envelope transactions when they are translated. For more information, see the “Enveloping and Sending” on page 3-22 and “API - Translate to Standard” on page 3-27.

The API that is described below is the same API that the DataInterchange Utility uses internally when you issue any of the following commands:

- PERFORM ENVELOPE
- PERFORM ENVELOPE AND SEND
- PERFORM REENVELOPE
- PERFORM REENVELOPE AND SEND

The enveloping API is invoked once for each transaction that should be enveloped. As each transaction is received, the enveloping function checks the current transaction to determine whether it can be put into the current group and current interchange. If the current transaction does not belong in the current group, the current group is closed (a trailer is built) and a new group is started (a header is built). See “Fields Causing New Group” on page 3-83 for an explanation of which fields cause a new group.

If the current transaction does not belong in the current interchange, the current group and interchange are closed (trailers are built) and written to a file associated with the network, and a new interchange and group are started (headers are built). See “Fields Causing New Interchange” on page 3-83 for an explanation of which fields cause a new interchange.

When the DataInterchange utilities and facilities use the envelope API, the transactions to be enveloped (dictated by the selection criteria) are first sorted to yield the fewest possible interchanges and the fewest groups within interchanges.

Your application program using the API must consider that without a sort of the transactions to be enveloped it would be possible to generate an interchange for each transaction. For example, if there are 10 transactions, 5 for trading partner **A** and 5 for trading partner **B** and the sequence were **A,B,A,B,A,B,A,B,A,B** then 10 interchanges would be created. Sorting the above transactions would yield 2 interchanges rather than 10.

An application using the envelope API must obtain a list of the transactions that should be enveloped. Some possible methods to obtain a transaction list are:

- The PERFORM QUERY command could be used to select transactions from the Transaction Store. The transaction handles for the selected transactions are written to the EDIQUERY file and are in transaction handle sequence. This would not result in optimal enveloping.
- The application program that is adding the transactions to the Transaction Store could save the transaction handle values (and any other information needed for sorting) to a file that is then sorted and read by your enveloping application.
- The PERFORM TRANSACTION DATA EXTRACT command could be used to select transactions from the Transaction Store. As with PERFORM QUERY, the transactions are written to the EDIQUERY file and are in transaction handle sequence, but DATA EXTRACT provides complete transaction information that could be used to sort the transactions into a optimal sequence for enveloping.

API - Envelope

The basic format of the API request to envelope a transaction is:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter Description

SNB	The service name block identifying translation services.	
	Field	Value
	ZSNBNAME	TRANPROC to identify translation services
	ZSNBPC	6 (number of parameters in the call)
CCB	The common control block used to initialize DataInterchange.	
FCB	The function control block with a ZFCBFUNC value of 215.	
TRCB	The translator control block. For more information on the control block, see “Translator Control Block (TRCB)” on page A-6. For more information on the INMEMTRANS and virtual storage considerations, see the “Translator Control Block (TRCB)” on page A-6.	
TRIDB	The input data block. This block is not used during enveloping operations, but a parameter must be provided. A minimum size of 16 bytes is sufficient during an enveloping operation. For more information on this block, see “Translator Input Data Block (TRIDB)” on page A-29.	

TRODB The output data block. The output data block is used by DataInterchange as a work buffer. This buffer must be at least 32000 bytes in length, but its maximum size must be the size of the maximum standard segment (excluding the BIN segment) that is processed. For more information on this block, see “Translator Output Data Block (TRODB)” on page A-31.

In the remainder of this section, the phrase *calling the translator* means *Invoking the DataInterchange API with a request to envelope transaction*. When a reference is made to a field within the TRCB, the field is uppercase and highlighted (for example, **EJECT**).

Initializing for Envelope API

In the control blocks defined for the enveloping API function, some field values need to be established only once. These values can be established before the first call and do not have to be refreshed or changed before any other call. Table 3-49, Table 3-50, and Table 3-51 show each of the control blocks, the fields within the control blocks, and the initialization considerations for initializing the envelope API.

Table 3-49. SNB Initialization for Enveloping Function

SNB	Initialization Value
ZSNBLL	32
ZSNBNAME	TRANPROC
ZSNBPC	6 (all calls for enveloping services have six parameters)

Table 3-50. FCB Initialization for Enveloping Function

FCB	Initialization Value
ZFCBLL	4
ZFCBFUNC	215

Table 3-51 (Page 1 of 3). TRCB Initialization for Enveloping Function

TRCB	Initialization Value
BLKLEN	1536 (the length of the translator control block in Version 2)
BLKNME	EDITRCB
BLKTYPE	There are two possible formats for the TRIDB and TRODB buffer. One format has a buffer size limited to 32768, while the other format has no limit. A value of H indicates you are using the unlimited format. Any other value indicates you are using the format with a limit of 32768.
XPANDED	Y
SCOPE	A value of E indicates interchange level recovery is wanted, and DataInterchange does not issue a database COMMIT until an interchange is complete. Any other value indicates the recovery scope should be at the transaction level, and DataInterchange issues a database COMMIT after each transaction is added to the current interchange. Note: This value affects the recovery scope during the session.

Table 3-51 (Page 2 of 3). TRCB Initialization for Enveloping Function

TRCB	Initialization Value
INMEMTRANS	<p>This parameter applies only if the value of SCOPE is E and indicates the maximum number of transactions that should be maintained in virtual storage before you attempt any database updates. The database is updated whenever INMEMTRANS or the end of the interchange is reached. This value is important in an environment where multiple application programs are requesting translation services concurrently. The higher the value for INMEMTRANS, the more concurrency is achieved. See “Translator Control Block (TRCB)” on page A-6 for a description of this field.</p> <p>Note: This value affects the recovery scope during the session.</p>
FILEID	<p>The ddname of the file where the transaction data is written when an interchange is complete. This field is optional. If this field is not provided, the ddname specified in the <i>Trans data queue</i> field of the network profile (NETPROF) is used. This value does not have to be a constant for the entire session. However, if you want to use this field, it must be set before an interchange is completed.</p>
IUSEREXIT	<p>Provides the logical name of a user exit to be called by DataInterchange instead of writing the envelope to a file. This exit is used when the envelope is to be retrieved via storage using the Get Envelope Service after enveloping is complete. For more information on the Get/Put Envelope Exit processing, see Chapter 4, “Exit Routines.”</p>
IUSERAREA	<p>A pointer to a user-defined area. The pointer is passed to the user exit defined in IUSEREXIT field.</p>
ITPBREAK	<p>A flag that indicates whether a change in the internal trading partner ID (vendor number) causes a new interchange. A value of Y indicates that each time the internal trading partner ID changes, the current interchange should be closed and a new interchange started. Any other value indicates that a new interchange should be started only when the trading partner nickname changes in value. For example, if your application has many different vendor numbers for the same trading partner, and you want each interchange sent to the trading partner to contain only data associated with one of those vendors, set the ITPBREAK value to Y. If you want all the data, regardless of the vendor number, to be put into one interchange for the trading partner, set the ITPBREAK value to N. Any value other than Y is treated like N.</p>
ENVCHK	<p>A flag that has the translator check the transaction for the status you expect for the operation being performed. Possible values are:</p> <ul style="list-style-type: none"> 1 Verifies that the transaction has never been enveloped before. If the transaction has been enveloped before, the translator will issue message TR0121 and return to the application with a transaction level error (extended return code value of 3). 2 Verifies that the transaction has been enveloped before. If the transaction has not been enveloped before, the translator will issue message TR0121 and return to the application with a transaction level error (extended return code value of 3). (other) No verification done on the transaction. Any value other than 1 or 2 is interpreted to mean that validation should not be performed. <p>See “Envelope Versus Reenvelope” on page 3-81 for more detailed information.</p>
ERRFILTER	<p>Values that indicate which errors should be filtered out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. See “Error Filtering” on page 1-4 for more detailed information.</p>
RAWDATAOUT	<p>A value of ‘Y’ indicates that RAWDATA output is desired for Fixed-to-Fixed mappings. Any other value indicates that ‘C’ and ‘D’ record output is wanted.</p>

Table 3-51 (Page 3 of 3). TRCB Initialization for Enveloping Function

TRCB	Initialization Value
FFILEID	The ddname of the file where the translated data for Fixed-to-Fixed translations is written. This field is optional. If this field is not provided, the ddname formed from the concatenation of <i>Application file name</i> from the target application data format and the <i>File suffix</i> from the trading partner profile will be used. This value does not have to be a constant for the entire session. However, if you want to use this field, it must be set before an interchange is completed.

Table 3-52. TRIDB Initialization for Enveloping Function

TRIDB	Initialization Value
BLKLEN	Set to the size of the data block, including the BLKLEN field.
RESERVED	Binary zeros.
Note: The input data block (TRIDB) is not used during an enveloping operation but must be supplied in the parameter list. A block with a BLKLEN value of 16 is sufficient.	

Table 3-53. TRODB Initialization for Enveloping Function

TRODB	Initialization Value
BLKLEN	Set to the size of the data block, including the BLKLEN field. The minimum value for this field is 32000 bytes.
RESERVED	Binary zeros.

Envelope Transaction

This section describes the initialization required before each call and the results returned to your program. The following fields must be set before the API request to envelope a transaction is made:

Table 3-54. Initialization - For Each Transaction

TRCB	Initialization Value
TSKEY	The transaction handle for the transaction you want to envelope. The format of the handle is YYYMMDDHHMSSxxnnnn, formatted as a packed field within 10 bytes. If your program does not pack these values, initialize this field to all blanks or all binary zeros using the TSKEYU field.
TSKEYU	The transaction handle for the transaction you want to envelope. The format of the handle is YYYMMDDHHMSSxxnnnn, formatted as a 20-byte character field. This field is only used if TSKEY does not have a value.

Once the initialization is complete, the enveloper is invoked with the following API request:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The transaction identified by the **TSKEY** or **TSKEYU** field is retrieved from the Transaction Store along with many other database records needed, such as the trading partner profile member and the send usage and mapping records. Everything needed to translate the transaction (except the control string) is necessary when the transaction is enveloped. Numerous errors can occur at this point. These errors are described in "Translation Return Codes" on page B-19. If an error occurs, the return code and extended return code for that error are posted in the **ZCCBRC** and **ZCCBERC** fields of the common control block

(CCB). The following fields within the translator control block (TRCB) indicate the status of the transaction and the translator:

- TRXACCEPT** Has a value of Y if this transaction is enveloped successfully. Any value other than Y indicates that something needed to process this transaction is not available or that the transaction is not eligible to be enveloped.
- TRABORT** Has a value of Y if the error is severe and the translator has stopped processing.
- If the value is Y and your program continues after such an error, the next call you make to the translator is considered the first call for the session.

The fields that are returned in the translator control blocks are explained in Table 3-55 and Table 3-56.

Transaction Related Fields (EV): Table 3-55 shows fields that provide some basic attributes of the transaction just processed. These fields are related to the transaction side of the processing.

Table 3-55. Transaction Attribute Fields in TRCB

TRCB	Description	
TRNID	The standard transaction or message ID for the transaction or message received.	
TEST	Value	Description
	P	Production transaction
	T	Test transaction
	I	Information transaction
MAPKEY	Identifies the trading partner transaction used to translate the application data.	
TPNICK	The trading partner nickname associated with the transaction.	
TRXACCEPT	Y if the transaction had an acceptable translation and indicates that application data has been returned in the translator output data block (TRODB) (unless the EJECT field has a value of Y).	
TRABORT	Y if the error is severe and the translator has stopped processing.	
TSKEY	The key value assigned to the transaction within the Transaction Store. This is called the transaction handle and has a format of YYYYMMDDHHMMSSxxnnnn. The transaction handle is returned as a 10-byte packed value in this field and is stored in this format in the database.	
TSKEYU	The key value assigned to the transaction within the Transaction Store. The same value as the TSKEY field, but TSKEYU is an unpacked (character) 20-byte value.	
ERRNUM	The total number of errors that were flagged while enveloping the transaction.	
ERRCDES	An array of the first 10 errors that were flagged in enveloping the transaction. Each possible error detected by the translator has a unique error code. These values are documented in "Translator Control Block (TRCB)" on page A-6 and "Translation Return Codes" on page B-19.	

Application Related Fields (EV): Table 3-56 shows the fields that are application-related. These fields are related to the application side of the processing.

Table 3-56 (Page 1 of 2). Application Related Fields in TRCB

TRCB	Description
ATFID	The application data format definition ID associated with this transaction.
INTPID	The internal trading partner ID (vendor number) taken from the trading partner transaction receive usage.
APPLTPID	The application trading partner is considered an internal trading partner within your business organization.

Table 3-56 (Page 2 of 2). Application Related Fields in TRCB

TRCB	Description
APPCTLNUM	See XACFIELD .
XACFIELD	The application control value defined by the application field with data type of AC or by the fields that are assigned when the trading partner transaction map is created. The application control value uniquely identifies the transaction to the application.
MAPCHAIN	When set to Y, indicates that the current transaction will be translated again rather than the next transaction being translated.

Enveloping - TRCB Fields: Some fields within the TRCB are related to the current status of an interchange that is being created. Table 3-57, Table 3-58 on page 3-79, and Table 3-59 on page 3-80 describe the fields relating to the following:

- The interchange header and trailer
- The group header and trailer
- The transaction header and trailer

Table 3-57 (Page 1 of 2). Fields in TRCB Related to Interchange Header or Trailer

TRCB	Description
NEWENV	Y if this transaction caused a new interchange to start. If a copy of the interchange header is desired, a function code value of 1 can be used to obtain an exact image of the interchange header. (See “API - Retrieve Interchange Header” on page 3-99.)
IHCTL	See IHXCTL .
IHXCTL	The interchange control number assigned to the current interchange. The value is returned in this field as a right-justified value with leading zeros.
GRPNUM	The total number of groups within the interchange at the current time. This number is stored as a 4-byte binary value (see IGT).
TRNNUM	The total number of transactions within the current interchange at the current time. This number is stored as a 4-byte binary value (see ITT).
SEGNUM	The total number of segments within the current interchange at the current time. This number is stored as a 4-byte binary value (see IST).
ESIZE	The total number of bytes within the current interchange at the current time. This number is stored as a 4-byte binary value (see IBT).
ISYNTAXID	The interchange syntax identifier used in EDIFACT and UNTDI envelopes.
ISYNTAXVER	The interchange syntax version used in EDIFACT and UNTDI envelopes.
ISIDQUAL	The interchange sender ID qualifier used in EDIFACT, ICS, and X12 envelopes.
ISID	The interchange sender ID (interchange header field with a data type of IS).
IrecvNAME	The interchange receiver name used in UNTDI envelopes. The application receiver code used in the UCS envelopes if UCS04 is not an <i>IR</i> data type.
IRouteADDR	The interchange routing address used in EDIFACT envelopes.
ISendNAME	The interchange sender name used in UNTDI envelopes. The application sender code used in the UCS envelopes if UCS03 is not an <i>IS</i> data type.
IREVROUT	The interchange reverse routing used in EDIFACT envelopes.
IRIDQUAL	The interchange receiver ID qualifier used in EDIFACT, ICS, and X12 envelopes.
IRID	The interchange receiver ID (interchange header field with a data type of IR).
IDATE	The interchange date (interchange header field with a data type of DT).

Table 3-57 (Page 2 of 2). Fields in TRCB Related to Interchange Header or Trailer

TRCB	Description
ITIME	The interchange time (interchange header field with a data type of TM).
IVERREL	The interchange version and release (interchange header field with a data type of VR or LV).
IGT	A character representation of GRPNUM .
ITT	A character representation of TRNNUM .
IST	A character representation of SEGNUM .
IBT	A character representation of ESIZE .
ISPW	The interchange password (interchange header field with a data type of PW).
IAPREF	The application reference (interchange header field with a data type of AP).
ISTDID	The interchange standard ID used in ICS and X12 envelopes.
IPRIOR	The interchange processing priority used in EDIFACT and UNTDI envelopes.
ICOMMAGREE	The interchange communication agreement used in EDIFACT envelopes.

Table 3-58. Fields in TRCB Related to Group Header or Trailer

TRCB	Description
NEWGRP	Y if this transaction caused a new group to start. If a copy of the group header is desired, a function code value of 2 can be used to obtain an exact image of the group header. For more information, see “API - Retrieve Group Header” on page 3-100.
GHCTL	See GHXCTL .
GHXCTL	The group control number assigned to the current group. The value is returned in this field as a right-justified value with leading zeros.
TRNGRP	The total number of transactions within the current group at the current time. This number is stored as a 4-byte binary value (see GTT).
GSIDQUAL	The group sender ID qualifier used in EDIFACT envelopes.
GSID	The group sender ID (group header field with a data type of AS).
GRIDQUAL	The group receiver ID qualifier used in EDIFACT envelopes.
GRID	The group receiver ID (group header field with a data type of AR).
GDATE	The group date (group header field with a data type of DT).
GTIME	The group time (group header field with a data type of TM).
GAPW	The group password (group header field with a data type of PW).
GVER	The group version (group header field with a data type of VR).
GREL	The group release (group header field with a data type of LV).
GTT	A character representation of TRNGRP .
GRESAGENCY	The group controlling agency used in EDIFACT envelopes. The group responsible agency used in ICS, UCS, and X12 envelopes.

Table 3-59. Fields in TRCB Related to Transaction Header or Trailer

TRCB	Description
NEWTRN	Y if this transaction caused a new transaction to start. If a copy of the transaction header is desired, a function code value of 3 can be used to obtain an exact image of the transaction header (see “API - Retrieve Transaction Header” on page 3-100).
THCTL	See THXCTL .
THXCTL	The transaction control number assigned to the current transaction. The value is returned in this field as a right-justified value with leading zeros.
SEGTRN	The total number of segments within the current transaction stored as a 4-byte binary value (see TST).
TTC	The current transaction or message ID value (see TRNID).
TVER	The transaction version (transaction header field with a data type of VR).
TREL	The transaction release (transaction header field with a data type of LV).
TST	A character representation of SEGTRN .

Queuing - TRCB Fields: Table 3-60 describes the fields in the translator control block returned when an interchange is written. The **EJECT** field indicates whether an interchange is written. The other fields in the table provide information about the interchange, the network, and the file receiving the interchange.

Table 3-60 (Page 1 of 2). Fields in TRCB Returned When an Interchange is Written

TRCB	Description						
EJECT	<p>If an interchange is written to the file associated with the network, the EJECT field has one of the following values:</p> <table> <tr> <th>Value</th><th>Definition</th></tr> <tr> <td>Q</td><td>The interchange is written successfully.</td></tr> <tr> <td>E</td><td>The interchange is NOT written successfully.</td></tr> </table> <p>Note: If EJECT has a value of E, indicating an error occurred, the QRC and QERC fields have values indicating the error. The specific error that occurred is logged by DataInterchange communications services. The most likely error is that the ddname could not be opened.</p>	Value	Definition	Q	The interchange is written successfully.	E	The interchange is NOT written successfully.
Value	Definition						
Q	The interchange is written successfully.						
E	The interchange is NOT written successfully.						
QRC	The return code associated with the error.						
QERC	The extended return code associated with the error.						
DSNAME	The physical data set name to which the interchange is written. In CICS, the DSNAME field contains the same value as the QDDNAME field, which is the name of the temporary storage queue containing the interchange.						
QNETID	The name of the network associated with the trading partner.						
QPTTOPT	Has a value of Y if the network identified by QNETID is a point-to-point network.						
QSRPGM	Has a value of Y if the network identified by QNETID has a network send and receive program associated with it. If no send and receive program is associated with a network, there is no need to attempt to send the interchange. A network might exist without a send and receive program for creating interchanges that get sent to a trading partner through some other means.						
QDDNAME	The ddname to which the interchange is written. This field does not have a value if a point-to-point network is being used. In CICS, this is the name of the temporary storage queue to which the interchange is written.						
QTPNICK	The trading partner nickname for which the interchange is built.						
QSIZE	The total number of bytes in the interchange. This number is stored as a 4-byte binary value.						

Table 3-60 (Page 2 of 2). Fields in TRCB Returned When an Interchange is Written

TRCB	Description
QBT	A character representation of QSIZE .

See “Sending Transaction Data” on page 3-82 for an explanation of items that must be considered if your application program is using the communications services send API for sending the data just written to the file.

Last Call of Session (EV)

If the translator is called at least once and the **TRABORT** flag does not have a value of Y, the translator should be invoked one last time with a termination function code. The termination function code signals the translator that the application is finished making enveloping requests. Upon receiving the request, the translator releases any resources that are acquired. If an interchange is active, the translator completes and writes the current interchange. See “API - End Translation/Enveloping” on page 3-85.

Special Considerations

The following sections list special considerations.

Envelope Versus Reenvelope

Both the DataInterchange Utility and Transaction Store Facility provide a means of enveloping and reenveloping transactions. When the **ENVELOPE** function is used, only transactions within the store that have never been enveloped are considered as candidate transactions. When the **REENVELOPE** function is used, only transactions within the store that are enveloped are considered as candidate transactions.

The API services do not provide a similar distinction. The application program making the API request must distinguish whether a transaction is enveloped or reenveloped. However, the **ENVCHK** field in the transaction control block can be set to indicate the intentions of the application (**ENVELOPE** or **REENVELOPE**) and the envelope service verifies that the status of the transaction matches the intentions of the application. The **ENVCHK** field has the following values and meanings:

Value Meaning

- 1 The intent is to envelope a transaction. If the transaction has ever been previously enveloped, it should be considered an error. If the transaction has been enveloped before, the translator will issue message TR0121 and return to the application with a transaction level error (extended return code value of 3).
- 2 The intent is to reenvelope a transaction. If the transaction has not been previously enveloped, it should be considered an error. If the transaction has not been enveloped before, the translator will issue message TR0121 and return to the application with a transaction level error (extended return code value of 3).
- (other) No status check should be made.

Clustered Transactions (EV)

It is possible during translation to CLUSTER transactions. The Transaction Store Facility and DataInterchange Utility ensure that any action that takes place against any member of the cluster also takes place against all members of the cluster. If any member is enveloped, all members are enveloped. If any member is purged, all members are purged. This is a feature of the utilities and facilities and not a feature of the API. The only check that the enveloping service makes is that the controlling transaction of a cluster is the first transaction that is enveloped. There is no check to ensure that all members are enveloped or that they are enveloped in any particular order, except for the requirement that the controlling transaction is first. For more information, see **BNDLFLAG** on page 3-32.

Sending Transaction Data

The **EJECT** field in the transaction control block indicates when transaction data has been written to the file associated with the network. If the send transaction data communications API function (see “API - Send Transactions and Restart Send Transactions” on page 3-109) is used to send the data, your program must determine whether the data is to be sent now or at the end of the translation or enveloping function.

Sending Transaction Data When it is Written: Sending the transaction data when it is written is the easiest method because the application program does not have to be as aware of its environment as it does if delayed sending is used. When the **EJECT** flag indicates that data is written, the **DSNAME** field contains the name of the data set to which the transaction data was written, and the **QNETID** field contains the network ID associated with the trading partner. The **QNETID** field can be moved to the CMCB **NETID** field and the **DSNAME** field can be moved to the CMCB **FILENAME** field (and set **DATATYP** to A). Communications can then be invoked to send the data.

During a DataInterchange session (between initialization and termination), the communications programs remember the names of data sets to which data has been written. The first time a data set is used, it is opened for output. Anything in the data set is then overlaid with the new transaction data. However, on subsequent use, the same data set is opened for extend, meaning the first transaction data is not overlaid, and the new transaction data is written at the end of the data set. Set the **CLRFILE** flag in the CMCB to a value of Y so the data set is cleared after the data is sent. Interchanges can be sent multiple times if the file is not cleared after being sent.

Notes:

1. You can use the **QDDNAME** if a point-to-point network is not being used. **DSNAME** works regardless of the network.
2. You must supply a requestor profile ID on the communications request, and there must be a way to associate a requestor ID with a network (if multiple networks are being used).

Sending Transaction Data Later: Deferring the sending of the data until translation or enveloping is complete is more complex than sending the data immediately. The application program must keep track of all data sets used and the networks associated with the data sets so each can be sent to the appropriate network.

The data set name should be used rather than the ddname, because multiple ddnames (that is, QDATA and QDATAE) can be associated with the same physical data set. Using the ddname can result in the data being sent twice. If the **FILEID** field is used and the sending is deferred, the application must verify that all interchanges created are for the same network. If point-to-point networks are being used, all interchanges created should be for the same trading partner.

Fields Causing New Interchange: During an enveloping operation, the current transaction is checked against the current interchange to determine whether this transaction belongs in the current interchange or whether the current interchange should be closed and a new one started. The following fields are checked for changes to see if a new interchange needs to be started:

- EDI trading partner nickname
- Application trading partner nickname
- Network ID used by the trading partner
- Envelope type
- BUNDLE status
- Any of the delimiters
- Decimal notation
- Release character
- Usage indicator
- Who is assigning transaction control numbers
- Internal trading partner ID, depending on the value of **ITPBREAK** within the transaction control block (TRCB)
- Any of the following override values in the translator control block (TRCB)
 - Interchange syntax ID (**ISYNTAXID**)
 - Interchange syntax version (**ISYNTAXVER**)
 - Interchange sender ID qualifier (**ISIDQUAL**)
 - Interchange sender ID (**ISID**)
 - Interchange sender name (**ISENDNAME**)
 - Interchange reverse routing (**IREVROUT**)
 - Interchange receiver ID qualifier (**IRIDQUAL**)
 - Interchange receiver ID (**IRID**)
 - Interchange receiver name (**RECVNAME**)
 - Interchange routing address (**IROUTEADDR**)
 - Interchange password (**ISPW**)
 - Interchange application reference (**IAPREF**)
 - Interchange standard ID (**ISTDID**)
 - Interchange priority (**IPRIOR**)
 - Interchange communication agreement (**ICOMMAGREE**)
 - Interchange version/release (**IVERREL**)

Fields Causing New Group: During an enveloping operation, the current transaction is checked against the current group to determine whether this transaction belongs in the current group or whether the current group should be closed and a new one started. The following fields are checked for changes to see if a new group needs to be started:

- Functional group ID assigned to the transaction
- Security profile member being used
- Group encryption key name
- Group authentication key name
- Envelope profile member being used
- Application sender ID in the send usage

- Application receiver ID in the send usage
- Application password in the send usage
- Any of the following override values within the translator control block (TRCB)
 - Application sender ID qualifier (**GSIDQUAL**)
 - Application sender ID (**GSID**)
 - Application receiver ID (**GRID**)
 - Application receiver ID qualifier (**GRIDQUAL**)
 - Group password (**GAPW**)
 - Group version (**GVER**)
 - Group release (**GREL**)
 - Group responsible agency (**GRESPAGENCY**)

API - Close and Queue Interchange

This function allows your application program to have direct control over the content and size of interchanges being created. Left alone, the translator/enveloper continues to add transactions to the current interchange until there is a change in one of the fields that signals a new interchange. For more information, see “Fields Causing New Interchange” on page 3-83. If your application has special requirements for the interchanges created, it is possible to issue a close-and-queue envelope request at any time. If C and D records are being used as input to the DataInterchange Utility, a Z1 record can be used to tell the utility to terminate the current interchange. When a Z1 record is read, the utility issues the API request to close and queue the current interchange. When this API request is issued, the current interchange is closed by adding a group trailer segment (if groups are being used) and an interchange trailer segment, and the complete interchange is given to the communications service, which writes the interchange to the file associated with the network.

The basic format of the API request to close and queue the current interchange is:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter Description

SNB	The service name block identifying translation services.	
	Field	Value
	ZSNBNAME	TRANPROC to identify translation services
	ZSNBPC	6 (number of parameters in the call)
CCB	The common control block used to initialize DataInterchange.	
FCB	The function control block with a ZFCBFUNC value of 990 to close and queue the current interchange.	
TRCB	The translator control block. This is the same transaction control block used in all other requests. There are no field initialization requirements for this control block unless you have special COMMIT requirements. See “Translator Control Block (TRCB)” on page A-6 for details.	
TRIDB	The input data block. No data is required in this block.	
TRODB	The output data block. DataInterchange uses the output data block as a work buffer. This buffer must be at least 32000 bytes in length, but its maximum size must be the size of the maximum standard segment (excluding the BIN segment) that is processed. See “Translator Output Data Block (TRODB)” on page A-31 for a general description of this block.	

Queueing - TRCB Fields: Table 3-61 describes the fields in the transaction control block returned when an interchange is written.

The **EJECT** field indicates whether an interchange is written. The other fields in the table provide information about the interchange, the network, and the file receiving the interchange.

Table 3-61. Fields in TRCB Returned When an Interchange is Written

TRCB	Description						
EJECT	<p>If an interchange was written to the file associated with the network, the EJECT field has one of the following values:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Q</td><td>The interchange was written successfully.</td></tr> <tr> <td>E</td><td>The interchange was NOT written successfully.</td></tr> </table> <p>Note: If EJECT has a value of E, indicating an error occurred, the QRC and QERC fields have values indicating the error. The specific error that occurred is logged by DataInterchange communications services. The most likely error is that the ddname could not be opened.</p>	Value	Description	Q	The interchange was written successfully.	E	The interchange was NOT written successfully.
Value	Description						
Q	The interchange was written successfully.						
E	The interchange was NOT written successfully.						
QRC	The return code associated with the error.						
QERC	The extended return code associated with the error.						
DSNAME	The physical data set name to which the interchange was written. In CICS the DSNAME has the same value as QDDNAME , which is the name of the temporary storage queue containing the interchange.						
QNETID	The name of the network associated with the trading partner						
QPTTOPT	Has a value of Y if the network identified by QNETID is a point-to-point network.						
QSRPGM	Has a value of Y if the network identified by QNETID has a network send and receive program associated with it. If there is no send and receive program associated with a network, there is no need to attempt to send the interchange. A network might exist without a send and receive program for creating interchanges that are sent to a trading partner using some other means.						
QDDNAME	The ddname to which the interchange was written. This field does not have a value if a point-to-point network is being used. In CICS, this is the name of the temporary storage queue to which the interchange is written.						
QTPNICK	The trading partner nickname for which the interchange is built.						
QSIZE	The total number of bytes that are in the interchange, stored as a 4-byte binary value.						
QBT	Character representation of QSIZE .						

See “Sending Transaction Data” on page 3-82 for an explanation of items that need to be considered if your application program uses the communications services send API for sending the data just written to the file.

API - End Translation/Enveloping

If the translator has been called at least once and the **TRABORT** flag does not have a value of Y, the translator should be invoked one last time with a termination function code. The termination function code signals the translator that the application is finished making translation or enveloping requests. On receiving the request, the translator releases any resources that have been acquired. If an interchange is active, the translator completes and writes the current interchange.

The basic format of the API request to end translation/enveloping is :

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The parameters for this API request are:

Parameter	Description
-----------	-------------

SNB	The service name block identifying translation services.	
	Field	Value
	ZSNBNAME	TRANPROC to identify translation services
	ZSNBPC	6 (number of parameters in the call)
CCB	The common control block used to initialize DataInterchange.	
FCB	The function control block with a ZFCBFUNC value of 1000.	
TRCB	The translator control block. The same TRCB that was used in all the other requests.	
TRIDB	The input data block. No data is required in this block.	
TRODB	The output data block. DataInterchange uses the output data block as a work buffer. This buffer must be at least 32000 bytes in length, but its maximum size should be the size of the maximum standard segment (excluding the BIN segment) that will be produced. See "Translator Output Data Block (TRODB)" on page A-31 for a general description of this block.	

If an interchange is active at the time of the termination request, fields in the translator control block (TRCB) indicate that an interchange was written. See "API - Close and Queue Interchange" on page 3-84 for an explanation of these fields.

Deenvelope Function

This section describes the details of the deenvelope API request. Deenveloping is the act of extracting interchanges from a file and parsing those interchanges to extract transactions to add to the Transaction Store. The syntax of the interchange or transaction is checked and functional acknowledgments are generated. Received acknowledgments are reconciled with the original transactions.

Note: Deenveloping does not need to be a separate API request. It is possible to deenvelope and translate transactions at the same time. For more information, see "Receiving and Deenveloping" on page 3-46 and "API - Translate File" on page 3-57. The API described in the following section is the same API the DataInterchange Utility uses internally when you issue any of the following commands:

- PERFORM DEENVELOPE
- PERFORM DEENVELOPE AND TRANSLATE
- PERFORM RECEIVE AND DEENVELOPE
- PERFORM RECEIVE AND TRANSLATE

Each time the deenveloping API is invoked, the next transaction from the file being processed is located and added to the Transaction Store. Sometimes locating the next transaction involves locating the next interchange within the file. Once an interchange is located, the complete interchange is read into virtual storage. A syntax check and validation are done on the service segments to verify that the interchange received has a valid format and is consistent.

API - Deenvelope

The basic format of the API request to deenvelope transaction from a file is:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter	Description
-----------	-------------

SNB	The service name block identifying translation services.
-----	--

Field	Value
ZSNBNAME	TRANPROC to identify translation services
ZSNBPC	6 (6 parameters in the call)

CCB	The common control block used to initialize DataInterchange.
-----	--

FCB	The function control block with a ZFCBFUNC value of 214.
-----	---

TRCB	The translator control block. For more information on this block, see “Translator Control Block (TRCB)” on page A-6.
------	--

TRIDB	The input data block. DataInterchange uses the input data block as a work buffer. This buffer must be at least 32000 bytes in length, but its maximum size should be the size of the maximum standard segment (excluding the BIN segment) that is received. See “Translator Input Data Block (TRIDB)” on page A-29 for a general description of this block.
-------	---

TRODB	The output data block. DataInterchanges uses the output data block as a work buffer. This buffer must be at least 32000 bytes in length. See “Translator Output Data Block (TRODB)” on page A-31 for a general description of this block.
-------	---

In the remainder of this section, the phrase *calling the translator* means *Invoking the DataInterchange API with a request to deenvelope*. When reference to a field within the TRCB, the field is uppercase and highlighted (for example, **EJECT**).

Initializing for Deenvelope API

Numerous fields in the control blocks defined for the deenvelope API function need to be established only once. These values can be established before the first call. They do not need to be refreshed or changed before any other call. The control blocks, the fields within the control blocks, and the initialization considerations are shown in the following tables.

Table 3-62. SNB Initialization for Deenvelope

SNB	Initialization Value
ZSNBLL	32
ZSNBNAME	TRANPROC (the service name for enveloping services)
ZSNBPC	6 (All calls for enveloping services have six parameters.)

Table 3-63. FCB Initialization for Deenvelope

FCB	Initialization Value
ZFCBLL	4
ZFCBFUNC	214 (for deenvelope)

Table 3-64 (Page 1 of 3). TRCB Initialization for Deenvelope

TRCB	Initialization Value
BLKLEN	1536 (the length of the translator control block in Version 2)
BLKNME	EDITRCB

Table 3-64 (Page 2 of 3). TRCB Initialization for Deenvelope

TRCB	Initialization Value						
BLKTYPE	There are two possible formats for the TRIDB and TRODB buffer. One format has a buffer size limited to 32768 while the other format has no limit. A value of H indicates you are using the unlimited format. Any other value indicates you are using the format with a limit of 32768.						
XPANDED	Y						
ENVLDELAY	This field indicates if enveloping of functional acknowledgments should occur. A value of Y indicates that enveloping should NOT occur. Any other value indicates enveloping should occur.						
DUPTRAN	<p>Indicates how duplicate interchanges should be processed. DUPTRAN is only an input field on the first call of a session and establishes if duplicate interchanges are errors. On all other requests, DUPTRAN is an output field. Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>N</td><td>Indicates that duplicate interchanges should not be processed but should be considered errors. If a duplicate interchange is received, the translator will issue message TR0211 and return to the application with an interchange level error (extended return code value of 5).</td></tr> <tr> <td>Y</td><td>Indicates that duplicate interchanges should be processed and transactions returned (transactions are flagged as duplicate transactions). Y is the suggested value.</td></tr> </table>	Value	Description	N	Indicates that duplicate interchanges should not be processed but should be considered errors. If a duplicate interchange is received, the translator will issue message TR0211 and return to the application with an interchange level error (extended return code value of 5).	Y	Indicates that duplicate interchanges should be processed and transactions returned (transactions are flagged as duplicate transactions). Y is the suggested value.
Value	Description						
N	Indicates that duplicate interchanges should not be processed but should be considered errors. If a duplicate interchange is received, the translator will issue message TR0211 and return to the application with an interchange level error (extended return code value of 5).						
Y	Indicates that duplicate interchanges should be processed and transactions returned (transactions are flagged as duplicate transactions). Y is the suggested value.						
SCOPE	<p>A value of E indicates that interchange level recovery is wanted, and DataInterchange does not issue a database COMMIT until an interchange is complete. Any other value indicates the recovery scope should be at the transaction level, and DataInterchange issues a database COMMIT at the start of every transaction.</p> <p>Note: This field contains values that affect the recovery scope during the session. For more information about this field, see “Send Recovery Scope” on page 3-42.</p>						
INMEMTRANS	<p>This parameter applies only if the value of SCOPE is E and indicates the maximum number of transactions that should be maintained in virtual storage before any database updates are attempted. Database updating occurs whenever INMEMTRANS or the end of the envelope is reached. This value is important only in an environment where multiple application programs are requesting translation services concurrently. The higher the value for INMEMTRANS, the more concurrency achieved. For more information about this field, see “Translator Control Block (TRCB)” on page A-6.</p> <p>Note: This field contains values that affect the recovery scope during the session. For more information about this field, see “Send Recovery Scope” on page 3-42.</p>						
FILEID	The dname of the file that contains the interchanges that should be processed. This is an optional field. If provided, REQID is ignored.						
IUSEREXIT	Provides the logical name of a user exit to be called by DataInterchange instead of reading the input file. This exit is used when the envelope is to be stored into DataInterchange through the Put Envelope Service before deenveloping. For more information on the Get/Put Envelope Exit processing, see Chapter 4, “Exit Routines.”						
IUSERAREA	A pointer to a user-defined area. The pointer is passed to the user exit defined in IUSEREXIT field.						
IUSERACCESS	<p>A flag that indicates how the interchange should be presented to the IUSEREXIT program.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>M</td><td>A value of M indicates that the interchange should be given to the exit in virtual storage. This option only applies when the IUSERTYPE is UE (user exit).</td></tr> <tr> <td>F</td><td>A value of F indicates that the interchange should be given to the exit in a file. With an IUSERACCESS value of F, the interchange is first written to the transaction data queue file and then the IUSEREXIT program is invoked.</td></tr> </table>	Value	Description	M	A value of M indicates that the interchange should be given to the exit in virtual storage. This option only applies when the IUSERTYPE is UE (user exit).	F	A value of F indicates that the interchange should be given to the exit in a file. With an IUSERACCESS value of F, the interchange is first written to the transaction data queue file and then the IUSEREXIT program is invoked.
Value	Description						
M	A value of M indicates that the interchange should be given to the exit in virtual storage. This option only applies when the IUSERTYPE is UE (user exit).						
F	A value of F indicates that the interchange should be given to the exit in a file. With an IUSERACCESS value of F, the interchange is first written to the transaction data queue file and then the IUSEREXIT program is invoked.						

Table 3-64 (Page 3 of 3). TRCB Initialization for Deenvelope

TRCB	Initialization Value						
IUSERTYPE	<p>The type of program specified in IUSEREXIT.</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>PG</td><td>A type of PG indicates that IUSEREXIT is a program that should be linked to (EXEC CICS LINK in CICS).</td></tr> <tr> <td>UE</td><td>A type of UE indicates that IUSEREXIT is a DataInterchange user exit program defined in the ADAMCTL profile</td></tr> </table>	Value	Description	PG	A type of PG indicates that IUSEREXIT is a program that should be linked to (EXEC CICS LINK in CICS).	UE	A type of UE indicates that IUSEREXIT is a DataInterchange user exit program defined in the ADAMCTL profile
Value	Description						
PG	A type of PG indicates that IUSEREXIT is a program that should be linked to (EXEC CICS LINK in CICS).						
UE	A type of UE indicates that IUSEREXIT is a DataInterchange user exit program defined in the ADAMCTL profile						
REQID	The requestor ID identifies a member in the requestor profile (REQPROF). The <i>Receive file name</i> field in the requestor profile contains the name of the file containing the interchanges that should be processed. This field is required only if FILEID is not provided.						
MRREQID	If the interchanges being deenveloped have not been recorded in the Management Reporting statistics database, the MRREQID value identifies the requestor ID for which statistics should be updated. Updating is necessary only if the interchanges were not received using the communications service receive function.						
FUNACKFLE	If enveloping of functional acknowledgments is occurring, FUNACKFLE is the ddname of the file where the transaction data is written when an interchange is complete. This field is optional. If not provided, the ddname specified in the <i>Trans data queue</i> field of the network profile (NETPROF) is used.						
TRXLIFE	<p>The amount of time this transaction should remain in the Transaction Store before it is eligible to be purged. The default value is 30 days.</p> <p>Note: This field is checked with each call to the translator and if a value is not supplied, the default value is used. The value that exists when the "deenvelope transaction" is made will be the value associated with the transaction.</p>						
IMGLIFE	<p>IMGLIFE is the amount of time this transactions image (the standard data produced) should remain in the Transaction Store. The default value is 30 days.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. This field is checked with each call to the translator and if a value is not supplied, the default value is used. The value that exists when the "deenvelope transaction" is made will be the value that is associated with the transaction. 2. This field is not currently supported. 						
HOLDFLAG	<p>A value of Y should be used if this transaction is to be placed in a hold status when added to the Transaction Store. A transaction in hold status is not available for any other activity until it is released. A value of N is suggested if the transaction is not to be held. Any value other than Y is treated like N.</p> <p>Note: This field is checked with each call to the translator and if a value is not supplied, the default value is used. The value that exists when the "deenvelope transaction" is made will be the value associated with the transaction.</p>						
ERRFILTER	Values that indicate which errors should be filtered out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more information, see "Error Filtering" on page 1-4.						
FORCETEST	<p>Indicates whether the deenvelope process is to be forced to select only a test usage regardless of the value of the test indicator in the envelope. A value of Y forces the process to test mode and selects only a test usage if it is defined. If a test usage is not found, an error is generated and the transaction is rejected. Any other value uses the test indicator from the envelope to determine the usage to select. For those envelopes without a test indicator, it is always considered a production envelope.</p> <p>This flag is most useful when receiving test envelopes, but the envelopes do not have a test indicator (such as the UCS BG). In this case, the FORCETEST(Y) can be used to force the translator to consider the envelopes to be tested and only look for test usages.</p>						

Table 3-65. TRIDB Initialization for Deenvelope

TRIDB	Initialization Value
BLKLEN	Set to the size of the data block, including the BLKLEN field. The minimum value for this field is 32000 bytes.
RESERVED	Binary zeros.

Table 3-66. TRODB Initialization for Deenvelope

TRODB	Initialization Value
BLKLEN	Set to the size of the data block, including the BLKLEN field. The minimum value for this field is 32000 bytes.
RESERVED	Binary zeros.

Because the first call for a session also qualifies as the first call for a transaction, follow the instructions in the next section.

Deenvelope Transaction

There are no initialization requirements before making the request to deenvelope a transaction, unless you want the next transaction to have different values for **TRXLIFE**, **IMGLIFE**, or **HOLDFLAG** from the current transaction. Once any initialization is complete, the translator is invoked with the following API request:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

On the first call to the translator for the session, the file containing the interchanges (identified by the **FILEID** or the **REQID**) is opened and the first interchange is located. The entire interchange is read into virtual storage and validated. The interchange sender ID and sender qualifier are extracted from the interchange header. These values are used to determine the trading partner that sent the data. See “Locate EDI and Application Trading Partners” on page 3-96 for an explanation of the search done for a trading partner.

On the first call to the translator for a transaction, the next transaction within the current file is located (this may involve finding the next interchange) and the transaction/message ID value is extracted from the transaction header. The translator must attempt to locate a trading partner transaction usage and map that provides the instructions for translating the transaction from the standard format into an application format. This search is done even though a translation does not take place. The usage indicates the type of functional acknowledgment that is wanted, and the mapping indicates the segments and data elements within the transaction in which an interest has been expressed.

Functional acknowledgment checking takes place only on segments and data elements that have been mapped. The mapping and the usages must have been established by the EDI Administrator before the API requests are made. See “Locate Receive Map” on page 3-97 for an explanation of the fields used and the search done to locate a mapping. The deenvelope operation still takes place without a usage or a map, but no functional acknowledgment is possible unless they exist. Numerous errors can occur at this point. All the errors from the translator are contained in “Translation Return Codes” on page B-19. If an error occurs, the return code and extended return code for that error are posted in the **ZCCBRC** and **ZCCBERC** fields of the CCB. Two fields within the translator control block (TRCB) should be checked to determine the status of the transaction and the status of the translator:

Field	Description
TRXACCEPT	Has a value of Y if this transaction was deenveloped successfully. Any value other than Y indicates that something needed to process this transaction is not available. If the value is not Y and your program chooses to continue, the next call to the translator is again interpreted as the first call for a transaction.
TRABORT	Has a value of Y if the error was so severe that the translator did not continue. If the value is Y and your program chooses to continue after such an error, the next call you make to the translator is considered the first call for the session. If the FILEID or REQID is not changed, the continuation results in an infinite loop.

Transaction TRCB Fields (DE): The fields in the following tables are returned on the deenvelope transaction request. Such a large amount of information is returned that multiple tables are used to show the TRCB fields.

Table 3-67 shows fields that provide some basic attributes of the transaction just processed. These fields are concerned with the transaction side of the processing rather than the application side.

Table 3-67. Transaction Attribute Fields in TRCB

TRCB	Description								
TRNID	The standard transaction or message ID for the transaction or message received.								
TEST	<table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>P</td><td>Production transaction</td></tr> <tr> <td>T</td><td>Test transaction</td></tr> <tr> <td>I</td><td>Information transaction</td></tr> </table>	Value	Description	P	Production transaction	T	Test transaction	I	Information transaction
Value	Description								
P	Production transaction								
T	Test transaction								
I	Information transaction								
MAPKEY	Identifies the trading partner transaction used to translate the standard data. Applies only if a usage was found.								
TPNICK	The trading partner nickname associated with the transaction.								
DUPTRAN	Y if this transaction is a duplicate. Indicates that the interchange being received has been received before.								
TRXACCEPT	Y if the transaction had an acceptable translation. Indicates that application data has been returned in the translator output data block (TRODB) (unless the EJECT field has a value of Y).								
TRABORT	Y if an error occurred that was so severe that the translator did not continue. If this flag is set, the translator has ended of its own accord.								
TSKEY	The key value assigned to the transaction within the Transaction Store. This is called the transaction handle and has a format of YYYYMMDDHHMMSSxxnnnn. It is returned as a 10-byte packed value in this field, and stored in the database.								
TSKEYU	The key value assigned to the transaction within the Transaction Store. The same value as the TSKEY field, but TSKEYU is an unpacked (character) 20-byte value.								
ERRNUM	The total number of errors flagged in processing the data.								
ERRCDES	An array of the first 10 errors flagged in processing the data. Each possible error detected by the translator has a unique error code. See Table A-5 on page A-28 for a list of errors.								

Table 3-68 on page 3-92 shows the fields that are application-related. These fields are concerned with the application side of the processing rather than the transaction side.

Table 3-68. Application-Related Fields in TRCB

TRCB	Description
ATFID	The application data format definition ID associated with this transaction. Applies only if a usage was found.
INTPID	The internal trading partner ID (vendor number) taken from the trading partner transaction receive usage. Applies only if a usage was found.

Table 3-69. Fields Related to Functional Acknowledgments

TRCB	Description																
FABUILT	<p>This field is set equal to the envelope type for the functional acknowledgment. The possible values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>E</td><td>UNB/UNZ</td></tr> <tr> <td>I</td><td>ICS/ICE</td></tr> <tr> <td>T</td><td>STX/END</td></tr> <tr> <td>U</td><td>BG/EG</td></tr> <tr> <td>X</td><td>ISA/IEA</td></tr> <tr> <td>G</td><td>Indicates the acknowledgment does not have an interchange header</td></tr> <tr> <td>S</td><td>Indicates the acknowledgment was written to the Transaction Store, but was not enveloped</td></tr> </table>	Value	Description	E	UNB/UNZ	I	ICS/ICE	T	STX/END	U	BG/EG	X	ISA/IEA	G	Indicates the acknowledgment does not have an interchange header	S	Indicates the acknowledgment was written to the Transaction Store, but was not enveloped
Value	Description																
E	UNB/UNZ																
I	ICS/ICE																
T	STX/END																
U	BG/EG																
X	ISA/IEA																
G	Indicates the acknowledgment does not have an interchange header																
S	Indicates the acknowledgment was written to the Transaction Store, but was not enveloped																
FARC	The return code from the translator for the functional acknowledgment translation done during deenvelope. This return code can be found in <i>DataInterchange Messages and Codes</i> .																
FAERC	The extended return code from the translator for the functional acknowledgment translation done during deenvelope. This return code can be found in <i>DataInterchange Messages and Codes</i> .																

Table 3-70 contains service segment fields (TF) concerned with the interchange to which the current transaction belongs.

Table 3-70 (Page 1 of 3). Interchange Header/Trailer Fields in TRCB

TRCB	Description
DSNAME	The physical data set name from which transactions are being processed. This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.
QTPNICK	The trading partner nickname for which the last transaction processed. This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.
QSIZE	The total number of bytes in the interchange, stored as a 4-byte binary value. This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.
QBT	Character representation of the QSIZE field. This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.

Table 3-70 (Page 2 of 3). Interchange Header/Trailer Fields in TRCB

TRCB	Description												
ENVTYPE	<p>Indicates the envelope type being received. Possible values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>E</td><td>UNB/UNZ</td></tr> <tr> <td>I</td><td>ICS/ICE</td></tr> <tr> <td>T</td><td>STX/END</td></tr> <tr> <td>U</td><td>BG/EG</td></tr> <tr> <td>X</td><td>ISA/IEA</td></tr> </table> <p>This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.</p>	Value	Description	E	UNB/UNZ	I	ICS/ICE	T	STX/END	U	BG/EG	X	ISA/IEA
Value	Description												
E	UNB/UNZ												
I	ICS/ICE												
T	STX/END												
U	BG/EG												
X	ISA/IEA												
IHCTL	See IHXCTL .												
IHXCTL	The interchange control number assigned to the current interchange. The value returned in this field is right-justified with leading zeros. This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.												
ISYNTAXID	The interchange syntax ID used in EDIFACT and UNTDI envelopes. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.												
ISYNTAXVER	The interchange syntax version used in EDIFACT and UNTDI envelopes. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.												
ISIDQUAL	The interchange sender ID qualifier used in EDIFACT, ICS, and X12 envelopes. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.												
ISID	The interchange sender ID (Interchange header field with IS data type). This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.												
IrecvNAME	The interchange receiver name used in UNTDI envelopes. The application receiver code used in the UCS envelopes if UCS04 is not an IR data type. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.												
IRouteADDR	The interchange routing address used in EDIFACT envelopes. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.												
ISendNAME	The interchange sender name used in UNTDI envelopes. The application sender code used in the UCS envelopes if UCS03 is not an IS data type. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.												
IREVROUT	The interchange reverse routing used in EDIFACT envelopes. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.												
IRIDQUAL	The interchange receiver ID qualifier used in EDIFACT, ICS, and X12 envelopes. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.												
IRID	The interchange receiver ID (Interchange header field with IR data type). This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.												
IDATE	The interchange date (Interchange header field with DT data type). This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.												

Table 3-70 (Page 3 of 3). Interchange Header/Trailer Fields in TRCB

TRCB	Description
ITIME	The interchange time (Interchange header field with TM data type). This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.
IVERREL	The interchange version and release (Interchange header field with VR or LV data type). This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.
ISPW	The interchange password (Interchange header field with PW data type). This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.
IAPREF	The application reference (Interchange header field with AP data type) This field is established when the first transaction for the interchange is processed and remains constant for all the transactions within this interchange.
ISTDID	The interchange standard ID used in ICS and X12 envelopes. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.
IPRIOR	The interchange processing priority used in EDIFACT and UNTDI envelopes. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.
ICOMMAGREE	The interchange communication agreement used in EDIFACT envelopes. This field is established when the first transaction for the interchange is processed and remains consistent for all the transactions within this interchange.
NEWENV	Y if the transaction is the first transaction of an interchange. If a copy of the interchange header is wanted, a function code value of 1 can be used to obtain an exact image of the interchange header. For more information, see "API - Retrieve Interchange Header" on page 3-99.
GRPNUM	Total number of groups processed so far within the current interchange stored as a 4-byte binary value (see IGT). This field changes as transactions within the interchange are processed and indicate how much of the interchange has been processed.
TRNNUM	Total number of transactions processed so far within the current interchange stored as a 4-byte binary value (see ITT). This field changes as transactions within the interchange are processed and indicate how much of the interchange has been processed.
SEGNUM	Total number of segments processed so far within the current interchange stored as a 4-byte binary value (see IST). This field changes as transactions within the interchange are processed and indicate how much of the interchange has been processed.
ESIZE	Total number of bytes processed so far within the current interchange stored as a 4-byte binary value (see IBT). This field changes as transactions within the interchange are processed and indicate how much of the interchange has been processed.
IGT	Character representation of GRPNUM .
ITT	Character representation of TRNNUM .
IST	Character representation of SEGNUM .
IBT	Character representation of ESIZE .

Table 3-71 on page 3-95 contains fields concerned with the group to which the current transaction belongs.

Table 3-71. Group Header/Trailer Fields in the TRCB

TRCB	Description
GHCTL	See GHXCTL .
GHXCTL	The group control number assigned to the current group. The value returned in this field is right-justified with leading zeros. This field is established when the first transaction for the group is processed and remains constant for all the transactions within this group.
GSIDQUAL	The group sender ID qualifier used in EDIFACT envelopes. This field is established when the first transaction for the group is processed and remains consistent for all the transactions within this group.
GSID	The group sender ID (Group header field with AS data type). This field is established when the first transaction for the group is processed and remains constant for all the transactions within this group.
GRIDQUAL	The group receiver ID qualifier used in EDIFACT envelopes. This field is established when the first transaction for the group is processed and remains consistent for all the transactions within this group.
GRID	The group receiver ID (Group header field with AR data type). This field is established when the first transaction for the group is processed and remains constant for all the transactions within this group.
GDATE	The group date (Group header field with DT data type). This field is established when the first transaction for the group is processed and remains constant for all the transactions within this group.
GTIME	The group time (Group header field with TM data type). This field is established when the first transaction for the group is processed and remains constant for all the transactions within this group.
GAPW	The group password (Group header field with PW data type). This field is established when the first transaction for the group is processed and remains constant for all the transactions within this group.
GVER	The group version (Group header field with VR data type). This field is established when the first transaction for the group is processed and remains constant for all the transactions within this group.
GREL	The group release (Group header field with LV data type). This field is established when the first transaction for the group is processed and remains constant for all the transactions within this group.
NEWGRP	Y if the transaction is the first transaction of a group. If a copy of the group header is wanted, a function code value of 2 can be used to obtain an exact image of the group header. For more information, see "API - Retrieve Group Header" on page 3-100.
TRNGRP	Total number of transactions processed so far within the current group, stored as a 4-byte binary value (see GTT). This field changes as transactions within the group are processed and indicate how much of the group has been processed.
GTT	Character representation of TRNGRP .
GRESAGENCY	The group controlling agency used in EDIFACT envelopes. The group responsible agency used in ICS, UCS, and X12 envelopes. This field is established when the first transaction for the group is processed and remains consistent for all the transactions within this group.

Table 3-72 on page 3-96 contains fields concerned with the transaction header for the current transaction.

Table 3-72. Transaction Header/Trailer Fields in the TRCB

TRCB	Description
NEWTRN	Y if a transaction header is available. If a copy of the transaction header is wanted, a function code value of 3 can be used to obtain an exact image of the transaction header. For more information, see "API - Retrieve Transaction Header" on page 3-100.
LASTINENV	Y if this transaction is the last transaction within the current interchange.
THCTL	See THXCTL .
THXCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros.
SEGTRN	Total number of segments within the current transaction stored as a 4-byte binary value (see TST).
TTC	The current transaction or message ID value (see TRNID).
TVER	The transaction version (Transaction header field with VR data type).
TREL	The transaction release (Transaction header field with LV data type).
TST	Character representation of SEGTRN .

Last Call of Session (DE)

The translator signals the last call for a session through the CCB return codes. A **ZCCBRC** value of 4 and a **ZCCBERC** value of 1 indicate that all the data from the current file has been processed and that the translator has terminated. If your application is to process another file, cycle back to the first call of the session and set the **REQID** or **FILEID** field to indicate the next file that should be processed.

Special Considerations

The following sections list special considerations.

Locate EDI and Application Trading Partners

The following search sequence is used in an attempt to locate a member in the trading partner profile (TPPROF):

1. The sender qualifier and sender ID are used to search the profile members to find an entry with matching Interchange Qualif and Interchange ID fields.
2. If the interchange ID is 15 or fewer bytes in length, the interchange ID is parsed into a 7-byte account and an 8-byte user ID. The trading partner profile members are searched to find a match on the Account Number and User ID fields.
3. If the interchange ID is 16 or fewer bytes in length, the interchange ID is parsed into a 8-byte account and an 8-byte user ID. The trading partner profile members are searched to find a match on the Account Number and User ID fields.
4. If the interchange ID is 35 or fewer bytes in length, the interchange ID is parsed into a 32-byte account and a 3-byte user ID. The trading partner profile members are searched to find a match on the Account Number and User ID fields.
5. The interchange ID is parsed into an account and user ID separated by blanks, periods, or slashes. The trading partner profile members are searched to find a match on the Account Number and User ID fields.
6. The interchange ID searches the trading partner profile members to find a match on the Contact Phone field.

7. A TTABLE is searched to locate an account number and user ID given the interchange ID. The trading partner profile members are searched to find a match on the Account Number and User ID fields taken from the TTABLE entry.

8. If a trading partner profile was not located, then the trading partner is considered to be unknown.

Locate Receive Map

The following are the fields used to locate a map:

- The EDI trading partner nickname, if found, located through the search detailed previously.
- The application trading partner nickname, if found, located through the search detailed previously.
- The application sender ID value extracted from the group header. If groups are not being used, the reverse sender ID (**RS** field) from the interchange header is used. If an **RS** field is not defined, the application sender ID (**AS** field) within the interchange header is used. If neither an **RS** or an **AS** field is defined within the interchange header, or if the defined field contains blanks, the interchange sender ID (**IS** field) is used.
- The application receiver ID value extracted from the group header. If groups are not being used, the reverse receiver ID (**RR** field) from the interchange header is used. If an **RR** field is not defined, the application receiver ID (**AR** field) within the interchange header is used. If neither an **RR** or an **AR** field is defined within the interchange header, or if the defined field contains blanks, the interchange receiver ID (**IR** field) is used.
- The test indicator (**TI** field) from the interchange header. Production is assumed if the envelope type does not contain a TI data type.
- The transaction or message ID (**TC** field) extracted from the transaction or message header.
- The responsible agency code (**AG** field) taken from the GS or UNH segments. If an **AG** field is not defined, a blank value is assumed and, therefore, the 'Responsible agency code' value in the usage must also be blank.
- The version (**VR** field) taken from the UNH, MHD, or GS segments. In the case of **GS**, the first three bytes of the **VR** or **LV** field are assumed to be the version. If a **VR** field is not defined, a blank value is assumed and, therefore, the 'Version' value in the usage must also be blank.
- The release (**LV** field) taken from the UNH, or GS segments. In the case of **GS**, the fourth, fifth, and sixth bytes of the **VR** or **LV** field are assumed to be the release. If a **VR** field is not defined, a blank value is assumed and, therefore, the 'Release' value in the usage must also be blank. The version and release fields may be filled in the TP usage to further define keys for which mapping should be used during translation.

DataInterchange will make a single call to the database to retrieve all the active usages for a given trading partner and transaction ID (810, ORDERS, and so on). If this is a production transaction, only production usages will be retrieved. If this is a test transaction, both test and production usages will be retrieved.

The FORCETEST(Y) keyword in the batch utility can be used to override the usage indicator and force the translator to look at only the test usages. In this mode, if a test usage is not found, the transaction is rejected. Once all usages have been retrieved, the data from the usage will be compared with the data from the transaction to find the best usage. The best usage is determined by adding up the weights that are assigned to the various fields by DataInterchange and picking the usage that has the greatest weight value. A usage is not considered at all if a value has been supplied in the usage that does not match a value from the transaction. For example, if an application sender ID was specified in the usage and the usage value does not match the transaction value, then that usage will not be considered.

The fields and weight values are:

Field	Weight Value
Application sender	4096
Application receiver	2048
Responsible agency code	1024
Version	512
Release	256
Test mode matches	128
Test mode does not match	64
Sending trading partner is specific	32
Receiving trading partner is specific	16
Sending trading partner is KNOWN	8
Receiving trading partner is KNOWN	4
Sending trading partner is ANY	2
Receiving trading partner is ANY	1

Note: When using generic receive usages, DataInterchange makes a second call to retrieve all of the generic usages (an ampersand in the trading partner nickname plus standard transaction ID) and the above weighting values select a generic receive usage.

API - Issue Commit

The only time this function call is necessary is if another request has a **NOCOMIT** field value of Y, indicating that the translator should not issue a COMMIT function at a point where it usually would do so. In this case, the the translator does not issue a COMMIT function. Your program then has an opportunity to change resources (such as updating a database) and then request a COMMIT function so that the changes made by DataInterchange and the changes made by your application program can be synchronized. The request for a COMMIT function is not honored if an interchange is active and interchange level recovery scope (**SCOPE** value of E) is in effect. The request for a COMMIT function is not honored if transaction level recovery scope is in effect and a BUNDLE is in progress. The request for a COMMIT function is not honored if the SYNCPOINT interval has not been reached. For more information on SYNCPOINT interval, see "SYNCPOINT Services" on page 3-132.

The basic format of the API request to request a COMMIT function is:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter	Description
-----------	-------------

SNB	The service name block identifying translation services.
-----	--

Field	Value
ZSNBNAME	TRANPROC to identify translation services
ZSNBPC	6 (number of parameters in the call)

CCB	The common control block used to initialize DataInterchange.
-----	--

FCB	The function control block with a ZFCBFUNC value of 991.
-----	---

TRCB	The translator control block. The same TRCB that was used in all the other requests with the NOCOMIT field set to N.
------	---

TRIDB	The input data block. The same TRIDB that has been used on other requests.
-------	--

TRODB	The output data block. The same TRODB that has been used on other requests.
-------	---

API - Retrieve Interchange Header

This API function allows the application to retrieve the image of the current interchange header segment (ISA, UNB, STX, ICS, and BG). This request is valid only when enveloping or deenveloping is taking place. This API function request is used by the DataInterchange Utility when the I optional record is requested. If you are interested in the interchange header image, make this request when the NEWENV flag in the translator control block (TRCB) is set to Y, indicating that a new interchange has just been started. However, the request can be made at any time until a new interchange is started.

The format of the API request is:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter	Description
-----------	-------------

SNB	The same SNB used in previous requests. No further initialization is required.
-----	--

CCB	The common control block used to initialize DataInterchange.
-----	--

FCB	The function control block with a ZFCBFUNC value of 1 to indicate the current interchange header is wanted.
-----	--

TRCB	The same TRCB used in previous requests. No further initialization is required.
------	---

TRIDB	The same TRIDB used in previous requests. No further initialization is required.
-------	--

TRODB	The same TRODB used in previous requests. The translator returns the current interchange header image in the DATA field. The length of the image is returned in the DATALEN field.
-------	--

API - Retrieve Group Header

This API function allows the application to retrieve the image of the current group header segment (GS, BAT and UNG). This request is valid only when enveloping or deenveloping is taking place. This API function request is used by the DataInterchange Utility when the G optional record is requested. If you are interested in the group header image, make this request when the **NEWGRP** flag in the TRCB is set to Y, indicating that a new group has just been started. However, the request can be made at any time until a new group is started.

Note: BAT is only supported on receive. DataInterchange never creates a BAT segment.

The format of the API request is:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter	Description
SNB	The same SNB used in previous requests. No further initialization is required.
CCB	The common control block used to initialize DataInterchange.
FCB	The function control block with a ZFCBFUNC value of 2 to indicate the current group header is wanted.
TRCB	The same TRCB used in previous requests. No further initialization is required.
TRIDB	The same TRIDB used in previous requests. No further initialization is required.
TRODB	The same TRODB used in previous requests. The translator returns the current group header image in the DATA field. The length of the image is returned in the DATALEN field.

API - Retrieve Transaction Header

This API function allows the application to retrieve the image of the current transaction header segment (ST, MHD, and UNH). This request is valid only when enveloping or deenveloping is taking place. This API function request is used by the DataInterchange Utility when the T optional record is requested. If you are interested in the transaction header image, make this request when the **NEWTRN** flag in the TRCB is set to Y, indicating that a new transaction has just been started. However, the request can be made at any time until the next transaction is started.

The format of the API request is:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter	Description
SNB	The same SNB used in previous requests. No further initialization is required.
CCB	The common control block used to initialize DataInterchange.
FCB	The function control block with a ZFCBFUNC value of 3 to indicate the current transaction header is wanted.
TRCB	The same TRCB used in previous requests. No further initialization is required.
TRIDB	The same TRIDB used in previous requests. No further initialization is required.
TRODB	The same TRODB used in previous requests. The translator returns the current transaction header image in the DATA field. The length of the image is returned in the DATALEN field.

Data Extraction Services

Data extraction services provide functions that enable an application program to retrieve the same information extracted and written to the EDIQUERY file when a PERFORM TRANSACTION DATA EXTRACT or a PERFORM ENVELOPE DATA EXTRACT is performed. Your application receives this data from the output data block (TRODB).

The format of the API request for data extraction is:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The parameters for this API request are:

Parameter	Description
-----------	-------------

SNB	The service name block identifying translation services.
-----	--

Field	Value
-------	-------

ZSNBNAME	TRANPROC to identify translation services
----------	---

ZSNBPC	6 (number of parameters in the call)
--------	--------------------------------------

CCB	The common control block used to initialize DataInterchange.
-----	--

FCB	The function control block with a ZFCBFUNC value of:
-----	--

216	Detailed transaction data is wanted.
-----	--------------------------------------

217	A transaction image is wanted.
-----	--------------------------------

218	A transaction acknowledgment image is wanted.
-----	---

219	A functional acknowledgment image is wanted.
-----	--

TRCB	The translator control block. See “Translator Control Block (TRCB)” on page A-6 for details on this control block.
------	--

TRIDB	The input data block. See “Translator Input Data Block (TRIDB)” on page A-29 for a description of this block.
-------	---

TRODB	The output data block. This block contains the returned detailed transaction data. TRODB has a minimum size of 32000 bytes. For more information on this block, see the “Translator Output Data Block (TRODB)” on page A-31.
-------	--

Initialization for Data Extract

Numerous fields in the control blocks are defined for data extraction functions whose values need to be established only once. These values can be established before the first call and do not have to be refreshed or changed before any other call. The control blocks, the fields within the control blocks, and the initialization considerations are shown in the following tables.

Table 3-73. SNB Initialization for Data Extraction

SNB	Initialization Value
ZSNBLL	32
ZSNBNAME	TRANPROC
ZSNBPC	6

Table 3-74. FCB Initialization for Data Extraction

FCB	Initialization Value
ZFCBLL	4
ZFCBFUNC	216, 217, 218, or 219

Table 3-75. TRCB Initialization for Data Extraction

TRCB	Initialization Value
BLKLEN	1536 (the length of the translator control block in Version 2)
BLKNME	EDITRCB
BLKTYPE	There are two possible formats for the TRIDB and TRODB buffer. One format has a buffer size limited to 32768. The second format has no limit. A value of H indicates you are using the unlimited format. Any other value indicates you are using the format with a limit of 32768.

Table 3-76. TRIDB Initialization for Data Extraction

TRIDB	Initialization Value
BLKLEN	Set to the size of the data block, including the BLKLEN field. Note: The TRIDB is not used during a data extraction operation. It still must be supplied in the parameter list. A BLKLEN value of 16 is sufficient.
RESERVED	Binary zeros.

Table 3-77. TRODB Initialization for Data Extraction

TRODB	Translator output data block
BLKLEN	Set to the size of the data block, including the BLKLEN . The minimum value for this field is 32000 bytes.
RESERVED	Binary zeros

API - Retrieve Detailed Data

This API function allows the application to retrieve detailed information concerning a transaction. Information about the transaction is extracted from the Transaction Store and returned as formatted records in the translator output data block. For more information, see “Transaction Store Data Extract Information Categories” on page 2-80. The fields in Table 3-78 should be set before the API request to retrieve detailed data is made.

Table 3-78 (Page 1 of 2). Initialization for Each Transaction

TRCB	Initialization Value
TSKEY	The transaction handle for the target transaction. The handle is YYYYMMDDHHMMSSxxxxnnn, formatted as a packed field within 10 bytes. If your program does not pack these values, initialize this field to all blanks or all binary zeros using the TSKEYU field.
TSKEYU	The transaction handle for the target transaction. The handle is YYYYMMDDHHMMSSxxxxnnn, formatted as a 20-byte character field. This field is used only if TSKEY does not have a value.

Table 3-78 (Page 2 of 2). Initialization for Each Transaction

TRCB	Initialization Value
CONCATENATE	Valid values are: Y Detailed information records should be concatenated in the TRODB. N Detailed information records should be returned individually.
EJECT	Blank

Once the initialization is complete, the data extraction service is invoked with the following API request:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

Complete information about the transaction identified by the **TSKEY** or **TSKEYU** field is retrieved from the Transaction Store and formatted into the data extract record formats. For more information, see “Transaction Store Data Extract Record Formats” on page 2-81. The data extraction records are returned in the TRODB data block in the following sequence:

1. Interchange record
2. Group record, which can be followed by:
 - Functional acknowledgment interchange record
 - Functional acknowledgment group record
 - Functional acknowledgment transaction record
3. Transaction record, which can be followed by:
 - Transaction acknowledgment interchange record
 - Transaction acknowledgment group record
 - Transaction acknowledgment transaction record
4. Application record

An outbound transaction can be enveloped more than once. If this is the case, the preceding sequence is repeated for each interchange to which the transaction belongs. An inbound transaction can be translated more than one time. If this is the case, more than one application record is returned. If there is a functional acknowledgment for the group, the interchange, group, and transaction record for the functional acknowledgment are returned with the group record even if **CONCATENATE** has not been requested.

If there is a transaction acknowledgment for the transaction, the interchange, group, and transaction record for the transaction acknowledgment are returned with the transaction record even if **CONCATENATE** has not been requested.

Numerous errors can occur at this point. These errors are described in “Translation Return Codes” on page B-19. If an error occurs, the return code and extended return code for that error are posted in the **ZCCBRC** and **ZCCBERC** fields of the CCB. The following fields within the translator control block (TRCB) indicate the status of the extraction request and the status of the translator:

- TRXACCEPT** Has a value of Y if information about this transaction is extracted successfully. Any value other than Y indicates that something went wrong in gathering the information about the transaction.
- TRABORT** Has a value of Y if the error is severe and the translator stopped processing.
- If the value is Y and your program continues after such an error, the next call you make to the translator is considered the first call for the session.

If the transaction was acceptable (**TRXACCEPT** value of Y), depending on the value of the **CONCATENATE** flag, one of the following will occur:

- As much information as possible is concatenated and returned in the TRODB data block (**CONCATENATE** value of Y).
- The first record concerning the transaction is returned in the TRODB data block (**CONCATENATE** value other than Y).

The following information is returned in response to the first request for detailed information about a transaction.

Table 3-79. Returned Information on Extract Request

TRCB	Description						
TRXACCEPT	Y if data is extracted for the specified transaction.						
TRABORT	Y if an error is severe and the translator has stopped processing. If this flag is set, the translator has terminated of its own accord.						
TSKEY	Transaction handle formatted as a 10-byte packed value.						
TSKEYU	Transaction handle formatted as a 20-byte character value.						
EJECT	Has one of the following values: <table> <tr> <td>X</td><td>Indicates that CONCATENATE was requested and all the data did not fit in the TRODB buffer. Issue the request again, leaving the EJECT value as X to get the residual data. Continue to do so until you receive an EJECT value of Y.</td></tr> <tr> <td>N</td><td>Indicates that CONCATENATE was not requested and more data remains for this transaction. Issue the request again, leaving the EJECT value at N to get the next detailed record. Continue to do so until you receive an EJECT value of Y.</td></tr> <tr> <td>Y</td><td>Indicates that the data returned on this request is the last data for the transaction.</td></tr> </table>	X	Indicates that CONCATENATE was requested and all the data did not fit in the TRODB buffer. Issue the request again, leaving the EJECT value as X to get the residual data. Continue to do so until you receive an EJECT value of Y.	N	Indicates that CONCATENATE was not requested and more data remains for this transaction. Issue the request again, leaving the EJECT value at N to get the next detailed record. Continue to do so until you receive an EJECT value of Y.	Y	Indicates that the data returned on this request is the last data for the transaction.
X	Indicates that CONCATENATE was requested and all the data did not fit in the TRODB buffer. Issue the request again, leaving the EJECT value as X to get the residual data. Continue to do so until you receive an EJECT value of Y.						
N	Indicates that CONCATENATE was not requested and more data remains for this transaction. Issue the request again, leaving the EJECT value at N to get the next detailed record. Continue to do so until you receive an EJECT value of Y.						
Y	Indicates that the data returned on this request is the last data for the transaction.						

Table 3-80. Returned Information in Output Data Block

TRCB	Description
TRODB	Translator output data block
DATALEN	The amount of data returned in DATA .
DATA	The formatted data extraction records. For more information, see "Transaction Store Data Extract Information Categories" on page 2-80.

API - Retrieve Transaction Image

This API function allows the application to retrieve the transaction image for the current transaction returned from the transaction detail API function. The current transaction is established using the 216 function request before an image is requested. Once the transaction detail record is returned, the following API request is made:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The only difference between this request and the request to get the detailed data is that the **ZFCBFUNC** value of 217 is used. The transaction image detail record is returned in the TRODB data buffer.

DATALEN indicates the amount of data returned in the **DATA** field of the TRODB. If the image is too large to fit in the TRODB buffer, the **EJECT** field of the TRCB has a value of X, indicating there is residual data. If this is the case, repeated API requests should be made (leaving the EJECT value of X) until the **EJECT** field is set to Y, indicating that no data remains.

API - Retrieve Transaction Acknowledgment Image

This API function allows the application to retrieve the transaction acknowledgment image for the current transaction returned from the transaction detail API function. The current transaction was established using the 216 function request before an acknowledgment image is requested. Once the transaction detail record has been returned, the following API request can be made:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The only difference between this request and the request to get the detailed data is that the **ZFCBFUNC** value of 218 is used. The transaction acknowledgment image detail record is returned in the TRODB data buffer. **DATALEN** indicates the amount of data returned in the **DATA** field of the TRODB. If the image is too large to fit in the TRODB buffer, the **EJECT** field of the TRCB has a value of X, indicating there is residual data. If this is the case, repeated API requests are made (leaving the EJECT value of X) until the **EJECT** field is set to Y indicating that no data remains.

API - Retrieve FA Image

This API function enables the application to retrieve the functional acknowledgment image for the current group returned from the transaction detail API function. The current group was established using the 216 function request before an acknowledgment image is requested. On return of the group detail record, the following API request is made:

FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The only difference between this request and the request to get the detailed data is that a **ZFCBFUNC** value of 219 is used. The functional acknowledgment image detail record is returned in the TRODB data buffer. **DATALEN** indicates the amount of data returned in the **DATA** field of the TRODB. If the image is too large to fit in the TRODB buffer, the **EJECT** field of the TRCB has a value of X indicating there is residual data. If this is the case, repeated API requests are made (leaving the **EJECT** value of X) until the **EJECT** field is set to Y, indicating that no data remains.

Communication Services

The functions provided in the communications application program interface (API) enable an application program to send or receive transaction data, files, and messages to and from trading partners. This section provides some general comments about the data flow and components of communication, followed by detailed data about each function in the communications API.

The transaction data or file data that is to be sent or received is not passed directly through an API parameter. Rather, it is provided indirectly as the name of an ordinary operating system (OS) sequential flat file that must contain the data to be sent, or be used to collect the data that is being received.

Note: In CICS, the data to be sent or received is restricted to a temporary storage queue.

Ordinarily, the functions that an API calls use an external file to contain the data. Therefore, if transaction data is to be sent to a trading partner, the transaction data must first be placed in an OS sequential file. The transaction can be placed in this sequential file in several ways, including:

- Writing an API program to request enveloping services or translation and enveloping services (see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27)
- The **PERFORM ENVELOPE**, **REENVELOPE**, or combination commands provided by the DataInterchange Utility are used
- The envelope and reenvelope functions provided by the Transaction Store facility are used.

If transactions are being received from a trading partner, the communication functions ask the appropriate network programs to collect the data in the specified file. Management reporting is called at this time to record the number of interchanges and the total number of bytes received. However, nothing is done with the transactions in the file until:

- A program using the API requests that the transactions in the file be deenveloped or deenveloped and translated (see “Deenvelope Function” on page 3-86 and “API - Translate File” on page 3-57).
- The PERFORM DEENVELOPE or combination commands provided by the DataInterchange Utility are used.
- The deenvelope functions provided by the Transaction Store facility are used.

There are a couple of levels of indirection between an application program requesting a communications API function and the network program that processes the request. Not all the programs involved in processing the request are necessarily written by or provided with the DataInterchange product. The path from an application program to a network program is:

1. The application program makes the communications API request by calling the appropriate STUB program (FXXZC, FXXZCBL, FXXZPLI, or FXXZASM).
2. DataInterchange invokes the communication component of DataInterchange (EDICM). EDICM reads various profiles, including the network profile (NETPROF). The network profile entry contains the names of the following programs:
 - The **Communication rtn.** DataInterchange provides communication routines for the networks directly supported by DataInterchange.
 - The **Network program.** The network program is not provided by DataInterchange, but by the network provider. For example, IEBASE is provided by the IBM Global Network, DSXMIT2 is provided by the GEIS network, and EDIBTCH is provided by the Harbinger In*Touch** Gateway. These programs have a defined interface. The purpose of the communication routine is to create this interface.
 - The **Message handler.** This is a program that processes responses from the network program so that results can be communicated to the original program making the API request. DataInterchange provides message handlers for the networks directly supported by DataInterchange. However, anyone can write a message handler for an internal network or one not directly supported by DataInterchange.

After validation is performed, EDICM invokes the communication routine to actually process the API request.

3. The communication routine is responsible for transforming the application API request into the proper requests as defined by the network program. The communication routine is the bridge between DataInterchange and the network, because it knows the interface on both ends and uses data provided by DataInterchange to build commands required by the network program. Once the commands are built, the network program is invoked.
4. The network program processes the commands and sends or receives data as directed by the commands. Once this is complete, the network program returns to the communication routine.
5. The communication routine invokes the message handler program to process the responses from the network. Control is returned to EDICM, which returns to the application.

For those about to write an API application and for those thinking of writing a network program or message handler, it is important to know that all communication requests made internally by DataInterchange utilities and facilities are made using the communications API functions described in this section.

Trading Partner Profile Data Block

One of the parameters for all functions of the communications API is the trading partner profile data block (TPPDB). This does not mean that an application making an API request is required to populate each field in the block with data. In fact, only a single field within the block has to be initialized: the **BLKLEN** field that contains the length of the block being provided.

The block is intended as an output block, but certain fields within the block might be provided. These fields are used by the communication routine when building commands for the network program. They are parameters that control or direct the network program. If your program does not provide values for these fields, corresponding values from the requestor profile entry or the trading partner profile entry retrieved by communications are used. Communications determines if you are providing values in this block by looking at the **TPNICKNM** field. If this field contains blanks, TPPDB is interpreted as the output block. If this field is not blank, TPPDB is interpreted as the input block and fields within the TPPDB override the values from the requestor profile member. The fields you can provide are described in Table 3-81.

Table 3-81 (Page 1 of 2). Trading Partner Fields Used by Communications

TPPDB	Description
BLKLEN	This field should have a value of 1532, which is the length of the trading partner profile data block.
BLKNME	Initialize with EDITPPDB.
TPNICKNM	The trading partner nickname that you want to send data to or receive data from. This can also be the nickname that should be used for getting parameters for the network program. For example, when transaction data is sent, more than one trading partner can be involved. There could be data in the file for multiple trading partners. If this field is left blank, communications uses the value from the TPNICKNM field of the CMCB, and the TPPDB becomes an output-only block. If this field contains a value, TPPDB is considered an input block and data from the following described fields is used in constructing the commands for the network program.
NETID	The ID of a network profile (NETPROF) entry that defines the network to which the request should be directed. This field is used as an input field only if the REQID and the NETID of the communication control block (CMCB) contain blanks.
ACCTNUM	The account number of the trading partner to which the current request applies.
USERID	The user ID of the trading partner to which the current request applies.
NETCLS	A code indicating any special status of the data being sent. The network specifies the acceptable codes.
NETCHG	A code indicating how charges are shared between sender and receiver. The network specifies the acceptable codes.
NETACK	A code indicating which network acknowledgments are requested. The network specifies the acceptable codes.
NETVCHK	A code indicating if the destination is to be verified before sending occurs. The network specifies the acceptable codes.
NETRETN	The number of days data is kept in a mailbox before it is purged. The network specifies the acceptable values.
NETEDIO	A code indicating whether you want EDI segments to be stored in the receiving file as separate records. The network specifies the acceptable codes.
NETEDIP	A code indicating if EDI data you receive is to have special EDI processing. The network specifies the acceptable codes.
STGFRMTO	A code indicating whether you want to use the storage format as known to the network program. The network specifies the acceptable codes.

Table 3-81 (Page 2 of 2). Trading Partner Fields Used by Communications

TPPDB	Description
MACHTYPE	If your trading partner is using the Personal Computer/Information Exchange (PC/IE) product to receive your data, enter 1. Otherwise, leave blank.
STGFRMT	A code indicating to the network how data is stored for nonstandard files. The network specifies the acceptable codes.
EOTID	The character that signifies the end of message text to the network. This applies only to sending or receiving free-form messages.
NETCMDS	The name of a member of the PDS that is allocated to EDINTCMD which contains network commands that DataInterchange should pass to the network.
TPDATALINE	The phone number to dial, to connect to your trading partner directly to pass data. This is the number needed by your computer to talk directly to your trading partner's computer.
TIMEOUT	The maximum allowable amount of time that the data line for communications can be idle without being dropped. The network specifies the acceptable values.

Common CMCB Output Fields

Some fields within the communication control block (CMCB) are established by the communications component of DataInterchange (EDICM). These fields are input data to the communication routine and output data for the application program making any communications API request. These fields define the capabilities of the network and are set based on the values found in the FSUPPORT network operation entry (NETOP profile) for the network. These fields are described in the following table and are not repeated in each of the API descriptions.

Table 3-82 (Page 1 of 2). Network Capability Fields in CMCB

CMCB	Description
FQUEUED	Set to Y if the network supports queuing functions. Queuing functions have been dropped from the documentation and will be dropped from support in the product with the next version of DataInterchange.
FMSGGS	Set to Y if the network supports free-form messages. The send free-form message API has been dropped from the documentation and will be dropped from support in the product with the next version of DataInterchange.
FFILE	Set to Y if the network supports nonstandard files.
FEDIX	Set to Y if the network supports X12 ISA/IEA interchanges.
FEDIE	Set to Y if the network supports EDIFACT UNB/UNZ interchanges.
FEDIU	Set to Y if the network supports BG/EG interchanges.
FEDIG	Set to Y if the network supports GS/GE interchanges.
FEDII	Set to Y if the network supports ICS/ICE interchanges.
FEDIT	Set to Y if the network supports STX/END interchanges.
FCANCEL	Set to Y if the network supports the CANCEL function.
FCLASS	Set to Y if the network supports user message classes.
FAACK	Set to Y if the network supports network acknowledgments.
FSYSMSG	Set to Y if the network supports system messages.
FRCVBTP	Set to Y if the network supports receiving by trading partner.
FRESTART	Set to Y if the network supports restart.

Table 3-82 (Page 2 of 2). Network Capability Fields in CMCB

CMCB	Description
FNOUSERID	Set to Y if the network supports account numbers only.
FACCTSEP	The character used to separate the account number and user ID.

Return Codes from COMM

For all the communications API requests, the overall result of the request is indicated by the return code (**ZCCBRC**) and extended return code (**ZCCBERC**) in the CCB. You can find specific information about all possible return codes in “Communication Return Codes” on page B-27. In general, the **ZCCBRC** field has the following values:

RC Explanation

0 The request was processed without error.

4 Return code 4 is a warning. The request was processed, but something unusual was noticed. Two extended return codes are:

ERC Explanation

1 The network being used does not have a network program defined in the network profile, indicating that any SEND/RECEIVE/CANCEL requests are meaningless.

1040 No data was returned on a receive request.

8 An error occurred with the request. Another request might not have the problem.

12 A severe error occurred. This usually indicates a system error. Another request probably will not succeed.

API - Send Transactions and Restart Send Transactions

The communications request to send transaction data is used to build all the commands necessary for a network program to send a file of interchanges to the appropriate trading partners. If a point-to-point network is being used, it is assumed that all the interchanges in the file are for the same trading partner. If a generalized network is being used (such as IBM Global Network or GEIS), the file might contain interchanges for multiple trading partners and the network uses information in the interchange header to route the interchange to the correct trading partner. Before the network program is invoked to send the data, the file is scanned by DataInterchange and the status of each interchange in the file is set to SEND STARTED.

The communications request to restart send transactions is valid only when the network supports restart and you have specified checkpoint-level recovery when initially sending the data. If an error causes the network processing to enter a restart situation during a send operation, the restart send can be used to restart and complete the send. Refer to the *Expedite Base/MVS Programming Guide* for more information on checkpoint recovery.

Note: The control blocks must be initialized the same as on the initial send except for NETOP, which is not required on restart.

Note: The restart send request is not supported in CICS.

After the network program returns, the status of each interchange is updated to SEND REQUESTED (return code of 0 from the network program), SENT WITH ERRORS (return code greater than 0 but less than 5 from the network program), or SEND REQUEST ERROR (return code not between 0 and 4), based on the degree of success of the network program. Updating of status is accomplished using the Status

Update API (“Status Update Services” on page 3-126). The DataInterchange Utility uses the send transaction data API whenever any of the following commands are requested:

- PERFORM SEND
- PERFORM ENVELOPE AND SEND
- PERFORM REENVELOPE AND SEND
- PERFORM TRANSLATE AND SEND

The DataInterchange Utility uses the restart send transactions API when the following command is requested:

- PERFORM RESTART SEND

The basic format of the API request to send standard data is:

FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)

The parameters for this API request are:

Parameter Description

SNB	The service name block identifying communications services.						
	<table> <tr> <th>Field</th><th>Value</th></tr> <tr> <td>ZSNBNAME</td><td>COMMBbbb to identify communications services</td></tr> <tr> <td>ZSNBPC</td><td>5 (number of parameters in the call)</td></tr> </table>	Field	Value	ZSNBNAME	COMMBbbb to identify communications services	ZSNBPC	5 (number of parameters in the call)
Field	Value						
ZSNBNAME	COMMBbbb to identify communications services						
ZSNBPC	5 (number of parameters in the call)						
CCB	The common control block used to initialize DataInterchange.						
FCB	The function control block with a ZFCBFUNC value of 211 for send transactions or 260 for restart send transactions.						
CMCB	The communications control block. For more information, see “Communication Interface Control Block (CMCB)” on page A-34.						
TPPDB	The trading partner profile data block. For more information, see “Trading Partner Profile Data Block” on page 3-107.						

CMCB Initialization

Some fields in the CMCB must be initialized before the API request is made. The following table contains the fields and descriptions of the initialization requirements.

Table 3-83 (Page 1 of 3). Initialization for Send Transaction Data

CMCB	Initialization Value
BLKLEN	254 (the length of the CMCB)
BLKNME	Initialize with any 8-character name desired. A recommended value is EDICMCB.
TPNICKNM	The trading partner nickname that is used when building the commands for the network program. This field is not necessary for a send transaction data request, because the network distributes the data based on the destination in the interchange header. This field is ignored if the TPNICKNM field of the TPPDB has a value.
NETID	The ID of network profile (NETPROF) entry that defines the network that is invoked. The value placed in this field is ignored if the REQID field has a value. When a REQID is provided, the NETID is taken from the requestor profile entry and this field is updated with that NETID value.
NETOP	See “Send Network Operation” on page 3-112.

Table 3-83 (Page 2 of 3). Initialization for Send Transaction Data

CMCB	Initialization Value								
REQID	The requestor profile ID value that is used for this request. The requestor profile identifies the mailbox you want to use to identify yourself. Numerous fields are taken from this profile entry while building the commands for the network.								
CLRFILE	<p>This field indicates whether you want DataInterchange to clear the file containing the transaction data at the end of the processing. The following are the possible values:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Clear if the data was sent successfully</td></tr> <tr> <td>U</td><td>Clear unconditionally</td></tr> <tr> <td>N</td><td>Do not clear (N is a suggested value and any value other than Y or U is treated like N)</td></tr> </table>	Value	Description	Y	Clear if the data was sent successfully	U	Clear unconditionally	N	Do not clear (N is a suggested value and any value other than Y or U is treated like N)
Value	Description								
Y	Clear if the data was sent successfully								
U	Clear unconditionally								
N	Do not clear (N is a suggested value and any value other than Y or U is treated like N)								
ACCTYP	This applies only if the SENDFILE network operation is being used. If so, ACCTYP should contain D, indicating that a trading partner account number and user ID are being supplied.								
DATATYP	<p>The type of file name supplied in the FILENAME field (if a FILENAME value is provided). The following are the possible values:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>A</td><td>FILENAME contains a data set name</td></tr> <tr> <td>D</td><td>FILENAME contains a ddname</td></tr> </table> <p>If FILENAME is not provided on input, both FILENAME and DATATYP are provided on output.</p> <p>Note: DATATYP is ignored in CICS.</p> <p>Special IEBASE considerations: Unless an entire data set name is supplied in the FILENAME field and an A is placed in the DATATYP field, a D must be placed in the DATATYP field. This is necessary so that DataInterchange can build the proper IEBASE INMSG FILEID parameter. If FILENAME is left blank, a D is still required in the DATATYP field.</p>	Value	Description	A	FILENAME contains a data set name	D	FILENAME contains a ddname		
Value	Description								
A	FILENAME contains a data set name								
D	FILENAME contains a ddname								
FILENAME	<p>The name of a file containing the data to be sent. This is either a ddname or data set name based on the value of DATATYP. If this field is blank, DataInterchange assigns a value using the following rules:</p> <p>Note: FILENAME must be a temporary storage queue in CICS.</p> <ul style="list-style-type: none"> • The Trans data queue field from the network profile member is used as a starting point. If this field is blank, a default value of QDATA is used. • If the network operation is SENDEDI, an E is appended to the ddname. For example, the QDATA default name becomes QDATAE. If the ddname is already 8 characters long, the last character is overlaid with an E. • If the network operation is SENDUCS, a U is appended to the ddname. For example, the QDATA default name becomes QDATAU. If the ddname is already 8 characters long, the last character is overlaid with a U. <p>Special IEBASE considerations: Unless an entire data set name is supplied in the FILENAME field and an A is placed in the DATATYP field, a D must be placed in the DATATYP field. This is necessary so that DataInterchange can build the proper IEBASE INMSG FILEID parameter. If FILENAME is left blank, a D is still required in the DATATYP field.</p>								

Table 3-83 (Page 3 of 3). Initialization for Send Transaction Data

CMCB	Initialization Value
DCIND	<p>A delivery class for the data being sent. This class should contain one of the following values:</p> <ul style="list-style-type: none"> • Blank for general priority • P for high priority <p>The value is not interpreted by DataInterchange but is handled and understood by the network program.</p>
ACKIND	Indicates the type of acknowledgment wanted. See the definition of this field in "Communication Interface Control Block (CMCB)" on page A-34 for a list of values.
ENAME	The message user class that should be associated with each interchange in the file being sent. See "Default Message User Class" on page 3-114 for the default values assigned if a specific value is not supplied.
MSGNAME	The message name that should be associated with each interchange in the file being sent. See "Default Message Name" on page 3-114 for the default values assigned if a specific value is not supplied.

Send Network Operation

The function used to invoke communications indicates in general what the application is requesting, but it is the network operation (NETOP) that specifies exactly what commands should be built for the network program to process. The function of sending transaction data has four possible network operations that can be used:

- SENDX12
- SENDEDI
- SENDUCS
- SENDFILE

In previous versions of DataInterchange, the networks required a different command based on the type of data being sent. If the data being sent had ISA/IEA interchange headers, SENDX12 was required. If the data being sent had UNB/UNZ or STX/END interchange headers, SENDEDI was needed. If the data being sent contained BG/EG interchange headers, SENDUCS was required. If the data being sent contained ICS/ICE or GS/GE headers, the SENDFILE command was required. All the interchanges in the file had to be directed to the trading partner specified in the **TPNICKNM** field of the CMCB.

Separation of data based on the envelope type being used is no longer required by the networks directly supported by DataInterchange. However, DataInterchange and the DataInterchange utilities and facilities still set the network operator based on the type of data contained in the file.

Networks not directly supported by DataInterchange might still require the distinction. However, for the networks supported by DataInterchange the SENDEDI, SENDX12, and SENDUCS network operations all generate the same commands for the network program. SENDFILE is still unique because SENDFILE indicates the entire file should be sent to the trading partner without any interrogation required by the network program. This process is still necessary for the IBM Global Network if the ICS/ICE or GS/GE headers are used. These headers are not supported as transaction data headers and, therefore, cannot be parsed like the other interchange headers to determine the destination trading partner. Therefore, when using the API to send transaction data, the network operation used depends on the type of data being sent.

However, if you know the network that you are using does not distinguish between types of data, you can use SENDX12, SENDEDI, or SENDUCS. Use SENDFILE only if you do not want the network to parse

the file looking for interchanges, but just want the entire file sent to the trading partner whose name is contained in the **TPNICKNM** field of the CMCB.

Note: The NETOP information is not required for the restart send request.

Send Transactions Return Information

The return code (**ZCCBRC**) and extended return code (**ZCCBERC**) fields of the CCB indicate whether the request to send transaction data was successful. Numerous errors are possible. These error conditions are explained in “Communication Return Codes” on page B-27. Fields within the communications control block (CMCB) that are returned are documented in Table 3-84.

Table 3-84 shows the fields in the CMCB that are returned on a request to send transaction data.

Table 3-84 (Page 1 of 2). Fields Returned from 211 Request

CMCB	Description
SEQNUM	The network sequential number assigned to this transmission. The number is maintained in the network profile entry <i>Network sequence</i> field and is incremented on each request to send data. This number might be important later if you want to RECALL (CANCEL) this transmission.
ENAME	If a value was not supplied as input but a value does exist in the requestor profile entry <i>Message user class</i> field, the <i>Message user class</i> field value is returned.
ACKIND	If a value was not supplied as input, the value from the <i>Net acknowledgment</i> field of the requestor profile or the <i>Net acknowledgment</i> field of the trading partner profile is returned.
DATATYP	<p>If a value was not supplied as input, D is returned for generalized networks and A is returned for point-to-point networks, indicating the type of value in the FILENAME field.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. When using point-to-point networks, the DATATYP and FILENAME fields normally indicate that a data set name was used. However, if a trading partner was not provided with the request, the ddname taken from the <i>Trans data queue</i> field of the network profile is returned. 2. DATATYP is ignored in CICS.
FILENAME	<p>If a value was not supplied as input, the ddname is returned for generalized networks and the data set name is returned for point-to-point networks.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. When using point-to-point networks, the DATATYP and FILENAME fields indicate that a data set name was used. However, if a trading partner was not provided with the request, the ddname taken from the <i>Trans data queue</i> field of the network profile is returned. 2. FILENAME must be a temporary storage queue in CICS.
NPSSCDE	<p>Start session response code.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>
NPESCDE	<p>End session response code.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>

Table 3-84 (Page 2 of 2). Fields Returned from 211 Request

CMCB	Description
NPERRCD	<p>The most severe error code. The value in this field is included in the VN1015 error message when errors are met.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>
NPSEVER	<p>The severity of the most severe error. The value in this field is included in the VN1015 error message when errors are met.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>

Default Message User Class

The message user class is a code that trading partners agree to use for identifying classes of information to be sent or received. It allows users to select one kind of information from a mailbox that might hold various kinds of data. The message user class is assigned to an interchange when the interchange is sent by supplying a value in the **ENAME** field of the communications control block (CMCB).

If a value is not supplied, the message user class from the requestor profile member is used. If a message user class is not present in the requestor profile, a default is based on the type of interchange. The defaults assigned are:

Default	Description
ISA, ICS or GS envelopes	If not supplied, the message user class is set to #E2.
BG envelopes	If not supplied, the message user class is set to #EC.
UNB envelopes	If not supplied, the message user class is taken from the UNB14 (APRF) field. If UNB14 field is not present or contains blanks, a default value of #EE is used.
STX envelopes	If not supplied, the message user class is taken from the STX11 (APRF) field. If the STX11 field is not present or contains blanks, a default value of #EU is used.

Default Message Name

The message name is an arbitrary value that can be assigned to an interchange when the interchange is sent. It is part of an alternate key that can later be used to update the status of an interchange. It can also be used if it is necessary to recall an interchange. The **MSGNAME** field of the communications control block (CMCB) contains the message name that should be assigned to each interchange within the file being sent. If a message name is not supplied, a default assignment based on the type of interchange is made as follows:

Default	Description
ISA or ICS envelopes	If not supplied, the message name is the last 8 bytes of the interchange control number.
GS envelopes	If not supplied, the message name is the group control number, left-justified and padded with blanks.

BG envelopes	If not supplied, the message name is the interchange control number, left-justified and padded with blanks.
UNB envelopes	If a message name is not supplied, the interchange control number is used, left-justified and padded with blanks.
STX envelopes	If a message name is not supplied, the interchange control number is used, left-justified and padded with blanks.

API - Send Files

The communications request to send files is used to build all the commands necessary for a network program to send a file containing nonstandard data to a particular trading partner. A **PERFORM** command is not provided with the DataInterchange Utility to allow a file of nonstandard data to be sent. The DataInterchange Interactive Entry Facility (IEF) allows sending files and messages. IEF does so by using the API described in this section.

The basic format of the API request to send a file is:

FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)

The parameters for this API request are:

Parameter Description

SNB	The service name block identifying translation services.						
	<table> <tr> <th>Field</th><th>Value</th></tr> <tr> <td>ZSNBNAME</td><td>COMMbbbb to identify communications services</td></tr> <tr> <td>ZSNBPC</td><td>5 (number of parameters in the call)</td></tr> </table>	Field	Value	ZSNBNAME	COMMbbbb to identify communications services	ZSNBPC	5 (number of parameters in the call)
Field	Value						
ZSNBNAME	COMMbbbb to identify communications services						
ZSNBPC	5 (number of parameters in the call)						
CCB	The common control block used to initialize DataInterchange.						
FCB	The function control block with a ZFCBFUNC value of 221.						
CMCB	The communications control block. For more information, see "Communication Interface Control Block (CMCB)" on page A-34.						
TPPDB	The trading partner profile data block. For more information, see "Trading Partner Profile Data Block" on page 3-107.						

CMCB Initialization

Some fields in the CMCB must be initialized before the API request is made. The following table contains the fields and descriptions of the initialization requirements.

Table 3-85 (Page 1 of 2). Initialization for Send Transaction Data

CMCB	Initialization Value
BLKLEN	254 (the length of the CMCB)
BLKNME	EDICMCB
TPNICKNM	The trading partner nickname to which the file should be sent. This field is ignored if the TPNICKNM field of the TPPDB has a value.
NETID	The network that is invoked. The value placed in this field is ignored if the REQID field has a value. When a REQID is provided, the NETID is taken from the requestor profile entry and this field is updated with that NETID value.
NETOP	Initialize with a value of SENDFILE.

Table 3-85 (Page 2 of 2). Initialization for Send Transaction Data

CMCB	Initialization Value								
REQID	The requestor profile ID value that should be used for this request. The requestor profile identifies the mailbox that you want to use to identify yourself. Numerous fields are taken from this profile entry while building the commands for the network.								
CLRFILE	<p>This field indicates whether you want DataInterchange to clear the file at the end of the processing. The following are the possible values:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Clear if the data was sent successfully</td></tr> <tr> <td>U</td><td>Clear unconditionally</td></tr> <tr> <td>N</td><td>Do not clear (N is a suggested value and any value other than Y or U is treated like N)</td></tr> </table>	Value	Description	Y	Clear if the data was sent successfully	U	Clear unconditionally	N	Do not clear (N is a suggested value and any value other than Y or U is treated like N)
Value	Description								
Y	Clear if the data was sent successfully								
U	Clear unconditionally								
N	Do not clear (N is a suggested value and any value other than Y or U is treated like N)								
ACCTYP	Initialize with D indicating that a trading partner account number and user ID are being supplied.								
DATATYP	<p>The type of file name supplied in the FILENAME field. The following are the possible values:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>A</td><td>FILENAME contains a data set name</td></tr> <tr> <td>D</td><td>FILENAME contains a ddname</td></tr> </table> <p>Note: DATATYP is ignored in CICS.</p>	Value	Description	A	FILENAME contains a data set name	D	FILENAME contains a ddname		
Value	Description								
A	FILENAME contains a data set name								
D	FILENAME contains a ddname								
FILENAME	<p>The name of a file containing the data to be sent. This is either a ddname or data set name based on the value of DATATYP.</p> <p>Note: FILENAME must be a temporary storage queue in CICS.</p>								
DCIND	<p>A delivery class for the data being sent. This class should contain one of the following values:</p> <ul style="list-style-type: none"> • Blank for general priority • P for high priority • I for express delivery <p>The value is not interpreted by DataInterchange, but is handled and understood by the network program.</p>								
ACKIND	Indicates the type of acknowledgment wanted. See the definition of this field in “Communication Interface Control Block (CMCB)” on page A-34 for a list of values.								
ENAME	The message user class that should be associated with the file. If a value is not supplied, the MESSAGE USER CLASS field from the requestor profile entry is used.								
MSGNAME	The message name to be associated with the file.								

Send Files Return Information

The return code (**ZCCBRC**) and extended return code (**ZCCBERC**) fields of the CCB indicate if the request to send a file was successful. Numerous errors are possible. These error conditions are explained in “Communication Return Codes” on page B-27. Fields within the communications control block (CMCB) that are returned are documented in Table 3-86 on page 3-117. Table 3-86 on page 3-117 shows the fields in the CMCB that are returned on a request to send a file.

Table 3-86. Fields Returned from 221 Request

CMCB	Description
SEQNUM	The network sequential number assigned to this transmission. The number is maintained in the network profile entry <i>Network sequence</i> field and is incremented on each request to send data. This number might be important later if you want to RECALL (CANCEL) this transmission.
ENAME	If a value was not supplied as input, but a value does exist in the requestor profile <i>Message user class</i> field, its value is returned.
ACKIND	If a value is not supplied as input, the value from the requestor profile entry <i>Net acknowledgment</i> field or the trading partner profile <i>Net acknowledgment</i> field is returned.
NPSSCDE	Start session response code. Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPESCDE	End session response code. Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPERRCD	The most severe error code. The value in this field is included in the VN1015 error message when errors are met. Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPSEVER	The severity of the most severe error. The value in this field is included in the VN1015 error message when errors are met. Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.

API - Receive and Restart Receive

The DataInterchange Utility uses the receive API when any of the following commands are requested:

- PERFORM RECEIVE
- PERFORM RECEIVE AND DEENVELOPE
- PERFORM RECEIVE AND TRANSLATE

The communications request to restart receive transactions is valid only when the network supports restart and you have specified checkpoint level recovery when initially receiving the data. If an error causes the network processing to enter a restart situation during a receive operation, the restart receive can be used to restart and complete the receive. Refer to the *Expedite Base/MVS Programming Guide* for more information on checkpoint recovery.

Note: The control blocks must be initialized the same as on the initial receive except for NETOP, which is not required for restart.

Note: The restart receive request is not supported in CICS.

The DataInterchange Utility uses the restart receive API when the following command is requested:

- PERFORM RESTART RECEIVE

The format of the API request for receiving is:

FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)

The parameters for this API request are:

Parameter	Description
-----------	-------------

SNB	The service name block identifying communication services.
-----	--

Field	Value
ZSNBNAME	COMMbbbb to identify communications services
ZSNBPC	5 (number of parameters in the call)

CCB	The common control block used to initialize DataInterchange.
-----	--

FCB	The function control block with a ZFCBFUNC value of 232 for receive or 261 for restart receive.
-----	--

CMCB	The communications control block. For more information, see “Communication Interface Control Block (CMCB)” on page A-34.
------	--

TPPDB	The trading partner profile data block. For more information, see “Trading Partner Profile Data Block” on page 3-107.
-------	---

CMCB Initialization

Some fields in the CMCB must be initialized before the API request is made. The following table contains the fields and descriptions of the initialization requirements.

Table 3-87 (Page 1 of 2). Initialization for Receive

CMCB	Initialization Value						
BLKLEN	254 (the length of the CMCB)						
BLKNME	EDICMCB						
TPNICKNM	The trading partner nickname used when building the commands for the network program. This is not a necessary field for a receive. If supplied, only data from this trading partner is requested. If not supplied, data from all trading partners is requested. This field is also ignored if the TPNICKNM field of the TPPDB has a value.						
NETID	The network that is invoked. The value placed in this field is ignored if the REQID field has a value. When a REQID is provided, the NETID is taken from the requestor profile entry and this field is updated with that NETID value.						
NETOP	See “Receive Network Operation” on page 3-119.						
REQID	The requestor profile ID value that is used for this request. The requestor profile identifies the mailbox from which you want to receive data.						
ACCTYP	If a TPNICKNM is provided, this field contains D. Otherwise, it should contain a blank.						
DATATYP	<p>The type of file name supplied in the FILENAME field (if a FILENAME value is provided). The following are the possible values:</p> <table><thead><tr><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>A</td><td>FILENAME contains a data set name</td></tr><tr><td>D</td><td>FILENAME contains a ddname</td></tr></tbody></table> <p>If FILENAME is not provided on input, both the FILENAME and DATATYP are provided on output.</p> <p>Note: DATATYP is ignored in CICS.</p>	Value	Description	A	FILENAME contains a data set name	D	FILENAME contains a ddname
Value	Description						
A	FILENAME contains a data set name						
D	FILENAME contains a ddname						

Table 3-87 (Page 2 of 2). Initialization for Receive

CMCB	Initialization Value
FILENAME	The name of a file for receiving the data. This is either a ddname or data set name based on the value of DATATYP . If this field is blank, DataInterchange uses the value from the requestor profile member <i>Receive file name</i> field and forces the DATATYP field to D. FILENAME is a mandatory field when the RECVFILE network operation is used. Note: FILENAME must be a temporary storage queue in CICS.
ENAME	The message user class used to identify the data received. If a value is not supplied, the <i>Message user class</i> field from the requestor profile entry is used. The TPNICKNM and ENAME fields form the selection criteria indicating what data from the mailbox is desired.
RECVTYP	Set to G if only the first file that meets the selection criteria (TPNICKNM and ENAME combination) should be returned. Set to blank if all files meeting the criteria should be returned.
RESRECL	This field controls what the network program does when the data received has records larger than the logical record length (LRECL) of the file to which the data is being written (FILENAME field). An S indicates that the records are split into smaller records with a maximum size of LRECL. A blank indicates the network program stops with an A03 system abend.

Receive Network Operation

The function used to invoke communications indicates in general what the application is requesting, but it is the network operation (NETOP) that indicates exactly what commands should be built for the network program to process. The function of receiving data has four possible network operations that can be used:

- RECVX12
- RECVEDI
- RECVUCS
- RECVFILE

None of the networks currently supported by DataInterchange allow a distinction to be made when asking for data as far as the type of data wanted is concerned. However, DataInterchange ships support for these four network operations, because it is possible to single out the type of interchanges you want to receive in the future. The RECVEDI, RECVX12, and RECVUCS network operations all generate the same command for the network program. They also all indicate that the intent is to receive transaction data or tell the network program to interpret the data received as transaction data. When using the API for receiving, the network operation used depends on the type of data that is intended to be received. If you know the network that you are using does not distinguish between types of data, you can use RECVX12, RECVEDI, or RECVUCS. Use RECVFILE only if you do not want the network to parse the data being received as if it were transaction data.

Note: The NETOP information is not required for the restart receive request.

Receive Return Information

The return code (**ZCCBRC**) and extended return code (**ZCCBERC**) fields of the CCB indicate if the receive request was successful. Numerous errors are possible. These error conditions are explained in “Communication Return Codes” on page B-27. Fields within the communications control block (CMCB) also are returned. These are documented in Table 3-88 on page 3-120. Table 3-88 on page 3-120 shows the fields in the CMCB that are returned on a request to receive data.

Table 3-88 (Page 1 of 2). Fields Returned from 232 Request

CMCB	Description
DATATYP	<p>If a value was not supplied as input, D is returned for generalized networks and A is returned for point-to-point networks indicating the type of value in the FILENAME field.</p> <p>Note: When using point-to-point networks, the DATATYP and FILENAME fields normally indicate that a data set name was used. If a trading partner was not provided with the request, the ddname taken from the network profile entry <i>Trans data queue</i> field is returned.</p>
FILENAME	<p>If a value was not supplied as input, the ddname is returned for generalized networks and the data set name is returned for point-to-point networks. The ddname returned is the <i>Receive file name</i> field from the requestor profile entry if a RECVX12, RECVEDI, or RECVUCS network operation is being used. For other network operations, the FILENAME is a required input parameter.</p> <p>Note: When using point-to-point networks, the DATATYP and FILENAME fields normally indicate that a data set name was used. If a trading partner was not provided with the request, the ddname taken from the network profile entry <i>Trans data queue</i> field is returned.</p>
TPNICKNM	<p>If a value is not supplied on input indicating a request to receive data from any trading partner, on output this contains the trading partner nickname for the first data received if the response data created by the network program provides enough information for this to be determined.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>
FILERCVD	<p>Contains a value of Y if data was received. The message handler sets this flag if it is determined by processing the network responses that data was received.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>
ENAME	<p>The message user class associated with the first file received is returned if the network program provides this information. This field is returned by the message handler while processing the responses generated by the network program.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>
MSGNAME	<p>The message name associated with the first file received is returned if the network program provides this information. This field is returned by the message handler while processing the responses generated by the network program.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>
NPSSCDE	<p>Start session response code.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>
NPESCDE	<p>End session response code.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>

Table 3-88 (Page 2 of 2). Fields Returned from 232 Request

CMCB	Description
NPERRCD	<p>The most severe error code. The value in this field is included in the VN1015 error message when errors are encountered.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>
NPSEVER	<p>The severity of the most severe error. The value in this field is included in the VN1015 error message when errors are encountered.</p> <p>Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.</p>

API - Cancel

The communications request to cancel is used to build all the commands necessary for a network program to cancel (or recall) data previously sent. If a file is sent to a trading partner by mistake, it can be canceled up to the time that the trading partner receives the data. Not all networks support the cancel request. The following description is directed at the IBM Global Network. There is no PERFORM command provided with the DataInterchange Utility that supports the CANCEL API.

The DataInterchange Interactive Entry Facility (IEF) allows recalling of files and messages that were previously sent using IEF. The recall is done using the API described in this section.

The basic format of the API request to CANCEL is:

FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)

The parameters for this API request are:

Parameter Description

SNB	The service name block identifying communications services.	
	Field	Value
	ZSNBNAME	COMMbbbb to identify communications services
	ZSNBPC	5 (number of parameters in the call)
CCB	The common control block used to initialize DataInterchange.	
FCB	The function control block with a ZFCBFUNC value of 233.	
CMCB	The communications control block. For more information, see "Communication Interface Control Block (CMCB)" on page A-34.	
TPPDB	The trading partner profile data block. For more information, see "Trading Partner Profile Data Block" on page 3-107.	

CMCB Initialization

Some fields in the CMCB must be initialized before the API request is made. The following table contains the fields and descriptions of the initialization requirements.

Table 3-89. Initialization for Cancel

CMCB	Initialization Value
BLKLEN	254 (the length of the CMCB)
BLKNME	EDICMCB
TPNICKNM	The trading partner nickname from which files are being recalled. This field is ignored if the TPNICKNM field of the TPPDB has a value.
NETID	The network that is invoked. The value placed in this field is ignored if the REQID field has a value. When a REQID is provided, the NETID is taken from the requestor profile entry and this field is updated with that NETID value.
NETOP	CANCEL
REQID	The requestor profile ID value used for this request. The requestor profile identifies the mailbox that you want to use to identify yourself. Numerous fields are taken from this profile entry while building the commands for the network.
ACCTYP	Initialize with D, indicating a trading partner account number and user ID is being supplied.
ACKIND	Indicates the type of acknowledgment wanted regarding the cancellation. See the definition of the field in "Communication Interface Control Block (CMCB)" on page A-34.
DCIND	This must match the delivery class used when the file was originally sent.
ENAME	The message user class associated with the file when it was originally sent. If a value is not supplied, the <i>Message user class</i> field from the requestor profile entry is used.
MSGNAME	The message name associated with the file when it was originally sent.
SEQNUM	The sequential number assigned to the file when it was originally sent.
CANSDD	Cancellation start date in YYMMDD format. Initialize with blanks if not used.
CANST	Cancellation start time in HHMMSS format. Initialize with blanks if not used.
CANED	Cancellation end date in YYMMDD format. Initialize with blanks if not used.
CANET	Cancellation end time in HHMMSS format. Initialize with blanks if not used.
TMZONE	Identifies the time zone used for CANSDD, CANST, CANED, and CANET. L indicates local time. G indicates Greenwich mean time.

Cancel Return Information

The return code (**ZCCBRC**) and extended return code (**ZCCBERC**) fields of the CCB indicate whether the request to CANCEL was successful. Numerous errors are possible. These error conditions are explained in "Communication Return Codes" on page B-27. Fields within the communications control block (CMCB) that are returned are documented in Table 3-90. Table 3-90 shows the fields in the CMCB that are returned on a CANCEL request.

Table 3-90 (Page 1 of 2). Fields Returned from 233 Request

CMCB	Description
NPSSCDE	Start session response code. Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.

Table 3-90 (Page 2 of 2). Fields Returned from 233 Request

CMCB	Description
NPESCDE	End session response code. Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPERRCD	The most severe error code. The value in this field is included in the VN1015 error message when errors are encountered. Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPSEVER	The severity of the most severe error. The value in this field is included in the VN1015 error message when errors are encountered. Note: This field is returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.

API - Return Filename

The API request to return the filename is issued to determine the name of the file associated with the network for writing transaction data. The API request is issued internally during any translation or enveloping operation. The network program (for example, IEbase or DSXMIT2) is not involved in processing this API request. The request is processed directly through the communication routine.

The basic format of the API request to return the filename is:

FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)

The parameters for this API request are:

Parameter Description

SNB	The service name block identifying communications services.	
	Field	Value
	ZSNBNAME	COMMBbbb to identify communications services
	ZSNBPC	5 (number of parameters in the call)
CCB	The common control block used to initialize DataInterchange.	
FCB	The function control block with a ZFCBFUNC value of 300.	
CMCB	The communications control block. For more information, see "Communication Interface Control Block (CMCB)" on page A-34.	
TPPDB	The trading partner profile data block. For more information, see "Trading Partner Profile Data Block" on page 3-107.	

CMCB Initialization

Some fields in the CMCB must be initialized before the API request is made. The following table contains the fields and descriptions of the initialization requirements.

Table 3-91. Initialization for Query Filename

CMCB	Initialization Value
BLKLEN	254 (the length of the CMCB)
BLKNME	EDICMCB
TPNICKNM	The trading partner nickname for which the current interchange is being created. This is important only when a point-to-point network is being used because the data set name used for transaction data is constructed using the TPNICKNM field.
NETID	The network for which information is wanted. The value placed in this field is ignored if the REQID field has a value. When a REQID is provided, the NETID is taken from the requestor profile entry and this field is updated with that NETID value.
REQID	The requestor profile ID value used for this request.

Return Filename Return Information

The return code (**ZCCBRC**) and extended return code (**ZCCBERC**) fields of the CCB indicate whether the request was successful. Numerous errors are possible. These error conditions are explained in “Communication Return Codes” on page B-27. Fields within the communications control block (CMCB) that are returned are documented in Table 3-92. Table 3-92 shows the fields in the CMCB that are returned on a request to return the file name associated with the network.

Table 3-92. Fields Returned from 300 Request

CMCB	Description
DATATYP	D is returned for generalized networks, and A is returned for point-to-point networks, indicating the type of value in the FILENAME field.
FILENAME	The ddname associated with the network (<i>Trans data queue</i> field in the network profile entry) is returned for generalized networks, and the data set name associated with the trading partner is returned for point-to-point networks.

Internal Calls

The following are internal API calls.

API - Queue Standard Data

The API request to queue standard data is an internal only API function issued by the translator or enveloper when an interchange is complete and needs to be written to a file associated with the network. The network program is not called in the processing of this request. Not until the request to send the transaction data (“API - Send Transactions and Restart Send Transactions” on page 3-109) is received is the network program involved. See “Sending Transaction Data” on page 3-82 for considerations about the sending of the data and “API - Translate File” on page 3-57 and “Envelope Function” on page 3-72 for information about how to provide the name of the file to which the transaction data should be written.

API - Process Network Acknowledgments

The communications request to process network acknowledgments is used to build all the commands necessary to instruct the network program to return any network acknowledgment data. The DataInterchange Utility uses this API when the PERFORM STATUS UPDATE command is requested.

The basic format of the API request to process network acknowledgments is:

FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)

The parameters for this API request are:

Parameter	Description
-----------	-------------

SNB	The service name block identifying communications services.
-----	---

Field	Value
-------	-------

ZSNBNAME	COMMBbbb to identify communications services
----------	--

ZSNBPC	5 (number of parameters in the call)
--------	--------------------------------------

CCB	The common control block used to initialize DataInterchange.
-----	--

FCB	The function control block with a ZFCBFUNC value of 252.
-----	---

CMCB	The communications control block. For more information, see "Communication Interface Control Block (CMCB)" on page A-34.
------	--

TPPDB	The trading partner profile data block. For more information, see "Trading Partner Profile Data Block" on page 3-107.
-------	---

CMCB Initialization

Some fields in the CMCB must be initialized before the API request is made. The following table contains the fields and descriptions of the initialization requirements.

Table 3-93. Initialization for Process Network Acknowledgments

CMCB	Initialization Value
BLKLEN	254 (the length of the CMCB)
BLKNME	EDICMCB
NETID	The network that is invoked. The value placed in this field is ignored if the REQID field has a value. Note: This function must be invoked for each network individually. The PERFORM STATUS UPDATE command issues the API request for each network defined in the network profile unless the NETID keyword is used, indicating a specific network is wanted.
NETOP	If a file of network acknowledgments has already been received and just needs to be processed, use a NETOP value of N0-NETOP. Otherwise, leave the field blank and the communication routine invokes the network program to return the acknowledgment data.
REQID	The requestor profile ID value used for this request. If specified, network acknowledgments are only requested and processed for this specific requestor. If a requestor is not provided, acknowledgments are requested and processed for every requestor that is defined for the network identified by NETID.

There are no output fields in the CMCB for this request. The return code (**ZCCBRC**) and extended return code (**ZCCBERC**) indicate whether the request was successful. See "Communication Return Codes" on page B-27 for all the possible return codes from communications.

Status Update Services

The status update service enables updating of the status of an interchange as it progresses from being enveloped, to sent, to received by the trading partner. The status update service is used by the communication routine and message handling programs whenever an interchange is sent and during the processing of network acknowledgments. Certain characteristics of an interchange become known only when the interchange is sent. Therefore, the status update service also enables other fields within the interchange to be set when status is updated. These are established using the update status data block (USDB), which is described in “Status Update Data Block” on page 3-131.

Six function codes are supported by the status update service. All the functions provide the same support, which is to update the status (and possibly other fields provided in the USDB) of an interchange. The different function codes are used to indicate how the interchange whose status should be updated is identified. The key for an interchange is:

- Trading partner nickname – 16 characters, left-justified with trailing blanks
- Direction of the interchange – S for send, R for receive
- Interchange receiver ID as taken from the interchange header – 35 characters, left-justified with trailing blanks
- Interchange control number from the interchange header – 14 characters, right-justified with leading zeros

Two alternate keys for an interchange (established at the time an interchange is sent) also can be used when processing network acknowledgments. One of these is a unique 8-character value associated with the interchange by the network program. The other alternate key might not be a unique value and consists of the following fields:

- Trading partner nickname – 16 characters, left-justified with trailing blanks
- Message name – 8 characters, left-justified with trailing blanks
- Message user class – 8 characters, left-justified with trailing blanks
- Message sequence number – 5 characters, right-justified with leading zeros

The difference in key values that can be supplied to the status update service depends on whether the primary key or one of the alternate keys is being used and on how the trading partner is identified. The format of the different key values is provided in the sections that follow.

API - Status Update

The basic format of the API request to update status is:

```
FXXZccc(SNB,CCB,FCB,USKB,status[,USDB['reqid']])
```

The parameters for this API request are:

Parameter	Description
-----------	-------------

SNB	The service name block identifying status update services.
-----	--

Field	Value
-------	-------

ZSNBNAME	TRANSSRV to identify status update services
----------	---

ZSNBPC	The number of parameters in the call: 5 parameters in the call if USDB not provided 6 parameters in the call if USDB provided 7 parameters in the call if reqid is provided
--------	--

CCB	The common control block used to initialize DataInterchange.
FCB	<p>The function control block with a ZFCBFUNC value that identifies the type of key provided. The possible values are:</p> <ul style="list-style-type: none"> 210 Full envelope key with the trading partner identified by account number and user ID 211 The transaction handle value of one of the transactions within the interchange 212 Alternate key provided with the trading partner identified by account number and user ID 213 Full envelope key with the trading partner identified by interchange qualifier and interchange ID 214 Alternate key provided with the trading partner identified by interchange qualifier and interchange ID 215 Full envelope key with the trading partner identified by the trading partner nickname
USKB	The key block with a format based on the value of ZFCBFUNC. The values of ZFCBFUNC are defined above next to FCB. There are six formats to choose from. For more information, see "Trading Partner Nickname" on page 3-128, "Qualifier and Interchange ID" on page 3-129, "Account Number and User ID" on page 3-129, "Transaction Handle" on page 3-129, "Alternate Key 1 - Account Number and User ID" on page 3-130, and "Alternate Key 2 - Interchange Qualifier and ID" on page 3-130.
status	<p>The new status for the interchange. This is the address of a 2-character field containing one of the following values:</p> <ul style="list-style-type: none"> 41 Send error 42 Send request error 43 Not sent, network error 46 Send started 48 Send requested 49 Sent to network 50 Accepted by the network 51 Delivered by the network 52 Purged by the network 53 Recall requested from network 54 Recall request error 55 Recalled from network
USDB	The optional update status data block (see "Status Update Data Block" on page 3-131).
reqid	If this parameter is supplied, the management reporting component of DataInterchange is called to update the statistics on the number of bytes sent from the specified requestor ID (applies only if status is 48 or going from 48 to some other status). This is an optional parameter.

Status Update Return Codes

The results of a status update request are posted in the return code and extended return code fields of the common control block. The following are possible values:

Return Code	Explanation								
0	Status update was successful.								
8	Status update failed. Extended return codes are:								
	<table><tr><th>ERC</th><th>Explanation</th></tr><tr><td>210</td><td>No interchange was found</td></tr><tr><td>211</td><td>Internal program error</td></tr><tr><td>212</td><td>Error attempting to update the database</td></tr></table>	ERC	Explanation	210	No interchange was found	211	Internal program error	212	Error attempting to update the database
ERC	Explanation								
210	No interchange was found								
211	Internal program error								
212	Error attempting to update the database								

Full Envelope Key

There are three ways to identify a trading partner when using the full envelope key:

- Trading partner nickname, which identifies the trading partner profile member
- Interchange qualifier and interchange ID fields, which are used to search the trading partner profile members to find an entry with matching values
- Account number and user ID fields, which are used to search the trading partner profile members to find an entry with matching values

The formats for these values are shown in the following tables.

Trading Partner Nickname

Table 3-94 shows the format for a full key using a trading partner nickname. This is the update status key block that should be used when the ZFCBFUNC value is 215.

Table 3-94. Full Key Using Trading Partner Nickname

Name	Format	Size	Offset	Description
TPNICKNM	CHAR	16	0	Trading partner nickname for an entry in the trading partner profile. Left-justified with trailing blanks.
FILLER	CHAR	48	16	Filler – initialize with blanks.
DIR	CHAR	1	64	Direction of interchange. Should have value of S.
INTCTLNO	CHAR	14	65	Interchange control number, right-justified with leading zeros.
RECEIVER	CHAR	35	79	Interchange receiver ID value as it was sent in the interchange with trailing blanks.

Qualifier and Interchange ID

Table 3-95 shows the format for full a key using a qualifier and interchange ID. This is the update status key block that should be used when the ZFCBFUNC value is 213.

Table 3-95. Full Key Using Interchange ID and Qualifier

Name	Format	Size	Offset	Description
QUALIFIER	CHAR	4	0	Interchange qualifier as it was sent in the interchange with trailing blanks.
RECEIVER	CHAR	35	4	Interchange receiver ID value as it was sent in the interchange with trailing blanks.
INTCTLNO	CHAR	14	39	Interchange control number, right-justified with leading zeros.

Account Number and User ID

Table 3-96 shows the format for a full key using an account number and user ID. This is the update status key block that should be used when the ZFCBFUNC value is 210.

Table 3-96. Full Key Using Account Number and User ID

Name	Format	Size	Offset	Description
ACCTNUM	CHAR	32	0	Trading partner account number, left-justified with trailing blanks.
USERID	CHAR	32	32	Trading partner user ID, left-justified with trailing blanks.
DIR	CHAR	1	64	Direction of interchange. Should have value of S.
INTCTLNO	CHAR	14	65	Interchange control number, right-justified with leading zeros.
RECEIVER	CHAR	35	79	Interchange receiver ID value as it was sent in the interchange with trailing blanks.

Transaction Handle

The status of an entire interchange can be updated by providing the transaction handle of any one of the transactions within the interchange (see Table 3-97). This is the update status key block that should be used when the ZFCBFUNC value is 211.

Table 3-97. Transaction Handle

Name	Format	Size	Offset	Description
THANDLE	CHAR	10	0	The transaction handle value of one of the transactions within the interchange. The transaction handle has a format of YYYYMMDDHHMMSSxxnnnn stored as a 10-byte packed value. All of the transactions within the interchange have the same status, and the handle value for any transaction within the interchange can be used on this request.

Alternate Key

One alternate key value consists of the combination of the **MSGCLASS**, **MSGNAME** and **SEQNUM** fields, and the other is a **UNIQUEID** value. The values for these fields are not known until an interchange is sent. Therefore, you must use the status update data block (see “Status Update Data Block” on page 3-131) along with one of the other keys to establish the alternate key values. Once this is done, subsequent updates to status based on network acknowledgments can be made with any of the key values. There are two ways to identify a trading partner when using an alternate key to update status:

- Interchange qualifier and interchange ID fields, which are used to search the trading partner profile members to find an entry with matching values
- Account number and user ID fields, which are used to search the trading partner profile members to find an entry with matching values

The formats for the two possible key values are shown in the Table 3-98 and Table 3-99.

Alternate Key 1 - Account Number and User ID

Table 3-98 shows the format for an alternate key using the account number and user ID. This is the update status key block that should be used when the ZFCBFUNC value is 212.

Table 3-98. Alternate Key Using Account Number and User ID

Name	Format	Size	Offset	Description
ACCTNUM	CHAR	32	0	Trading partner nickname account number, left-justified with trailing blanks.
USERID	CHAR	32	32	Trading partner user ID, left-justified with trailing blanks.
MSGCLASS	CHAR	8	64	Message user class assigned to the interchange, left-justified with trailing blanks.
MSGNAME	CHAR	8	72	Message name assigned to the interchange, left-justified with trailing blanks.
SEQNUM	CHAR	5	80	Sequence number assigned to the interchange, right-justified with leading zeros.
UNIQUEID	CHAR	8	85	The unique ID value assigned to the interchange by the network. This value is used rather than the MSGCLASS , MSGNAME , SEQNUM fields unless it contains blanks. If a UNIQUEID value is provided, ACCTNBR and USERID are not required.

Alternate Key 2 - Interchange Qualifier and ID

Table 3-99 shows the format of alternate key 2 using an interchange qualifier and ID. This is the update status key block that should be used when the ZFCBFUNC value is 214.

Table 3-99 (Page 1 of 2). Alternate Key Using Interchange Qualifier and ID

Name	Format	Size	Offset	Description
QUALIFIER	CHAR	4	0	Interchange qualifier as sent in the interchange with trailing blanks.
RECEIVER	CHAR	35	4	Interchange receiver ID as sent in the interchange with trailing blanks.
FILLER	CHAR	25	39	Filler field – initialize with blanks.

Table 3-99 (Page 2 of 2). Alternate Key Using Interchange Qualifier and ID

Name	Format	Size	Offset	Description
MSGCLASS	CHAR	8	64	Message user class assigned to the interchange, left-justified with trailing blanks.
MSGNAME	CHAR	8	72	Message name assigned to the interchange, left-justified with trailing blanks.
SEQNUM	CHAR	5	80	Sequence number assigned to the interchange, right-justified with leading zeros.
UNIQUEID	CHAR	8	85	The unique ID value assigned to the interchange by the network. This value is used rather than the MSGCLASS , MSGNAME , SEQNUM fields unless it contains blanks. If a UNIQUEID value is provided, QUALIFIER and RECEIVER are not required.

Status Update Data Block

The status update data block is optional on all the requests. This contains information about an interchange that is generally determined at the time an interchange is sent. The format and content of the data block are shown in Table 3-100.

Note: If a REQ ID provides in the call, a status update data block must also be available. If you do not have any data for the block, initialize all the fields to blank to indicate they do not apply.

In Table 3-100, the (**MSGCLASS**, **MSGNAME**, **SEQNUM**) fields are used to build one of the alternate keys that can be used later to retrieve and update status of this interchange.

Table 3-100. Interchange Fields Established During Status Update

Name	Format	Size	Offset	Description
MSGCLASS	CHAR	8	0	The message user class assigned to the interchange (see “API - Send Transactions and Restart Send Transactions” on page 3-109 for information on these values).
MSGNAME	CHAR	8	8	The message name assigned to the interchange (see “API - Send Transactions and Restart Send Transactions” on page 3-109 for information on these values).
SEQNUM	CHAR	5	16	The message sequence number assigned to the interchange (see “API - Send Transactions and Restart Send Transactions” on page 3-109 for information on these values).
UNIQUEID	CHAR	8	21	The unique message ID that was assigned to the interchange by the network program Note: This is an alternate key that can be used later to update status of this interchange. If this field is not blank, it must contain a value that is unique for all interchanges currently in the Transaction Store.
SENDDATE	CHAR	8	29	The date the interchange was sent (YYYYMMDD).
SENDTIME	CHAR	6	37	The time the interchange was sent (HHMMSS).
NETACK	CHAR	1	43	The type of network acknowledgment expected for this interchange. See “Trading Partner Profile Block (TPPDB)” on page A-42 for possible values.

SYNCPPOINT Services

The description of translation and enveloping services frequently refers to the recovery scope for the session (see “Send Recovery Scope” on page 3-42). SYNCPPOINT services provide the interface for controlling the recovery scope and provide an environment-independent interface for requesting either that changes to resources should be considered permanent (COMMIT work) or the changes should be removed (ROLLBACK work). Until a COMMIT work request is completed, any changes to resources are removed from the system in case of a system or application failure.

If a system or application fails, or if a specific ROLLBACK work request is made, all the resources changed since the last COMMIT point are backed out of the system. It will be as if the changes were never made.

Within an MVS/DB2 environment, a request to commit work is made directly to DB2 with a COMMIT request. A request to roll back work is made directly to DB2 with a ROLLBACK request. However, in the CICS/DB2 or CICS/VSAM environment, a request to commit work is made to CICS with an EXEC CICS SYNCPPOINT request. CICS passes this request on to DB2 and controls the committing of CICS resources with DB2 resources. In a CICS environment, a request to roll back work is made to CICS with an EXEC CICS SYNCPPOINT ROLLBACK request. Again, CICS passes this request to DB2 and controls the removal of CICS resources with DB2 resources.

SYNCPPOINT services takes care of these differences and issues the proper request based on the execution environment. The recovery scope is established using the API through the **SCOPE** field in the translator control block (TRCB). When you use the utilities, the RECOVERY keyword on the PERFORM command establishes the recovery scope. You can use SYNCPPOINT services to lengthen the recovery scope by setting the SYNCPPOINT interval to a positive integer value greater than 1. You can effectively turn off SYNCPPOINT by setting the SYNCPPOINT interval to -1. In this case, the application program is controlling the SYNC interval and must ensure that it does not shorten the recovery scope.

For example, if an interchange recovery scope is in effect and the application issues a COMMIT (or EXEC CICS SYNCPPOINT) request in the middle of the interchange, the chances of deadlock increase, and the consistency of the application databases and the DataInterchange Transaction Store is compromised.

Note: During API translation, setting the TRCB NOCOMMIT flag to Y prevents the translator from issuing commits. However, NOCOMMIT does not prevent DataInterchange termination from issuing a commit. A syncpoint service call with syncpoint interval set to -1 made prior to the DataInterchange termination call will prevent this terminate commit. For more information, see “Send Recovery Scope” on page 3-42.

DB/2 TIMEOUT / DEADLOCK Processing

The DB/2 TIMEOUT / DEADLOCK processing occurs within a DB2 environment when multiple translations are being run concurrently and the trading partners are not in the same order for all the translations. The DB/2 terminates one of these transactions and issues:

Return Code	Description
-911	The DB/2 parameter is ROLBE=YES.
-913	The DB/2 parameter is ROLBE=NO.

The translator attempts to recognize that a rollback occurred, issues a return code of -911, and allows the translation to continue, even though another translation may have acquired the next sequential control number. The DB/2 termination is flagged with an RS0000 message. The translator will attempt to retry

after a return code of -911 or -913 when appropriate. If DB/2 has issued a rollback, the translation will be terminated with a TR0810 message.

During retry, DataInterchange has the options to issue:

- A ROLLBACK command (IR)? and 2)
- OR
- A FETCH command for the Trading Partner Control numbers again (TA)

The following table describes the conditions and actions.

<i>Table 3-101. DataInterchange DB/2 Recovery Decision Table</i>						
Row	Conditions True or False				Actions Yes or No	
	Control	Deadlock	Updates	Rollback	DI Rollback	Fetch Control Numbers
1	F	F	F	F	N	Y
2	F	F	F	T	N	N
3	F	F	T	F	N	Y
4	F	F	T	T	N	N
5	F	T	F	F	N	N
6	F	T	F	T	N	N
7	F	T	T	F	N	N
8	F	T	T	T	N	N
9	T	F	F	F	N	Y
10	T	F	F	T	N	Y
11	T	F	T	F	N	Y
12	T	F	T	T	N	N
13	T	T	F	F	Y	Y
14	T	T	F	T	N	Y
15	T	T	T	F	Y	N
16	T	T	T	T	N	N

Notes:

1. Control specifies that DataInterchange is in control (for example, if SYNCVAL is not equal to -1).
2. Deadlock specifies that the problem was a deadlock (not a timeout).
3. Updates specifies that the updates were made before the error occurred.
4. Rollback specifies that DB2 issued a ROLLBACK statement when the error occurred.
5. DI Rollback specifies that DataInterchange will issue a ROLLBACK command.
6. Fetch Control Numbers specifies that DataInterchange will again attempt to FETCH the TP Control numbers.

API - Initialize SYNC

The syntax of the function request is:

FXXZccc(SNB,CCB,FCB,interval)

The parameters for this function request are:

Parameter	Description						
SNB	The service name block identifying SYNCPOINT services. <table><tr><th>Field</th><th>Value</th></tr><tr><td>ZSNBNAME</td><td>SYNCSERV to identify SYNCPOINT services</td></tr><tr><td>ZSNBPC</td><td>4 (number of parameters in the call)</td></tr></table>	Field	Value	ZSNBNAME	SYNCSERV to identify SYNCPOINT services	ZSNBPC	4 (number of parameters in the call)
Field	Value						
ZSNBNAME	SYNCSERV to identify SYNCPOINT services						
ZSNBPC	4 (number of parameters in the call)						
CCB	The same common control block used to initialize DataInterchange.						
FCB	The function control block with a ZFCBFUNC value of 1.						
interval	When the SYNCPOINT interval is desired, the interval is a four-byte binary value. Possible values are: <table><tr><td>0</td><td>General SYNCPOINT processing based on the recovery scope in effect.</td></tr><tr><td>N</td><td>A positive number indicating a SYNCPOINT should be taken every N intervals. For example, if N had a value of 5 and there is a transaction recovery scope, a COMMIT is issued only for every 5 transactions. (Actually, a COMMIT is requested after each transaction, but only every 5th one is honored by the SYNCPOINT service.)</td></tr><tr><td>-1</td><td>Indicates the application is controlling the SYNCPOINT, and all DataInterchange requests for COMMIT requests are ignored. The application controlling the SYNCPOINT does not mean that COMMIT requests can be issued indiscriminately. For example, if the application requests an interchange recovery scope and then issues a COMMIT after each transaction, the results are not as expected and the chance of deadlock increases dramatically. It is acceptable to lengthen the DataInterchange recovery scope value (as in the preceding example where a COMMIT is issued after every 5th transaction), but shortening the recovery scope leads to database integrity and deadlock problems.</td></tr></table>	0	General SYNCPOINT processing based on the recovery scope in effect.	N	A positive number indicating a SYNCPOINT should be taken every N intervals. For example, if N had a value of 5 and there is a transaction recovery scope, a COMMIT is issued only for every 5 transactions. (Actually, a COMMIT is requested after each transaction, but only every 5th one is honored by the SYNCPOINT service.)	-1	Indicates the application is controlling the SYNCPOINT, and all DataInterchange requests for COMMIT requests are ignored. The application controlling the SYNCPOINT does not mean that COMMIT requests can be issued indiscriminately. For example, if the application requests an interchange recovery scope and then issues a COMMIT after each transaction, the results are not as expected and the chance of deadlock increases dramatically. It is acceptable to lengthen the DataInterchange recovery scope value (as in the preceding example where a COMMIT is issued after every 5th transaction), but shortening the recovery scope leads to database integrity and deadlock problems.
0	General SYNCPOINT processing based on the recovery scope in effect.						
N	A positive number indicating a SYNCPOINT should be taken every N intervals. For example, if N had a value of 5 and there is a transaction recovery scope, a COMMIT is issued only for every 5 transactions. (Actually, a COMMIT is requested after each transaction, but only every 5th one is honored by the SYNCPOINT service.)						
-1	Indicates the application is controlling the SYNCPOINT, and all DataInterchange requests for COMMIT requests are ignored. The application controlling the SYNCPOINT does not mean that COMMIT requests can be issued indiscriminately. For example, if the application requests an interchange recovery scope and then issues a COMMIT after each transaction, the results are not as expected and the chance of deadlock increases dramatically. It is acceptable to lengthen the DataInterchange recovery scope value (as in the preceding example where a COMMIT is issued after every 5th transaction), but shortening the recovery scope leads to database integrity and deadlock problems.						

Initialize SYNC Return Codes

The results of the initialization request are posted in the return code and extended return code fields of the common control block. The following are possible values:

Return Code	Explanation				
0	Initialization was successful.				
12	Initialization failed. The extended return code is: <table><tr><th>ERC</th><th>Explanation</th></tr><tr><td>12</td><td>Insufficient virtual storage</td></tr></table>	ERC	Explanation	12	Insufficient virtual storage
ERC	Explanation				
12	Insufficient virtual storage				

COMMIT Work

The syntax of the function request is:

FXXZccc(SNB,CCB,FCB)

The parameters for this function request are:

Parameter	Description						
SNB	The service name block identifying SYNCPOINT services. <table><tr><th>Field</th><th>Value</th></tr><tr><td>ZSNBNAME</td><td>SYNCSERV to identify SYNCPOINT services</td></tr><tr><td>ZSNBPC</td><td>3 (number of parameters in the call)</td></tr></table>	Field	Value	ZSNBNAME	SYNCSERV to identify SYNCPOINT services	ZSNBPC	3 (number of parameters in the call)
Field	Value						
ZSNBNAME	SYNCSERV to identify SYNCPOINT services						
ZSNBPC	3 (number of parameters in the call)						
CCB	The same common control block used to initialize DataInterchange.						
FCB	The function control block with a ZFCBFUNC value of 2.						

The results of the COMMIT work request are posted in the return code and extended return code fields of the common control block. The following are possible values:

Return Codes	Explanation				
0	COMMIT work successful.				
4	COMMIT ignored. The extended return code is: <table><tr><th>ERC</th><th>Explanation</th></tr><tr><td>1</td><td>The request to COMMIT was ignored for one of the following reasons:<ul style="list-style-type: none">• An interval of -1 was established on the SYNCPOINT initialization function.• The interval has not been reached yet.</td></tr></table>	ERC	Explanation	1	The request to COMMIT was ignored for one of the following reasons: <ul style="list-style-type: none">• An interval of -1 was established on the SYNCPOINT initialization function.• The interval has not been reached yet.
ERC	Explanation				
1	The request to COMMIT was ignored for one of the following reasons: <ul style="list-style-type: none">• An interval of -1 was established on the SYNCPOINT initialization function.• The interval has not been reached yet.				
12	COMMIT failed. The extended return code is: <table><tr><th>ERC</th><th>Explanation</th></tr><tr><td>N</td><td>The SQLCODE from DB2 indicating why the COMMIT was not honored.</td></tr></table>	ERC	Explanation	N	The SQLCODE from DB2 indicating why the COMMIT was not honored.
ERC	Explanation				
N	The SQLCODE from DB2 indicating why the COMMIT was not honored.				

ROLLBACK Work

The syntax of the function request is:

FXXZccc(SNB,CCB,FCB)

The parameters for this function request are:

Parameter	Description						
SNB	The service name block identifying SYNCPOINT services. <table><tr><th>Field</th><th>Value</th></tr><tr><td>ZSNBNAME</td><td>SYNCSERV to identify SYNCPOINT services</td></tr><tr><td>ZSNBPC</td><td>3 (3 parameters in the call)</td></tr></table>	Field	Value	ZSNBNAME	SYNCSERV to identify SYNCPOINT services	ZSNBPC	3 (3 parameters in the call)
Field	Value						
ZSNBNAME	SYNCSERV to identify SYNCPOINT services						
ZSNBPC	3 (3 parameters in the call)						
CCB	The same CCB used to initialize DataInterchange.						
FCB	The function control block with a ZFCBFUNC value of 3.						

The results of the ROLLBACK work request are posted in the return code and extended return code fields of the common control block. The following are possible values:

Return Codes	Explanation
--------------	-------------

0	ROLLBACK work successful.
12	ROLLBACK work failed. The extended return code is:

ERC	Explanation
-----	-------------

N	The SQLCODE from DB2 indicating the reason for ROLLBACK failure.
---	--

Note: A ROLLBACK request is always honored and issued even when the SYNCPOINT interval has not been reached or has a value of -1.

API - Get Envelope Service

The Get Envelope Service can be used to get data directly from the translator, rather than from the file the translator writes to. It is used during outbound processing. Normally, after DataInterchange creates an interchange, the interchange is routed automatically to a file (in CICS, this file is always a temporary storage queue). A user written exit may be used to circumvent this action. In other words, using the Get Envelope Service, a user written exit can retrieve the envelope directly from the translator and then do whatever it wants with it. (In CICS, for example, the interchange could be routed to a transient data queue). DataInterchange recognizes this type of user exit when the following keywords are used on PERFORM ENVELOPE type commands: IEXIT(exitname), IACCESS(M), and ITYPE(UE). For more information, see "Get Envelope Service Example" on page D-117 and "Get/Put Envelope Exit and Service" on page 4-13.

The syntax of this API request is:

```
FXXZASM(GPCB CCB FCB BUFFER LEN)
```

The parameters for this function request are:

Parameter	Description
GPCB	The Get/Put envelope control block. It is the fourth parameter passed to IEXIT user exits.
CCB	The common control block used to initialize DataInterchange. It is the second parameter passed to IEXIT user exits.
FCB	The function control block. It is the third parameter passed to IEXIT user exits.
BUFFER	The working storage into which the envelope will be read. The length of this area must be specified in the LEN parameter.
LEN	The full-word length of the working storage that will contain the envelope. It pertains to the length of BUFFER.

API - Put Envelope Service

The Put Envelope Service can be used to put data directly into the translator, rather than having the translator read a file. It is used during inbound processing. Normally, in order to deenvelope an interchange, DataInterchange reads a file containing the interchange. (In CICS, this file is always a temporary storage queue.) A user-written exit may be used to circumvent this action. In other words, using the Put Envelope Service, a user-written exit can pass an envelope directly into the translator. (In CICS, for example, an interchange could be read from a transient data queue and passed directly into the translator.) DataInterchange recognizes this type of user exit when the following keywords are used on PERFORM DEENVELOPE type commands: IEXIT(exitname), IACCESS(M), and ITYPE(UE). For more information, see "Put Envelope Service Example" on page D-118 and "Get/Put Envelope Exit and Service" on page 4-13.

The syntax of this API request is:

FXXZASM(GPCB CCB FCB BUFFER LEN)

The parameters for this function request are:

Parameter	Description
GPCB	The Get/Put envelope control block. It is the fourth parameter passed to IEXIT user exits.
CCB	The common control block used to initialize DataInterchange. It is the second parameter passed to IEXIT user exits.
FCB	The function control block. It is the third parameter passed to IEXIT user exits.
BUFFER	The working storage into which the envelope will be read. The length of this area must be specified in the LEN parameter.
LEN	The full-word length of the working storage that will contain the envelope. It pertains to the length of BUFFER.

Chapter 4. Exit Routines

Return Codes	4-1
DataInterchange User Extensions	4-1
Exit Languages	4-2
Exit Linkage Editor Instructions	4-3
Field Exit Routines	4-3
Field Exit Routines Shipped with DataInterchange	4-4
Send Parameters	4-4
Receive Parameters	4-6
Field Exit Parameter Language Definition	4-7
Transaction Exit Routines	4-9
Pre-Translation Exit	4-9
Post-Translation Exit	4-9
Pre/Post Parameters	4-10
Get/Put Envelope Exit and Service	4-13
Get Envelope	4-13
Put Envelope	4-14
Identifying Get Envelope Service Return Codes	4-15
Identifying Put Envelope Service Return Codes	4-15
Security Routines	4-15
Enable Security During Send	4-17
Enable Security During Receive	4-17
Security Parameters Introduction	4-18
Encryption Routine	4-18
Authentication Routine	4-22
Compression Routine	4-26
Filtering Routine	4-26
Security Support Routines	4-30
Independent Programs	4-32
Data Extract Exit	4-32
Get/Put Envelope Program	4-33

Chapter 4. Exit Routines

Return Codes

The tables in this chapter refer to the DataInterchange Utility Severity (UTILSEV) and Condition (UTILCCODE) codes. For MVS batch users, the UTILCCODE code is returned as the Job Step Condition Code. For CICS users, these codes are returned in the Utility Control Block. See “Format of DataInterchange Utility Control Information” in Chapter 5, “Using DataInterchange in the CICS Environment.” UTILSEV refers to the Utility Control Block field, CCBRC. And UTILCCODE refers to the Utility Control Block field, CCBERC. The Utility Control Block fields CCBRC and CCBERC should not be confused with the Common Control Block (CCB) fields ZCCBRC and ZCCBERC.

An exit routine is a program that you provide to perform some service for your application or data. DataInterchange calls the exit routine at an appropriate time, and passes to the exit routine information needed to accomplish the task. When it finishes, the exit routine returns the results to DataInterchange. For example, DataInterchange may pass encrypted data to the exit routine and receive back the data in decrypted form. The information passed to each exit routine (its parameters) is defined in detail as each possible type of exit routine is described.

There are two types of exit routines. One type of exit routine is a user extension to DataInterchange and can interact directly in the current DataInterchange session. The other type is an independent program and has no knowledge of the current session. The next section discusses the details of the extension exits and is followed by a description of the independent program exits.

DataInterchange User Extensions

There are five instances where application programs can be defined to extend or enhance the capabilities of DataInterchange. The five instances are:

- Field exit routines

These routines provide additional processing for application data (translate to standard) and standard data (translate to application) during translation.

- Pre-translation and post-translation exit routines

These routines provide additional processing for an entire transaction after translate to standard or before translate to application.

- Security exit routines

These routines protect transaction data through encryption and authentication. Filtering and compression exits are also defined as part of this process.

- Point-to-point network program exit

This routine gets invoked on communication requests for a point-to-point network.

- Message handling exits

These routines are invoked to process responses from the networks that are not directly supported by DataInterchange.

The first three instances are described in the following sections. Point-to-point network programs and message handlers are discussed in Chapter 6, “Interfacing to Other Networks and Applications.”

The architecture for user extensions to DataInterchange is very similar to the architecture used when calling services using the DataInterchange application program interface (API). Services are given logical names (for example, TRANPROC for translation services) and the association between a logical name and a physical load module is accomplished at execution time. User extensions use this same architecture in that exits are identified with a logical name. A logical name for an exit is specified at various points in the customization process.

1. The logical name for a field exit routine is specified in the User exit routine name field on the special handling panels (TP30 for send, TP31 for receive) during the mapping process.
2. The logical name for a pre-translation routine is specified in the **Pre-translation exit routine** field on the Add Trading Partner Usage for Receiving panel (TP22). The logical name for a post-translation routine is specified in the **Post-translation exit routine** field on the Add Trading Partner Usage for Sending panel (TP17).
3. The logical names for security exit routines are specified in the security profile (SECUPROF).
 - The logical name for a compression routine is in the **Comp. program** field.
 - The logical name for a filtering routine is in the **Filtering program** field.
 - The logical name for an encryption/decryption routine is in the **Encr. program** field.
 - The logical name for an authentication routine is in the **Auth. program** field.

Before using the logical name of a service in one of the fields mentioned above, it is necessary to **DEFINE** the exit routine to DataInterchange. The definition of an exit routine consists of associating a logical name for an exit with a physical load module name and the programming language used to write the exit routine. This information is defined in the user program information profile (ADAMCTL). The acronym ADAMCTL stands for Application Data Access Method Control. A complete description of this profile can be found in the *DataInterchange Administrator's Guide*.

Exit Languages

A user exit may be written in Assembler, C, or COBOL. DataInterchange needs to know the implementation language for a user exit and this information is provided in the user program information profile (ADAMCTL). The **Program language** field of the profile entry has a 1-character code that specifies the programming language. The possible values are:

Value	Description
A	Assembler programs
C	C language programs. DataInterchange supports only the <i>System Application Architecture (SAA) C/370</i> compiler.
J	COBOL programs written using a COBOL compiler other than <i>IBM COBOL II</i> . DataInterchange only supports non-IBM COBOL II programs for field exit routines, pre-translation exit routines, and post-translation exit routines.
K	COBOL II programs. DataInterchange fully supports the <i>IBM COBOL II</i> compiler.

Note: References to COBOL in this book refer to COBOL II programs and non-COBOL II programs. However, non-IBM COBOL II support is limited to field exit routines, pre-translation exit routines, and post-translation exit routines, unless specified otherwise.

Exit Linkage Editor Instructions

When DataInterchange determines that a user exit should be called, the logical name of the user exit is used to read the ADAMCTL profile entry to determine the physical load module name (Load module name) and the implementation language (Program language). DataInterchange issues an operating system load for the load module and then passes control to the exit routine. Information about the exit is retained by DataInterchange so if the exit routine is needed more than once, the subsequent requests are satisfied by passing control to the exit routine. Because programs are only loaded once, all exits have to be REUSEABLE at a minimum and they should be REENTRANT (COBOL programs should use the RENT compile time option).

For assembler, C, and COBOL II exit programs, the parameters that are provided to an exit routine may either be above or below the 16M line. Therefore, these exit programs must be linked with 31-bit addressing (AMODE 31). These exit routines may reside either above or below the 16M line.

The **Load module name** field in the ADAMCTL profile must match the name given to the load module in the linkage editor control statements. This is the name that DataInterchange attempts to load and if a load module by this name does not exist in any load library (STEPLIB/JOBLIB/LINKLIB), the result is a system 806 ABEND.

Note: In CICS, a PPT entry must exist for the program or a CICS load failure results.

The entry point for a program written in Assembler or COBOL should be the same as the physical load module name. Programs written in C have the following requirements:

1. The function name for the C routine must be **main**.
2. The linkage editor control statements should have an INCLUDE for the FXXZCITF load module. FXXZCITF is in the DataInterchange load module data set (EDI.V2R1M0.SEDILMD1).
3. FXXZCITF should be made the ENTRY point for the load module. The FXXZCITF program establishes the C environment and then transfers control to the **main** function within the program.

Field Exit Routines

Field exit routines provide additional processing for application data (translate to standard) and standard data (translate to application) during translation.

The logical name for a field exit routine is specified in the **User exit routine name** field on the special handling panels (TP30 for send, TP31 for receive) during the mapping process. The physical characteristics of the exit are defined in the ADAMCTL profile.

During translate-to-standard operations, the translator calls a field exit routine before taking any action with the application data field.

During translate-to-application operations, the translator calls a field exit routine before taking any action with a standard data element.

In either case, the exit routine can inspect the data and do any of the following:

- Verify the data against a predefined set of rules. The exit can tell DataInterchange to ignore the data or to ignore the mapping by return code settings. For more information, see “Send Parameters” on page 4-4 and “Receive Parameters” on page 4-6.

- Change the value of the data and tell DataInterchange to use the new value.
- Save the value of the data for use by other field exit routines.

A sample field exit routine is provided in “Field Exit Program” on page D-57.

Field Exit Routines Shipped with DataInterchange

These exits allow you to use or not use the value of an application field based on the data within another application field. They must always be paired. An EDICHKI or EDICHKU must always be followed by an EDIQQF. These field exit routines are designed to work with send translation only.

The following field exit routines are shipped with the product:

- EDICHKI checks the value of a field and ignores the application data. This exit sets a flag indicating whether the field contained all blanks.
- EDICHKU checks the value of a field and still uses the application data in the field. This exit sets a flag indicating whether the field contained all blanks.
- EDIQQF uses the flag set by either EDICHKI or EDICHKU to determine whether the field contained non-blank data. If the field contained blanks, this exit returns to the translator and tells it to ignore the data. If a literal is associated with this field, the literal will also be ignored if the flag indicated an all-blank field.

For example, if you have application field A mapped to standard field X, and application field B mapped to standard field Y, but you only want standard field X to appear if application field B contains data, you could:

1. Map application field B to standard field Y and field exit EDICHKI.
2. Map application field A to standard field X and field exit EDIQQF.

If you have the above case, but you want both standard fields X and Y created only if application field B contains data, you could:

1. Map application field B to standard field Y and field exit EDICHKU.
2. Map application field A to standard field X and field exit EDIQQF.

Send Parameters

The list below describes the parameters that are passed to a field exit routine during translate-to-standard operations. At this point, the exit routine can inspect the application data and determine how that data should be processed in creating standard data element values.

See “Field Exit Parameter Language Definition” on page 4-7 for examples on how the parameters should be declared within Assembler, COBOL, and C programs.

The parameter list for field exit routines during translate to standard operations consists of the following pointers:

- Service name block (SNB)

See “Service Name Block (SNB)” on page A-1 for a detailed description of this block. The **ZSNBNAME** field contains the logical name for the exit (value of **User exit routine name** from the Send Special Handling panel (TP30)). It is possible to combine many field exit routines into a single physical load module and if this is done, the value of **ZSNBNAME** can be used to determine the reason the exit is being called.

- Common control block (CCB)

See “Common Control Block (CCB)” on page A-2 for a detailed description of this block. The **ZCCBRC** field is used to tell DataInterchange what further actions should be taken against the current application data. **ZCCBRC** valid values are:

Value	Meaning
0	Continue normal processing for the application field. A new value for the field may have been provided in the temporary work area if the return field length is not zero.
1	The application field should be ignored but any default literal processing still applies.
2	The application field and default literal should be ignored.
3 - 20	Reserved.
21 and higher	An error detected by the user exit. DataInterchange creates a log record (message TR0006) indicating a user exit error occurred, which is treated as a level 1 (data element) error and does not process the field any further (same as a return code of 1).

- Field value

This is the actual application field from the data format structure. You can change the field's value directly in this buffer if you do not increase the length. If the value is changed directly in the buffer, however, and the field is mapped more than once, subsequent mappings and exits might see the changed value rather than the original value. If this is not desired, use the temporary work area and the return field length to provide the changed value to DataInterchange.

- Field offset (4-byte binary value)

You can use this value to determine the field that you are processing. However, DataInterchange does not pass the entire structure to the field exit routine. It passes only the field specified during mapping. You cannot use this field to access the entire structure.

- Fields length (4-byte binary value)

Your exit routine can change the value in this field to a smaller value if the field should be shortened. If you need to increase the size of this field, you must use the temporary work area and the new length field.

- Permanent work area (4096-byte buffer)

Your exit routine can use this work area for its processing. DataInterchange initializes the work area to binary zeros at the start of translation. After initialization, your exit routine determines the content and format of the work area. The same work area is passed to all exit routines during the translation session.

- Temporary work area (1024-byte buffer)

Your field exit routine can use this work area to store a modified version of the input data. DataInterchange initializes this work area with blanks before calling the exit routine.

- Return field length (4-byte binary value)

This field has a value of zero on entry to the exit routine. A nonzero value in this field, when the exit routine returns to DataInterchange, indicates that the data in the temporary work area should be used.

Receive Parameters

The list below describes the parameters that are passed to a field exit routine during translate-to-application operations. At this point, the exit routine can inspect standard data elements and determine how that data should be processed in creating application field values.

See “Field Exit Parameter Language Definition” on page 4-7 for examples on how the parameters should be declared within Assembler, COBOL, and C programs.

The parameter list for field exit routines during translate to application operations consists of the following pointers:

- Service name block (SNB)

See “Service Name Block (SNB)” on page A-1 for a detailed description of this block. The **ZSNBNAME** field contains the logical name for the exit (value of the **User exit routine name** field from the Receive Special Handling panel (TP31). It is possible to combine many field exit routines into a single physical load module and if this is done, the value of **ZSNBNAME** can be used to determine the reason the exit is being called.

- Common control block (CCB)

See “Common Control Block (CCB)” on page A-2 for a detailed description of this block. The **ZCCBRC** field is used to tell DataInterchange what further actions should be taken against the current application data. **ZCCBRC** valid values are:

Value	Meaning
0	Continue normal processing for the standard data element. A new value for the data element may have been provided in the temporary work area if the return field length is not zero.
1	Ignore the data element, but any default literal processing still applies.
2	Ignore the data element and default literal.
3 - 20	Reserved
21 and higher	An error detected by the user exit. DataInterchange creates a log record (message TR0006) indicating a user exit error occurred, which is treated as a level 1 (data element) error and does not process the data element any further (same as a return code of 1). To set the functional acknowledgment code, see the temporary work area.

- Data element value

This is the actual data element value from the segment. You can change the value directly in this buffer if you do not increase the length. If the value is changed directly in the buffer, however, and the data element is mapped more than once, subsequent mappings and exits may use the changed value rather than the original value. If this is not desired, use the temporary work area and the return field length to provide the changed value to DataInterchange.

- Field offset (4-byte binary value)

You can use this value to determine the data element that you are processing. However, DataInterchange does not pass the entire segment to the field exit routine. It passes only the data element specified during mapping. You cannot use this field to access the entire segment.

- Fields length (4-byte binary value)

Your exit routine can change the value in this field to a smaller value if the data element should be shortened. If you need to increase the size of this field, you must use the temporary work area and the new length field.

- Permanent work area (4096-byte buffer)

Your exit routine can use this work area for its processing. DataInterchange initializes the work area to binary zeros at the start of translation. After initialization, your exit routine determines the content and format of the work area. The same work area is passed to all exit routines during the translation session.

- Temporary work area (1024-byte buffer)

Your field exit routine can use this work area to store a modified version of the input data. DataInterchange initializes this work area with blanks before calling the exit routine. If the user exit found an error in the data and rejects the data by setting the CCB return code to a value greater than 20, you can use the first byte of this work area to set the functional acknowledgment error code that should be used in the AK4 segment. The return value must be numeric, or it will be ignored. Other than that, the return value is neither edited or validated. If 999 or CONTRL functional acknowledgments are being created, DataInterchange will transform the 997 AK4 value to the appropriate 999 or CONTRL value.

- Return field length (4-byte binary value)

This field has a value of zero on entry to the exit routine. A nonzero value in this field, when the exit routine returns to DataInterchange, identifies that the data in the temporary work area should be used.

Field Exit Parameter Language Definition

The following sections show how the parameters provided to field exit routines should be defined in COBOL, C, and Assembler.

COBOL definition: The COBOL definitions for parameters are shown below. The SNB and CCB are omitted here, but are provided in “COBOL SNB” on page C-3 and “COBOL CCB” on page C-2.

LINKAGE SECTION.

```
*****
*      ADF - APPLICATION DATA FORMAT  FIELD DATA      *
*****
```

```
01  FLD-DATA              PIC X(100).
```

```
*****
*      OFFSET OF ADF FIELD DATA WITHIN STRUCTURE      *
*****
```

```
01  FLD-OFFSET            PIC 9(09) COMP.
```

```
*****
*      LENGTH OF THE FIELD BEING PASSED TO THE EXIT ROUTINE  *
*****
```

```
01  FLD-LENGTH            PIC 9(09) COMP.
```

```
*****
*      THE 4096 BYTE PERMANENT WORK AREA                *
*****
```

```
01 PERM-AREA          PIC X(4096).
```

```
*****
*          THE 1024 BYTE TEMPORARY WORK AREA          *
*****
```

```
01 TEMP-AREA          PIC X(1024).
```

```
*****
*          LENGTH OF DATA IN THE TEMPORARY WORK AREA    *
*****
```

```
01 TEMP-LENGTH        PIC 9(09) COMP.
```

```
PROCEDURE DIVISION USING SNB-DATA
                        CCB-DATA
                        FLD-DATA
                        FLD-OFFSET
                        FLD-LENGTH
                        PERM-AREA
                        TEMP-AREA
                        TEMP-LENGTH.
```

C Definition: The C definitions for parameters are shown below. The SNB and CCB are omitted here but are provided in “C SNB” on page C-25 and “C CCB” on page C-25.

```
typedef struct WORKAREA
worka;
struct WORKAREA {
    char    working[4096];
};
```

```
main(snbdata,
     ccbdata,
     flddata,
     fldoffset,
     fldlength,
     permarea,
     temparea,
     templength)
snb    *snbdata;    /* Service name block pointer set up by DI */
ccb    *ccbdata;    /* Common block pointer used by DI */
char   *flddata;    /* Pointer to the data */
long   *fldoffset;  /* Address of offset of field */
long   *fldlength;  /* Length of the data */
worka  *permarea;   /* Address of a work area */
char   *temparea;   /* Area when I can move result data */
long   *templength; /* Length of data moved to result area */
```

Assembler Definition: The Assembler definitions for parameters are shown below. The SNB and CCB are omitted here but are provided in “ASSEMBLER SNB” on page C-37 and “ASSEMBLER CCB” on page C-36.

```

*
* DSECT describing parameters to this routine
* The address of this parameter list is contained in Register 1
* at entry to the program.
*
PARMS    DSECT
SNBDATA  DS      A           Address of the SNB
CCBDATA  DS      A           Address of the CCB
FLDDATA  DS      A           Address of data
FLDOFF   DS      A           Address of 4 bytes containing offset
FLDLLEN  DS      A           Address of 4 bytes containing length
PERMAREA DS      A           Address of 4096 work area
TEMPAREA DS      A           Address of 1024 temporary area
TEMPLEN  DS      A           Address of 4 bytes to return length value
*
USREXIT  CSECT
        USING PARMS,R1

```

Transaction Exit Routines

Pre-translation and post-translation exit routines provide additional processing for an entire transaction after translate-to-standard or before translate to application.

The logical name for a pre-translation routine is specified in the **Pre-translation exit routine** field on the Add Trading Partner Usage for Receiving panel (TP22). The logical name for a post-translation routine is specified in the **Post-translation exit routine** field on the Add Trading Partner Usage for Sending panel (TP17). The physical characteristics of the exit are defined in the ADAMCTL profile.

Pre-Translation Exit

During translate-to-application operations, the translator calls a pre-translation routine before any translation takes place. The exit has access to the entire transaction (all data between but not including the transaction set header and transaction set trailer). The exit routine can perform any operation on this data. Modifications become part of the transaction image. In modifying the data, observe the following restrictions:

- Do not change the length of the data.
- Do not change a character to make it look like a segment delimiter.

Post-Translation Exit

During translate-to-standard operations, the translator calls a post-translation routine after the translation is complete and before the transaction image is written to the Transaction Store. The exit program has access to the entire transaction (all the data between but not including the transaction set header and transaction set trailer). The post-translation exit routine can perform various modifications on this transaction. In modifying the data, observe the following restrictions:

- Do not change the length of the data.
- Do not change any segment delimiters.
- Do not change any non-segment delimiter character to be the segment delimiter character.

This restriction ensures that the receiving translator can verify that it is receiving the correct number of segments, specified by the transaction set trailer.

- Do not change the data to appear as though it is a transaction set trailer, group trailer, or interchange trailer.

Pre/Post Parameters

The parameter list for pre-translation and post-translation exit routines consists of the following pointers:

- Service name block (SNB)

See “Service Name Block (SNB)” on page A-1 for a detailed description of this block. The **ZSNBNAME** field contains the logical name for the exit (value of the **Post-translation exit routine** field from the Add Trading Partner Usage for Sending panel (TP17), or the value of the **Pre-translation exit routine** field from the Add Trading Partner Usage for Receiving panel (TP22). It is possible to combine many field exit routines into a single physical load module and if this is done, the value of **ZSNBNAME** can be used to determine the reason the exit is being called.

- Common control block (CCB)

See “Common Control Block (CCB)” on page A-2 for a detailed description of this block. The **ZCCBRC** field is used to tell DataInterchange what further actions should be taken against the current application data. **ZCCBRC** valid values are:

Value	Meaning
0	The modified data returned by the exit routine becomes the transaction image.
1 - 20	Reserved.
21 and higher	The original data is used as the transaction image. DataInterchange creates a log record (message TR0006) indicating a user exit error occurred, which is treated as a level 1 (data element) error.

- Transaction image

This is the complete transaction image between the transaction set header and transaction set trailer (not including the header or trailer).

- Compatibility parameter (4-byte binary value containing 0)

This parameter is used only to maintain compatibility between the user exit parameter list and the pre-translation and post-translation parameter list.

- Image length (4 byte-binary value)

This is the length of the transaction image. The exit routine must not change this value. Unpredictable results occur if this value changes.

- Permanent work area (4096-byte buffer)

Your exit routine can use this work area for its processing. DataInterchange initializes the work area to binary zeros at the start of translation. After initialization, your exit routine determines the content and format of the work area. The same work area is passed to all exit routines during the translation session.

- Temporary work area (1024-byte buffer)

Your exit routine can use this work area as necessary. DataInterchange initializes this work area with blanks before calling the exit routine.

- Compatibility field (4-byte binary value containing 0)

This parameter is used only to maintain compatibility between the user exit parameter list and the pre-translation and post-translation parameter list.

The following sections show how the parameters provided to the pre-translation and post-translation exit routines should be defined in COBOL, C and Assembler.

COBOL Definition: The COBOL definitions for parameters are shown below. The SNB and CCB are omitted here, but are provided in “COBOL SNB” on page C-3 and “COBOL CCB” on page C-2.

LINKAGE SECTION.

```
*****
*      TRANSACTION IMAGE BETWEEN HEADER AND TRAILER      *
*****

01  TRX-DATA                PIC X(32768).

*****
*      COMPATIBILITY FIELD                                *
*****

01  FLD-COMPAT              PIC 9(09) COMP.

*****
*      LENGTH OF THE TRANSACTION IMAGE                    *
*****

01  TRX-LENGTH              PIC 9(09) COMP.

*****
*      THE 4096 BYTE PERMANENT WORK AREA                  *
*****
```

```

01  PERM-AREA          PIC X(4096).

*****
*          THE 1024 BYTE TEMPORARY WORK AREA          *
*****

01  TEMP-AREA          PIC X(1024).

*****
*          COMPATIBILITY FIELD                        *
*****

01  FLD-COMPAT1        PIC 9(09) COMP.

PROCEDURE DIVISION USING SNB-DATA
                        CCB-DATA
                        TRX-DATA
                        FLD-COMPAT
                        TRX-LENGTH
                        PERM-AREA
                        TEMP-AREA
                        FLD-COMPAT1.

```

C Definition: The C definitions for parameters are shown below. The SNB and CCB are omitted here, but are provided in “C SNB” on page C-25 and “C CCB” on page C-25.

```

typedef struct WORKAREA worka;
struct WORKAREA {
    char    working-4096-;
};

main(snbdata,
     ccbdata,
     trxdata,
     fldcompat,
     trxlenght,
     permarea,
     temparea,
     fldcompat1)

snb    *snbdata; /* Service name block pointer set up by DI */
ccb    *ccbdata; /* Common block pointer used by DI          */
char   *trxdata; /* Pointer to the transaction image           */
long   *fldcompat; /* Compatibility field                                     */
long   *trxlenght; /* Length of the transaction image                         */
worka  *permarea; /* Address of a work area                                  */
char   *temparea; /* Area when I can move result data                        */
long   *fldcompat1; /* Compatibility field                                    */

```

Assembler Definition: The Assembler definitions for parameters are shown below. The SNB and CCB are omitted here but are provided in “ASSEMBLER SNB” on page C-37 and “ASSEMBLER CCB” on page C-36.

```

*
* DSECT describing parameters to this routine
* The address of this parameter list is contained in Register 1
* at entry to the program.
*
PARMS    DSECT
SNBDATA  DS      A           Address of the SNB
CCBDATA  DS      A           Address of the CCB
TRXDATA  DS      A           Address of transaction image
COMPAT   DS      A           Compatibility field
TRXLEN   DS      A           Address of 4 bytes containing length of trx.
PERMAREA DS      A           Address of 4096 work area
TEMPAREA DS      A           Address of 1024 temporary area
COMPAT1  DS      A           Compatibility field
*
USREXIT  CSECT
        USING PARMS,R1

```

Get/Put Envelope Exit and Service

DataInterchange provides the capability of retrieving an envelope through storage after an enveloping operation (bypasses the write of the output file), and providing an envelope through storage before the deenveloping operation (bypasses the read of an input file). For more information, see “API - Get Envelope Service” on page 3-136 and “API - Put Envelope Service” on page 3-136.

The following section describes the Get/Put Envelope exit processing and service when the exit program is defined as a USER EXIT rather than a program. An exit program is specified in the utility control statements (IEXIT, ITYPE, IAREA and IACCESS) or in the translator control block (TRCB) fields (IUSEREXIT, IUSERTYPE, IUSERAREA, IUSERACCESS). A user exit type is defined with an ITYPE value of UE. See “Get/Put Envelope Program” on page 4-33 for a description of the parameters when the exit is an independent program rather than a user exit.

Get Envelope

The retrieval of envelope data (Get) is initiated by an API envelope call to DataInterchange from a user-written API program. DataInterchange envelopes the data into internal storage and then checks for a user exit in the Translator Control Block (TRCB) field IUSEREXIT. If an exit is found, instead of writing the envelope to a file, DataInterchange calls the user exit. A pointer field (IUSERAREA) is provided in the TRCB to allow the user's API program to pass a user-defined area to the user exit. This user-defined area may be used by the API program to provide parameters for the exit program.

The user exit then calls the Get Service to retrieve the envelope from the DataInterchange internal storage into a user-specified buffer and transfers it to its ultimate destination. Repeated calls must be made to retrieve subsequent pieces of the envelope until a return code of 8 with extended return code of 3 is detected (indicating no more data). No data is returned with this return code.

On successful return from the user exit (CCB return code is zero), DataInterchange continues processing. On any unsuccessful return (CCB return code not zero), a message (TR1255 or TR1256) is logged with the return code, processing terminates, and the return code is returned to the API program.

Put Envelope

The providing of envelope data (Put) is initiated by an API deenvelope call to DataInterchange from a user-written API program. DataInterchange checks for a user exit in the Translator Control Block (TRCB) field IUSEREXIT. If an exit is found, instead of reading the envelope from a file, DataInterchange calls the user exit. A pointer field (IUSERAREA) is provided in the TRCB to allow the user's API program to pass a user-defined area to the user exit. This user-defined area may be used by the API program to provide parameters for the exit program.

The user exit then retrieves the envelope from its origin and calls the Put Service to store the envelope data into the DataInterchange internal storage from a user-specified buffer. If the API program does not want to pass all of the envelope data in a single buffer, multiple calls can be made to the Put Service to store subsequent pieces of the envelope.

On successful return from the user exit (CCB return code is zero), DataInterchange deenvelopes the data that was stored by the user exit. On any unsuccessful return (CCB return code not zero), a message (TR1255 or TR1256) is logged with the return code, processing terminates, and the return code is returned to the API program.

When DataInterchange calls a Get Envelope or Put Envelope exit, it passes two parameters in addition to the normal SNB, CCB, and FCB parameters. One is a Get/Put control block that is used as the first parameter to call the Get/Put service. The other is the user-defined area pointed to by TRCB field IUSERAREA.

The interface to the Get or Put Envelope exit is the DataInterchange language-dependent stub (FXXZccc). The format of the stub follows:

```
FXXZccc(snb,ccb,fcg,pcb,userarea)
```

The following are the parameters for the FXXZccc stub:

Parameter	Description
snb	Indicates the service name block that DataInterchange passes as input to the exit routine.
ccb	Indicates the common control block that DataInterchange passes as input to the exit routine.
fcg	Indicates the function control block that DataInterchange passes as input to the exit routine. This block is initialized by DataInterchange with a function code of 1 on an enveloping operation (to allow its use in calling the Get Service), and a function code of 2 on a deenveloping operation (to allow its use in calling the Put Service).
pcb	Indicates the Get/Put control block that DataInterchange passes as input to the exit. This block was initialized by DataInterchange with values necessary to call the Get or Put Service. It must be passed as the first parameter to the service.
userarea	Indicates the address of the user-defined area. DataInterchange uses the value from IUSERAREA in the TRCB for this address.

The interface to the Get or Put Service (called by the exit) is the DataInterchange language-dependent stub (FXXZccc) and is dependent on the language in which the exit was written. The format of the stub follows:

```
FXXZccc(pcb,ccb,fcg,buffer,length)
```


The following are the parameters for the FXXZccc stub:

Parameter	Description
gpcb	Indicates the Get/Put control block that DataInterchange passed to the exit. This block was initialized by DataInterchange with values necessary to call the Get or Put Service.
ccb	Indicates the common control block that DataInterchange passed to the exit routine.
fcbl	Indicates the function control block that DataInterchange passed to the exit routine. This block was initialized by DataInterchange with a function code of 1 on an enveloping operation (to allow its use in calling the Get Service), and a function code of 2 on a deenveloping operation (to allow its use in calling the Put Service).
buffer	Indicates the address where DataInterchange should place the envelope on a Get function or the address from which DataInterchange should move the envelope on a Put function.
length	Indicates the address of a 4-byte field, which contains the size of the buffer on a Get function call and the actual number of bytes retrieved is returned here, or it contains the actual number of bytes in the buffer on a Put function.

Identifying Get Envelope Service Return Codes

The Get Envelope Service indicates in the CCB whether the Get process was successful. The CCB contains the following return codes (RC) and extended return codes (ERC).

These return codes Indicate

RC = 8, ERC = 1	The function code in the FCB is invalid
RC = 8, ERC = 3	End of data (no data is returned with this code)
RC = 8, ERC = 4	Buffer too small for minimum data (no data is returned but the minimum size required for the call is returned in the length field)
RC = 0, ERC = 0	Envelope data was returned

Identifying Put Envelope Service Return Codes

The Put Envelope Service indicates in the CCB whether the Put process was successful. The CCB contains the following return codes (RC) and extended return codes (ERC).

These return codes Indicate

RC = 8, ERC = 1	The function code in the FCB is invalid
RC = 12, ERC = 2	A virtual storage failure occurred during the Put function
RC = 0, ERC = 0	Envelope data was stored

Security Routines

Security exit routines protect transaction data through encryption and authentication. Encryption protects data against unauthorized viewing. Authentication protects data against unauthorized changes.

Filtering and compression are also defined as part of the encryption and authentication architecture. Filtering ensures that data does not contain characters that could conflict with the control characters used in transmitting an interchange. Filtering is only necessary if the network used to send the data has restrictions on the characters that can be transmitted. Compression can be used to decrease the amount of data that is being transmitted, which results in more efficient transmission and storage of the

interchange. If a network is sensitive to the data that is transmitted, a compression routine needs a filtering routine to ensure there are no conflicting characters.

There needs to be some way to communicate with your trading partner that data you are sending requires security processing, and there needs to be some way for your trading partner to tell you that data being sent to you requires security processing. ANSI has defined how this communication takes place by defining security segments that are placed within the interchange to signal that security processing is required and the exact nature of that processing. The security segments consist of a security header to flag the start of processing and a security trailer segment to flag the end of security processing. Further, it is defined that security processing can only occur at specific points within an interchange.

An entire functional group can require security processing and is identified by the S1S security header segment and the S1E security trailer segment.

A transaction can require security processing and is identified by the S2S security header segment and the S2E security trailer segment.

Note: It is possible for a transaction to be secured using the S2S and S2E and to exist within a functional group that has been secured with a S1S and S1E. Group and transaction security are independent.

The order of security processing has also been standardized. When data is being prepared for sending (ENVELOPE function), the order is:

1. Authentication
2. Compression
3. Encryption
4. Filtration

When data is being received that requires security processing (DEENVELOPE function), the order is:

1. Filtration
2. Decryption
3. Decompression
4. Authentication

A transaction image is always stored in the Transaction Store in clear text. Security processing takes place during the enveloping and deenveloping processes. During the envelope process:

- A transaction image is retrieved from the Transaction Store.
- Security processing takes place against the image.
- The secured image within an interchange is written to the file associated to the network and the interchange is delivered to the trading partner by the network.

During the deenvelope process:

- The secured interchange is read from the file.
- Security processing takes place against the image.
- The transaction image (now in clear text) is written to the Transaction Store and future translations for the transaction do not require security processing.

The following section describes how security processing is enabled for both the send and receive side, and is followed by detailed descriptions of the interface to the user exit routines that provide the encryption, authentication, compression, and filtration functions.

Note: DataInterchange provides encryption, authentication, and filtration routines that may be used if they suit your needs. These routines are defined in the *DataInterchange Administrator's Guide* in the description for the security profile (SECUPROF).

Enable Security During Send

Enabling security for transactions that you create is a two step process. The first step signals that you want security processing and provides the names for keys that should be used. The second step defines exactly what processing should take place and provides data necessary for the automatic building of the security header and trailers segments (S1S and S1E, S2S and S2E).

1. Security is enabled using the following fields on the Add Trading Partner Usage for Sending panel (TP17).

- Group encryption key name
- Group authentication key name

Note: If either of the above two fields is provided, then during the enveloping process, the group in which this transaction is placed has security processing. S1S and S1E segments are built by DataInterchange and the group is encrypted and/or authenticated.

- Trans encryption key name
- Trans authentication key name

Note: If either of the above two fields is provided, then during the enveloping process, this transaction will have security processing. S2S and S2E segments are built by DataInterchange and the transaction is encrypted and/or authenticated.

2. Security is defined by providing a member in the security profile (SECUPROF). There are two places where the security profile member ID can be specified. If you have unique security requirements for a particular transaction, use the Group security profile member name or the Trans security profile member name field on the Trading Partner Usage Overrides for Sending panel (TP26). The Security ID field in the trading partner profile (TPPROF) contains the default security profile member ID that is used. The security profile member provides the following information:

- Fields to indicate if authentication and/or encryption should take place. If authentication or encryption is asked for in the profile member but a key name is not provided on the Add Trading Partner Usage for Sending panel (TP17), it is not done. If an authentication and/or encryption key name is provided on Add Trading Partner Usage for Sending panel (TP17), but it is not asked for in the profile member, it is not done.
- Fields to indicate what type of filtering, if any, should be done.
- Data that is used to build the S1S and/or S2S security segment.
- The names of programs that provide the security support being requested.

If security has been enabled and defined, DataInterchange invokes the user exits defined in the profile during the enveloping process. The user exits receive the data that needs to be encrypted, authenticated, compressed, or filtered and are expected to return the processed data. DataInterchange automatically builds the required S1S and S1E, or S2S and S2E segments.

Enable Security During Receive

On the receive side, security is self enabled. The S1S and S2S segments within the data being received define the security processing that needs to take place. During the deenveloping process, DataInterchange detects the security segments and invokes the necessary security user exits based on the data received. The user exits that get invoked are defined in the security profile (SECUPROF). The security profile member used is provided in the **Security ID** field of the trading partner profile member.

On the send side, it was possible to have a different set of programs for each transaction. However, on the receive side there can only be one set of programs associated with a trading partner since the security

profile member is pointed to from the trading partner profile member. For more information on how one exit routine can disperse processing to other exit routines, see “Call Exit Routine” on page 4-31.

Security Parameters Introduction

The following sections define the parameters for the security routines of encryption, authentication, compression, and filtration. There is a lot of similarity in the parameters passed to these routines. This is because, at a very high level, there is a lot of similarity in the function performed by each routine. Each of these routines can be passed a tremendous amount of data; each routine processes the data and returns the data to DataInterchange. The amount of data received by these routines is not necessarily the same as the amount of data produced. A compression routine should produce less than it receives, while a filtration routine generally produces more than it receives. An encryption routine usually does not change the length of the data but DataInterchange allows the length to change.

The amount of data processed by these routines can be quite large; sometimes there is too much data to fit in the space available to process it. For this reason, the interface to these routines provides a mechanism for processing the data in pieces. The size of the buffer used to pass data to these routines is controlled by the **buffer size** field specified in the security profile member. If you do not specify a buffer size, DataInterchange obtains a buffer large enough to hold all the data up to 32 K bytes. When one of the security routines is called, the parameters indicate the size of the buffer it is dealing with, the amount of data in that buffer, and the number of residual bytes remaining that would not fit in the buffer. If the amount of data to be processed exceeds **Buffer size**, a **Get data routine** is provided to retrieve the residual data. Once the data is processed, a **Put data routine** is provided so that the results can be returned to DataInterchange. For more information, see “Put Data Routine” on page 4-31. The put data routine can be called multiple times if the amount of data produced exceeds the value of **Buffer size**.

One of the parameters to all of the security routines is the security profile member data block, which is a copy of the data from the security profile member (SECUPROF) that caused the exit routine to be invoked. The COBOL, C, and Assembler definitions for this block are in “COBOL SPDB” on page C-12 and “ASSEMBLER SPDB” on page C-46.

Encryption Routine

The **Encr. program** field in the security profile (SECUPROF) specifies the logical name of an encryption routine. This logical name must match an entry in the ADAMCTL profile, which contains the physical name of the routine and the implementation language.

The list below describes the encryption routine parameters. It is expected to encrypt or decrypt the data and return the results to DataInterchange using a put data routine. For more information, see “Put Data Routine” on page 4-31. As much data as possible is passed to the encryption routine in an input buffer. If more data needs to be processed than can fit in the buffer, a get data routine is provided to obtain this residual data. For more information, see “Get Data Routine” on page 4-30. The encryption routine is provided an output buffer for holding the output data. The size of the output buffer is the same size as the input buffer. A put data routine is provided for putting the results in the buffer. The exit routine must call the put data routine whenever the output buffer is full and at the end of the process to put any data that remains in the output buffer.

A sample encryption routine is provided in “Encryption Examples” on page D-95.

Encryption Parameters: The list below describes the parameters that are passed to an encryption or decryption exit routine. For examples on how the parameters should be declared within Assembler, COBOL, or C programs, see “COBOL definition” on page 4-20, “C definition” on page 4-21, and “Assembler definition” on page 4-22.

The parameter list consists of the following pointers:

- Service name block (SNB)

See “Service Name Block (SNB)” on page A-1 for a detailed description of this block. The **ZSNBNAME** field contains the logical name for the exit (value of the **Encr. program** field from the security profile member). It is possible to combine many exit routines into a single physical load module and if this is done, the value of **ZSNBNAME** can be used to determine the reason the exit is being called.

- Common control block (CCB)

See “Common Control Block (CCB)” on page A-2 for a detailed description of this block. The **ZCCBRC** and **ZCCBERC** fields are used to report any errors found by the exit routine. If the return code (ZCCBRC) and extended return code (ZCCBERC) are not zero, DataInterchange assumes that encryption or decryption failed and logs a message (TR0849) with the return code and extended return code as part of the message. The extended return code (ZCCBERC) field has the following reserved or predefined values:

Value	Description
0	Exit terminated without errors.
1	The ZFCBFUNC function code is not valid.
2 - 3	Reserved.
4	The service requested has not been defined in the ADAMCTL profile.
5 - 10	Reserved.
11	Encryption key name is not known.
12 - 20	Reserved.
21 and higher	Errors defined by the exit routine.

- Function control block (FCB)

See “Function Control Block (FCB)” on page A-4 for a detailed description of this block. **ZFCBFUNC** valid values are:

Value	Description
1	Encrypting
2	Decrypting
3	Assigning an initialization vector

- Encryption handle (4-byte binary value)

Used to get residual data (“Get Data Routine” on page 4-30) and to put results (“Put Data Routine” on page 4-31).

- Key name (16 characters)

The 16-byte key name to be used during encryption or decryption.

- Security data block (SECUDB)

This is the security profile member (SECUPROF) that defined the exit being called.

- Buffer size (4-byte binary value)
The size of the input and output buffers.
- Input buffer (**buffer size**)
The data that should be processed.
- Output buffer (**buffer size**)
A buffer that may be used to hold the resulting data.
- Input data length (4-byte binary value)
The amount of data in the input buffer.
- Residual length (4-byte binary value)
The residual number of characters that need to be processed but could not be put into the input buffer because of size restrictions. If this value is not zero, the exit routine, after processing all the data in the input buffer, can request the residual data by calling a get data routine ("Get Data Routine" on page 4-30).
- Initialization vector (8-byte character value)
If this is a request to obtain an initialization vector (ZFCBFUNC value of 3), this is where the exit routine returns the value. If this is an encryption request, the value returned in this field should be the encrypted initialization vector.

COBOL definition: The COBOL definitions for parameters are shown below. The SNB, CCB, FCB, and SECDB are omitted here but are provided in "COBOL SNB" on page C-3, "COBOL CCB" on page C-2, "COBOL FCB" on page C-3, and "COBOL SPDB" on page C-12.

LINKAGE SECTION.

```
*****
*      HANDLE - USED IN PUTDATA AND GETDATA ROUTINES      *
*****

01  HANDLE                PIC 9(09) COMP.

*****
*      KEYNAME - KEY NAME USED FOR ENCRYPTION/DECRYPTION *
*****

01  KEY-NAME              PIC X(16).

*****
*      THE SIZE OF THE INPUT AND OUTPUT DATA BUFFERS      *
*****

01  BUFFER-SIZE           PIC 9(09) COMP.

*****
*      BUFFER CONTAINING DATA TO ENCRYPT OR DECRYPT          *
*****

01  INPUT-DATA            PIC X(4096).

*****
*      BUFFER CONTAINING THE ENCRYPTED/DECRYPTED DATA        *
*****
```

```
01 OUTPUT-DATA          PIC X(4096).
```

```
*****
*      LENGTH OF DATA IN THE INPUT DATA BUFFER      *
*****
```

```
01 DATA-LENGTH          PIC 9(09) COMP.
```

```
*****
*  AMOUNT OF DATA THAT REMAINS TO BE PROCESSED THAT WOULD *
*  NOT FIT IN THE INPUT DATA BUFFER.  CALLS TO THE GETDATA*
*  ROUTINE SHOULD BE MADE UNTIL RESIDUAL-LENGTH IS ZERO  *
*****
```

```
01 RESIDUAL-LENGTH      PIC 9(09) COMP.
```

```
*****
*  INITIALIZATION VECTOR AREA                          *
*****
```

```
01 INITIALIZATION-VECTOR PIC X(08).
```

```
PROCEDURE DIVISION USING SNB-DATA
                        CCB-DATA
                        FCB-DATA
                        HANDLE
                        KEY-NAME
                        SECDB-DATA
                        BUFFER-SIZE
                        INPUT-DATA
                        OUTPUT-DATA
                        DATA-LENGTH
                        RESIDUAL-LENGTH
                        INITIALIZATION-VECTOR.
```

C definition: The C definitions for parameters are shown below. The SNB, CCB, FCB, and SECDB are omitted here but are provided in “C SNB” on page C-25, “C CCB” on page C-25, and “C FCB” on page C-26.

```
main(snbdata,
     ccbdata,
     fcbdata,
     handle,
     keyname,
     secdata,
     bufsize,
     inbuf,
     outbuf,
     datalen,
     residual,
     vector)
```

```
snb      *snbdata; /* Service name block pointer set up by DI */
ccb      *ccbdata; /* Common block pointer used by DI      */
fcb      *fcbdata; /* Function control block                */
```

```

void    *handle;    /* Handle used in getdata and putdata    */
char    *keyname;   /* Name of key for encryption/decryption    */
secdb   *secdata;   /* Security profile data block              */
long    *bufsize;   /* The size of inbuf and outbuf             */
char    *inbuf;     /* Data to be encrypted or decryption      */
char    *outbuf;    /* Work buffer to hold result data          */
long    *datalen;   /* Amount of data in inbuf                 */
long    *residual;  /* Amount of data remaining                 */
char    *vector;    /* Area for the initialization vector       */

```

Assembler definition: The Assembler definitions for parameters are shown below. The SNB, CCB, FCB, and SECDB are omitted here but are provided in “ASSEMBLER SNB” on page C-37, “ASSEMBLER CCB” on page C-36, “ASSEMBLER FCB” on page C-37, and “ASSEMBLER SPDB” on page C-46.

```

*
* DSECT describing parameters to this routine
* The address of this parameter list is contained in Register 1
* at entry to the program.
*
PARMS    DSECT
SNBDATA  DS      A           Address of the SNB
CCBDATA  DS      A           Address of the CCB
FCBDATA  DS      A           Address of the FCB
HANDLE   DS      A           Address of the handle for getdata/putdata
KEYNAME  DS      A           Address of keyname for encryption/decryption
SECDATA  DS      A           Address of the security data block
BUFSIZE  DS      A           Address of size for INBUF and OUTBUF
INBUF    DS      A           Address of data to be encrypted/decrypted
OUTBUF   DS      A           Address for resultant data
DATALEN  DS      A           Address of amount of data in INBUF
RESIDUAL DS      A           Address of amount of data not in INBUF
VECTOR   DS      A           Address of initialization vector
*
USREXIT  CSECT
        USING PARMS,R1

```

Authentication Routine

The **Auth. program** field in the security profile (SECUPROF) specifies the logical name of an authentication routine. This logical name must match an entry in the ADAMCTL profile, which contains the physical name of the routine and the implementation language.

The authentication routine is passed the parameters described below. It is expected to return a MAC value produced by the authentication process. A message authentication code (MAC) is a cryptographically computed value that is the result of passing text or numeric data through the authentication algorithm using a specific key.

As much data as possible is passed to the authentication routine in an input buffer. If more data needs to be processed than can fit in the buffer, a get data routine is provided to obtain this residual data. For more information, see “Get Data Routine” on page 4-30.

A sample authentication routine is provided in “Authentication Examples” on page D-85.

Authentication Parameters: The list below describes the parameters that are passed to an authentication exit routine. For more information on how the parameters should be declared within Assembler, COBOL, and C programs, see “COBOL definition” on page 4-24 and “C definition” on page 4-25.

The parameter list consists of the following pointers:

1. Service name block (SNB)

See “Service Name Block (SNB)” on page A-1 for a detailed description of this block. The **ZSNBNAME** field contains the logical name for the exit (value of the **Auth. program** field from the security profile member). It is possible to combine many exit routines into a single physical load module and if this is done, the value of **ZSNBNAME** can be used to determine the reason the exit is being called.

2. Common control block (CCB)

See “Common Control Block (CCB)” on page A-2 for a detailed description of this block. The **ZCCBRC** and **ZCCBERC** fields are used to report any errors found by the exit routine. If the return code (ZCCBRC) and extended return code (ZCCBERC) are not zero, DataInterchange assumes that encryption or authentication failed and logs a message (TR0849) with the return code and extended return code as part of the message. The extended return code (ZCCBERC) field has the following reserved or predefined values:

Value	Meaning
0	Exit terminated without errors.
1	The ZFCBFUNC function code is not valid.
2 - 3	Reserved.
4	The service requested has not been defined in the ADAMCTL profile.
5 - 10	Reserved.
11	Authentication key name is not known.
12 - 20	Reserved.
21 and higher	Errors defined by the exit routine.

3. Function control block (FCB)

See “Function Control Block (FCB)” on page A-4 for a detailed description of this block. The **ZFCBFUNC** field will have one of the following values:

Value	Description
1	Sending
2	Receiving

4. Authentication handle (4-byte binary value)

Used to get residual data (“Get Data Routine” on page 4-30).

5. Key name (16 characters)

The 16-byte key name to be used during authentication.

6. Security data block (SECUDB)

This is the security profile member (SECUPROF) that defined the exit being called.

7. Buffer size (4-byte binary value)

The size of the input buffer

8. Input buffer (**buffer size**)

The data that should be processed.

9. Input data length (4-byte binary value)

The amount of data in the **input buffer**.

10. Residual length (4-byte binary value)

The residual number of characters that need to be processed but could not be put into the **input buffer** because of size restrictions. If this value is not zero, the exit routine, after processing all the data in the input buffer, can request the residual data by calling a get data routine ("Get Data Routine" on page 4-30).

11. MAC value (4-byte binary value)

This is where the MAC value produced by the routine should be returned.

COBOL definition: The COBOL definitions for parameters are shown below. The SNB, CCB, FCB, and SECDB are omitted here but are provided in "COBOL SNB" on page C-3, "COBOL CCB" on page C-2, "COBOL FCB" on page C-3, and "COBOL SPDB" on page C-12.

LINKAGE SECTION.

```
*****
*      HANDLE - USED IN PUTDATA AND GETDATA ROUTINES      *
*****
```

```
01 HANDLE                PIC 9(09) COMP.
```

```
*****
*      KEYNAME - KEY NAME USED FOR AUTHENTICATION          *
*****
```

```
01 KEY-NAME              PIC X(16).
```

```
*****
*      THE SIZE OF THE INPUT AND OUTPUT DATA BUFFERS      *
*****
```

```
01 BUFFER-SIZE           PIC 9(09) COMP.
```

```
*****
*      BUFFER CONTAINING DATA TO AUTHENTICATE              *
*****
```

```
01 INPUT-DATA            PIC X(4096).
```

```
*****
*      LENGTH OF DATA IN THE INPUT DATA BUFFER           *
*****
```

```
01 DATA-LENGTH          PIC 9(09) COMP.
```

```
*****
*      AMOUNT OF DATA THAT REMAINS TO BE PROCESSED THAT WOULD *
*      NOT FIT IN THE INPUT DATA BUFFER. CALLS TO THE GETDATA*
*      ROUTINE SHOULD BE MADE UNTIL RESIDUAL-LENGTH IS ZERO  *
*****
```

```
01 RESIDUAL-LENGTH       PIC 9(09) COMP.
```

```
*****
*   MAC VALUE RETURNED                               *
*****
```

```
01  MAC-VALUE          PIC 9(09) COMP.
```

```
PROCEDURE DIVISION USING SNB-DATA
                        CCB-DATA
                        FCB-DATA
                        HANDLE
                        KEY-NAME
                        SECDB-DATA
                        BUFFER-SIZE
                        INPUT-DATA
                        DATA-LENGTH
                        RESIDUAL-LENGTH
                        MAC-VALUE.
```

C definition: The C definitions for parameters are shown below. The SNB, CCB, FCB, and SECDB are omitted here but are provided in “C SNB” on page C-25, “C CCB” on page C-25, and “C FCB” on page C-26.

```
main(snbdata,
     ccbdata,
     fcbdata,
     handle,
     keyname,
     secdata,
     bufsize,
     inbuf,
     datalen,
     residual,
     macvalue)

snb    *snbdata; /* Service name block pointer set up by DI */
ccb    *ccbdata; /* Common block pointer used by DI */
fcb    *fcbdata; /* Function control block */
void   *handle;  /* Handle used in getdata and putdata */
char   *keyname; /* Name of key for authentication */
secdb  *secdata; /* Security profile data block */
long   *bufsize; /* The size of inbuf and outbuf */
char   *inbuf;   /* Data to be authenticated */
long   *datalen; /* Amount of data in inbuf */
long   *residual; /* Amount of data remaining */
long   *macvalue; /* Area for MAC value produced */
```

Assembler definition: The Assembler definitions for parameters are shown below. The SNB, CCB, FCB, and SECDB are omitted here but are provided in “ASSEMBLER SNB” on page C-37, “ASSEMBLER CCB” on page C-36, “ASSEMBLER FCB” on page C-37, and “ASSEMBLER SPDB” on page C-46.

```

*
* DSECT describing parameters to this routine
* The address of this parameter list is contained in Register 1
* at entry to the program.
*
PARMS    DSECT
SNBDATA  DS    A           Address of the SNB
CCBDATA  DS    A           Address of the CCB
FCBDATA  DS    A           Address of the FCB
HANDLE   DS    A           Address of the handle for getdata/putdata
KEYNAME  DS    A           Address of keyname for authentication
SECDATA  DS    A           Address of the security data block
BUFSIZE  DS    A           Address of size for INBUF and OUTBUF
INBUF    DS    A           Address of data to be authenticated
DATALEN  DS    A           Address of amount of data in INBUF
RESIDUAL DS    A           Address of amount of data not in INBUF
MACVAL   DS    A           Address of 4 byte MAC value
*
USREXIT  CSECT
        USING PARMS,R1

```

Compression Routine

The **Comp. program** field in the security profile (SECUPROF) specifies the logical name of a compression routine. This logical name must match an entry in the ADAMCTL profile, which contains the physical name of the routine and the implementation language.

The compression routine is passed the parameters described below. It is expected to compress or decompress the data and return the results to DataInterchange using a put data routine. For more information, see “Put Data Routine” on page 4-31. As much data as possible is passed to the compression routine in an input buffer. If more data needs to be processed than can fit in the buffer, a get data routine is provided to obtain this residual data. For more information, see “Get Data Routine” on page 4-30. The compression routine is provided an output buffer for holding the output data. The size of the output buffer is the same size as the input buffer. A put data routine is provided for putting the results. The exit routine must call the put data routine whenever the output buffer is full and at the end of the process to put any data that remains in the output buffer.

Compression Parameters: The parameters for compression exits are exactly the same as those for filtering exits. See the description in the next section for the parameter description.

Filtering Routine

The **Filtering program** field in the security profile (SECUPROF) specifies the logical name of a filtering routine. This logical name must match an entry in the ADAMCTL profile, which contains the physical name of the routine and the implementation language.

The encryption routine is passed the parameters described below. It is expected to filter or defilter the data and return the results to DataInterchange using a put data routine. For more information, see “Put Data Routine” on page 4-31. As much data as possible is passed to the filtering routine in an input buffer. If more data needs to be processed than can fit in the buffer, a get data routine is provided to obtain this residual data. For more information, see “Get Data Routine” on page 4-30. The filtering routine is provided an output buffer for holding the output data. The size of the output buffer is the same size as the input buffer. A put data routine is provided for putting the results. The exit routine must call the put data routine whenever the output buffer is full and at the end of the process to put any data that remains in the output buffer.

A sample filtering routine is provided in “Test for Filter Type” on page D-65.

Filtering Parameters: The list below describes the parameters that are passed to a filtering or compression exit routine. For more information on how the parameters should be declared within Assembler, COBOL, and C programs, see “COBOL definition” on page 4-28 and “C Definition” on page 4-29.

The parameter list consists of the following pointers:

- Service name block (SNB)

See “Service Name Block (SNB)” on page A-1 for a detailed description of this block. The **ZSNBNAME** field contains the logical name for the exit (value of the **Filtering program** field from the security profile member). It is possible to combine many exit routines into a single physical load module and if this is done, the value of **ZSNBNAME** can be used to determine the reason the exit is being called.

- Common control block (CCB)

See “Common Control Block (CCB)” on page A-2 for a detailed description of this block. The **ZCCBRC** and **ZCCBERC** fields are used to report any errors found by the exit routine. If the return code (ZCCBRC) and extended return code (ZCCBERC) are not zero, DataInterchange assumes the exit failed and logs a message (TR0849) with the return code and extended return code as part of the message. The extended return code (ZCCBERC) field has the following reserved or predefined values:

Value	Description
0	Exit terminated without errors.
1	The ZFCBFUNC function code is not valid.
2 - 3	Reserved.
4	The service requested has not been defined in the ADAMCTL profile.
5 - 20	Reserved.
21 and higher	Errors defined by the exit routine.

- Function control block (FCB)

See “Function Control Block (FCB)” on page A-4 for a detailed description of this block. The **ZFCBFUNC** field will have one of the following values:

Value	Description
1	Filtering or compression
2	Defiltering or decompression

- Filter or compression handle (4-byte binary value).

Used to get residual data (“Get Data Routine” on page 4-30) and to put results (“Put Data Routine” on page 4-31).

- Security data block (SECUDB)

This is the security profile member (SECUPROF) that defined the exit being called.

- Buffer size (4-byte binary value)

The size of the input and output buffers

- Input buffer (**buffer size**)

The data that should be processed.

- Output buffer (**buffer size**)

A buffer that may be used to hold the resulting data. The put data routine must be used to communicate the results back to DataInterchange.

- Input data length (4-byte binary value)

The amount of data in the **input buffer**.

- Residual length (4-byte binary value)

The residual number of characters that need to be processed but could not be put into the **input buffer** because of size restrictions. If this value is not zero, the exit routine, after processing all the data in the input buffer, can request the residual data by calling a get data routine ("Get Data Routine" on page 4-30).

COBOL definition: The COBOL definitions for parameters are shown below. The SNB, CCB, FCB, and SECDB are omitted here, but are provided in "COBOL SNB" on page C-3, "COBOL CCB" on page C-2, "COBOL FCB" on page C-3, and "COBOL SPDB" on page C-12.

LINKAGE SECTION.

```
*****
*      HANDLE - USED IN PUTDATA AND GETDATA ROUTINES      *
*****
```

```
01  HANDLE                PIC 9(09) COMP.
```

```
*****
*      THE SIZE OF THE INPUT AND OUTPUT DATA BUFFERS      *
*****
```

```
01  BUFFER-SIZE           PIC 9(09) COMP.
```

```
*****
*      BUFFER CONTAINING DATA TO FILTER/COMPRESS           *
*****
```

```
01  INPUT-DATA            PIC X(4096).
```

```
*****
*      BUFFER CONTAINING THE FILTERED/COMPRESSED DATA      *
*****
```

```
01  OUTPUT-DATA           PIC X(4096).
```

```
*****
*      LENGTH OF DATA IN THE INPUT DATA BUFFER           *
*****
```

```
01  DATA-LENGTH          PIC 9(09) COMP.
```

```
*****
*      AMOUNT OF DATA THAT REMAINS TO BE PROCESSED THAT WOULD *
*      NOT FIT IN THE INPUT DATA BUFFER. CALLS TO THE GETDATA*
*      ROUTINE SHOULD BE MADE UNTIL RESIDUAL-LENGTH IS ZERO  *
*****
```

```
01  RESIDUAL-LENGTH       PIC 9(09) COMP.
```

```

PROCEDURE DIVISION USING SNB-DATA
                        CCB-DATA
                        FCB-DATA
                        HANDLE
                        SECDB-DATA
                        BUFFER-SIZE
                        INPUT-DATA
                        OUTPUT-DATA
                        DATA-LENGTH
                        RESIDUAL-LENGTH.

```

C Definition: The C definitions for parameters are shown below. The SNB, CCB, FCB, and SECDB are omitted here but are provided in “C SNB” on page C-25, “C CCB” on page C-25, and “C FCB” on page C-26.

```

main(snbdata,
     ccldata,
     fcldata,
     handle,
     secdata,
     bufsize,
     inbuf,
     outbuf,
     datalen,
     residual)

snb      *snbdata; /* Service name block pointer set up by DI */
ccb      *ccldata; /* Common block pointer used by DI */
fcb      *fcldata; /* Function control block */
void     *handle; /* Handle used in getdata and putdata */
secdb    *secdata; /* Security profile data block */
long     *bufsize; /* The size of inbuf and outbuf */
char     *inbuf; /* Data to be filtered or compressed */
char     *outbuf; /* Work buffer to hold result data */
long     *datalen; /* Amount of data in inbuf */
long     *residual; /* Amount of data remaining */

```

Assembler Definition: The Assembler definitions for parameters are shown below. The SNB, CCB, FCB, and SECDB are omitted here but are provided in “ASSEMBLER SNB” on page C-37, “ASSEMBLER CCB” on page C-36, “ASSEMBLER FCB” on page C-37, and “ASSEMBLER SPDB” on page C-46.

```

*
* DSECT describing parameters to this routine
* The address of this parameter list is contained in Register 1
* at entry to the program.
*
PARMS    DSECT
SNBDATA  DS    A           Address of the SNB
CCBDATA  DS    A           Address of the CCB
FCBDATA  DS    A           Address of the FCB
HANDLE   DS    A           Address of the handle for getdata/putdata
SECDATA  DS    A           Address of the security data block
BUFSIZE  DS    A           Address of size for INBUF and OUTBUF
INBUF    DS    A           Address of data to be filtered/compressed
OUTBUF   DS    A           Address for resultant data
DATALEN  DS    A           Address of amount of data in INBUF

```

```

RESIDUAL DS      A          Address of amount of data not in INBUF
*
USREXIT  CSECT
USING PARM,R1

```

Security Support Routines

The following section describes security support routines.

Get Data Routine: Authentication, encryption, filtering, and compression can involve large amounts of data. The size of the buffer the interface uses to pass data to these routines is controlled by the buffer size specified in the security profile member. If the DataInterchange administrator does not specify a buffer size, DataInterchange obtains a buffer that is large enough to hold the data; however, the buffer cannot exceed 32 K bytes.

When DataInterchange calls a security exit routine, the values that DataInterchange passes indicate the size of the buffer being used, the amount of data in the buffer, and the number of residual bytes remaining that would not fit in the buffer. Use the GET DATA routine to retrieve the residual data.

The interface to the GET DATA routine is the DataInterchange language-dependent stub (FXXZccc). The format of the stub follows:

```
FXXZccc(handle,ccb,fcb,buffer,length)
```

The following are parameters of the FXXZccc stub.

Parameter	Description
handle	Indicates the authentication, encryption, compression, or filtering handle that DataInterchange passes as input to the calling routine.
ccb	Indicates the common control block that DataInterchange passes as input to the calling routine.
fcb	Indicates a function control block with a function value of 1. This is a request to get data.
buffer	Indicates the address where DataInterchange should place the residual data. This address can be the same address that DataInterchange passes to the calling routine.
length	Indicates the address of a 4-byte field that contains the maximum number of bytes of residual data that DataInterchange should return.

Identifying GET DATA Routine Return Codes: The GET DATA routine indicates in the CCB whether the get process was successful. The CCB contains the following return codes (RC) and extended return codes (ERC):

These return codes Indicate

RC=8, ERC=1	The function code in the FCB is invalid.
RC=8, ERC=3	There is no data remaining to get.
RC=0, ERC=0	The GET DATA routine returned the data in the specified buffer.

Notes:

1. If the return codes are zero, the buffer contains the data and the length indicates the number of characters the GET DATA routine returned in the buffer.
2. The GET DATA routine decreases the number of residual bytes by the number of bytes returned in this request.

Put Data Routine: The encryption, compression, and filtering routines process input data and create output data that can have the same length as the input data. DataInterchange provides these routines with an output buffer that can be used as a work area in creating the output data. Code your encryption, compression, and filtering routines to return the output data to DataInterchange by calling the PUT DATA routine. The size of the output buffer is the same size as the input buffer.

The interface to the PUT DATA routine is the DataInterchange language-dependent stub (FXXZccc). The format of the stub is:

`FXXZccc(handle,ccb,fcb,buffer,length)`

The following are parameters of the FXXZccc stub.

Parameter	Description
handle	Indicates the encryption, compression, or filtering handle that DataInterchange passes as input to the calling routine.
ccb	Indicates the common control block that DataInterchange passes as input to the calling routine.
fcb	Indicates a function control block with a function value of 2. This is a request to put data.
buffer	Indicates the address where DataInterchange should put the data. This address can be the same address as the output buffer that DataInterchange passes as input to the calling routine.
length	Indicates the address of a 4-byte field that specifies the maximum number of bytes that DataInterchange should put.

Identifying PUT DATA Routine Return Codes: The PUT DATA routine indicates in the CCB whether the put process was successful. The CCB contains the following return codes (RC) and extended return codes (ERC):

These return codes	Indicate
RC=8, ERC=1	The function code in the FCB is invalid.
RC=8, ERC=2	A virtual storage failure occurred while the system put the data.
RC=0, ERC=0	The data is accepted.

Note: When the return codes are zero, the buffer continues to build additional output data if necessary.

Call Exit Routine: During receive processing, you may want to have a routine that inspects control blocks or data, and then calls a secondary exit routine based on the results of that inspection. For example, an exit routine can inspect a service code and call another routine that handles a certain type of filtering. The intermediary routine used to call the secondary routine is a CALL exit routine. The secondary routine that is called with the call exit service receives the same parameters as the original routine.

The interface to the CALL exit routine is the DataInterchange language-dependent stub (FXXZccc). The format of the stub is:

```
FXXZccc(handle,ccb,fcb,routname,0)
```

The following are parameters of the FXXZccc stub.

handle	Indicates the encryption, compression, or filtering handle that DataInterchange passes as input to the calling routine.
ccb	Indicates the common control block that DataInterchange passes as input to the calling routine.
fcb	Indicates a function control block with a function value of 3. This is a request to call a secondary routine.
routname	Indicates the name of the routine that is being called. This name must be 8 characters long, left-justified, and padded with blanks. This name must also be the same as the name that the DataInterchange administrator specifies in the ADAMCTL profile.

Identifying CALL Exit Routine Return Codes: The CALL exit routine indicates in the CCB whether the call process was successful. The CCB contains the following return codes (RC) and extended return codes:

These return codes:	Indicate:
RC=12, ERC=4	The CALL routine could not load or locate the secondary routine that routname specifies.
RC=nn, ERC=nn	Return code from the secondary routine that was called.

Independent Programs

There are two basic types of independent programs that DataInterchange can invoke. There are **response programs** which only apply in the CICS environment and are described in Chapter 5, "Using DataInterchange in the CICS Environment." There are programs that process data extraction records during the PERFORM DATA EXTRACTION and PERFORM MANAGEMENT REPORT commands and the interface to these programs will be described next.

An independent program does not directly participate in the current DataInterchange session. The program is not accessed using the service and exit architecture of DataInterchange but is **linked** using an MVS LINK macro or the EXEC CICS LINK facility. This being the case, there are no language restrictions imposed by DataInterchange for the implementation language of the program. However, in an MVS environment, the parameters passed to the routine can reside above the 16M line and, therefore, the language must be capable of supporting 31-bit addressing mode (AMODE 31).

Data Extract Exit

When performing any of the data extract commands described in "Reporting Using Management Reporting and Transaction Store" on page 1-48, it is possible to provide a program that gets control before the extraction record being written. This program may modify the data or indicate that the record should not be written at all. The name of the program is provided with the **USERPGM** keyword on the PERFORM command and the name provided is the load module name to which DataInterchange will LINK before writing an output record. In MVS, the parameters given to the program are:

- Pointer to a 4-byte binary value that is initialized with a zero. Setting the value of this field to any nonzero value indicates that the extract record should not be written.
- Pointer to the CCB. See "Common Control Block (CCB)" on page A-2 for a detailed description of the CCB.

- Pointer to a 4-byte binary value that contains the length of the extract record.
- Pointer to the extract record.

In CICS, the program receives a COMMAREA with the following format:

Table 4-1. Data Extract Exit Control Block

Name	Format	Size	Offset	Description
Return Code	Binary	4	0	4-byte binary value that is initialized with a zero. Setting the value of this field to any nonzero value indicates that the extract record should not be written.
CCB Address	Binary	4	4	The CCB used to initialize DataInterchange.
Record length	Binary	4	8	4-byte binary value containing the number of bytes in the extraction data record (next field).
Data	Character	1-32738	12	The extract record.

Get/Put Envelope Program

DataInterchange provides the capability of invoking a program that will get control for each interchange created. This program will get control after the interchange has been written to the output file. An exit program is specified in the utility control statements (IEXIT, ITYPE, IAREA and IACCESS) or in the translator control block (TRCB) fields (IUSEREXIT, IUSERATYPE, IUSERAREA, IUSERACCESS). An independent program is defined with an ITYPE value of PG. For more information on the parameters when the exit is a user exit program rather than an independent program, see “Get/Put Envelope Exit and Service” on page 4-13.

This program is invoked through an MVS LINK macro in the MVS environment or the EXEC CICS LINK command in the CICS environment. The control block passed to the program contains the following:

1. The address of the user area. In CICS, this is the address of the utility control block. In MVS, this is the address of the 16 bytes specified by the IAREA keyword.
2. The address of the current translator control block (TRCB).
3. The address of the current trading partner data block (TPPDB).
4. The address of the DataInterchange common control block (CCB).

In CICS, these 4 pointers define the COMMAREA passed to the program. In MVS, Register 1 contains the address of a pointer to the area in storage that contains the 4 addresses above.

```

R1 ----> control blk address ----> +0  user area address
                                     +4  TRCB address
                                     +8  TPPDB address
                                     +C  CCB address

```

Chapter 5. Using DataInterchange in the CICS Environment

Running the DataInterchange Utility in CICS	5-1
Invocation Options	5-1
Passing Control Information	5-2
Determining Results	5-2
CICS Storage Mechanisms	5-3
CICS Envelope Queue Alternatives	5-4
Pre- and Post-Envelope Programs	5-4
Processing Multiple Incoming TS Queues	5-4
Ensuring Serial Processing of DataInterchange Utility Files	5-5
Units of Work and Recovery	5-6
The DataInterchange Utility's Unit of Work	5-7
RECOVERY Utility Keyword	5-9
Tips to Ensure the Unit of Work	5-10
Making the DataInterchange Utility Part of Your Application's Unit of Work	5-10
Terminal Attached Applications	5-11
Running the DataInterchange Utility in a Separate CICS Region	5-11
DB2 Setup Considerations	5-11
CICS Startup Considerations	5-12
Running DataInterchange for CICS in a HOT-DI Environment	5-12
Steps for Implementing HOT-DI	5-12
Initialize DataInterchange	5-12
Initialization Function SYNTAX	5-13
Considerations	5-13
EDI Processing	5-13
Call Utility Services	5-14
Processing Function SYNTAX	5-14
Considerations	5-14
Termination of DataInterchange	5-14
Termination Function SYNTAX	5-15
Considerations	5-15
Outbound Communications	5-15
HOT-DI INITIALIZATION (DIINIT)	5-15
HOT-DI Non-Expedite/CICS - INBOUND DIAGRAM	5-16
HOT DI Non-Expedite/CICS - OUTBOUND DIAGRAM	5-17
HOT DI TERMINATION	5-18
Format of DataInterchange Utility Control Information	5-18
DataInterchange Utility Control Information Field Descriptions	5-20
Continuous Receive Considerations	5-23
Continuous Receive Using MQSeries	5-23
Continuous Receive Unique Selection Criteria	5-24
DataInterchange Processing After Data is Received	5-25
Effect of Defining the EDI1 TD Queue	5-26
Sent to Network Status	5-26
Continuous Receive Without Expedite/CICS	5-26
Response Applications	5-27
Methods of Invocation and Obtaining Results from DataInterchange	5-27
Kinds of Response Applications	5-27
DataInterchange Utility Response Application - Indicator U	5-27
Continuous Receive Response Application - Indicator C	5-28
Transaction Response Application - Indicator T	5-29

Where to Specify the Transaction Response Application	5-31
Persistent Environment	5-31
MVS Subtasks	5-32
The Data Space	5-32
Enablement/Disablement	5-32
Multiple Regions	5-33
Reserved Temporary Storage and Transient Data Queues	5-33
TS Queues that Might Require Additional Processing	5-33
TD Queues Used By Export and Import	5-34
TS Queues Used By Import	5-34
TD Queues EDI1, EDI2, and EDI3	5-34
Interface Between DataInterchange for CICS, Expedite/CICS and Information Exchange	5-35
Information Exchange Session	5-35
Information Exchange Session Cleanup	5-37
Continuous Receive Session	5-38
Starting and Stopping Continuous Receive Sessions	5-38
Continuous Receive Session Cleanup	5-39
Program List Table Considerations	5-41
Processing Program Table Considerations	5-42
DataInterchange Supplied Transactions	5-43
Performance Monitor User Exit	5-44
Using EDIW to Invoke the DataInterchange Utility	5-45

Chapter 5. Using DataInterchange in the CICS Environment

Most functions of DataInterchange operate similarly regardless of the environment in which they are executed. However, the programmer should be aware of the differences that occur before designing applications to use DataInterchange in the CICS environment. This chapter describes how to interface application programs with DataInterchange in the CICS environment and discusses the issues involved. The reader is expected to have a working knowledge of programming in a CICS environment.

Running the DataInterchange Utility in CICS

The DataInterchange Utility is the central point of access to DataInterchange functions and the most commonly used interface. In CICS, you develop an application program to run the DataInterchange Utility. You can write the application in any programming language supported by CICS.

The DataInterchange Utility provides standard CICS interfaces, and can be instructed to use different CICS storage mechanisms. This section describes the issues involved in application programming interaction with the DataInterchange Utility. For a description of the specific DataInterchange functions that can be accessed by the DataInterchange Utility, see Chapter 1, “Using the DataInterchange Utility.”

Invocation Options

The following methods can be used to invoke the DataInterchange Utility:

- The CICS LINK command

Using this command, your application links to the EDIFFUT program. This is the simplest interface you can use. Using this command, the DataInterchange Utility executes synchronously with your application. When the DataInterchange Utility finishes running, it issues a CICS RETURN command to give control back to your application. See “DB2 Setup Considerations” on page 5-11 if you use DB2.

- The CICS START command

Using this command, your application starts the CICS transaction EDIB. The DataInterchange Utility executes in a separate CICS transaction, asynchronously with your application. If you use the CICS START command, consider using response applications, because they are an essential element of the entire process. This will become apparent in “Response Applications” on page 5-27.

- The CICS Automatic Task Initiation (ATI)

To use this method, the CICS systems programmer must first add an intrapartition transient data (TD) queue to the Destination Control Table (DCT) and associate a trigger level and transaction EDIB with the TD queue. As with the CICS START interface, the DataInterchange Utility executes in a separate CICS transaction, asynchronously with your application. If you use ATI, consider using response applications. See “Response Applications” on page 5-27 for more details.

- The EDIW CICS transaction

Use this CICS transaction to enter adhoc PERFORM commands which, in turn, invokes the DataInterchange Utility. This is an easy way to test your utility commands before writing an application program.

Passing Control Information

You must supply control information to the DataInterchange Utility to process your request. See “Format of DataInterchange Utility Control Information” on page 5-18 for the format of the control information.

The method of passing the control information depends on the invocation method you use. The methods are:

1. If you are using the CICS LINK command, you have two options:
 - a. Pass control information through a communications area (COMMAREA). This is specified on the CICS LINK command by the COMMAREA and LENGTH keywords.
 - b. Pass control information through the Transaction Work Area (TWA). Your application should first move the control information into the TWA and then issue the CICS LINK command with no COMMAREA/LENGTH specified. Using this method, the DataInterchange Utility assumes that the control information begins at offset zero of the TWA.
2. If you are using the CICS START command, you pass the DataInterchange Utility the control information through the FROM and LENGTH keywords.
3. If you are using ATI, you issue a CICS WRITEQ TD command to write the control information to the associated TD queue.

Determining Results

The DataInterchange Utility passes the results back to the invoking application or forwards them to a response application. The format of the results is the same as the incoming control information, except that additional results fields are filled in. See “Format of DataInterchange Utility Control Information” on page 5-18.

The results are passed as follows:

1. If you used the CICS LINK command and the control information was supplied to the DataInterchange Utility through the COMMAREA, the results are returned to the same COMMAREA. If you used the TWA to pass control information, the results are returned to the TWA on return to your application.
2. If you used the CICS START command or ATI to initiate the DataInterchange Utility, you should provide a response application to process the results. The response application can be one of the following:
 - a. A program to which the DataInterchange Utility will issue a CICS LINK to give control to the program. When the CICS LINK command is issued, the COMMAREA and LENGTH keywords are used to pass the results. Your application then obtains the COMMAREA address and processes the results.
 - b. A CICS transaction that the DataInterchange Utility will CICS START. When the CICS START command is issued, the FROM and LENGTH keywords are used to pass the results. Your application then obtains the results by issuing a CICS RETRIEVE.

See “Response Applications” on page 5-27 for more details about response applications.

CICS Storage Mechanisms

The DataInterchange Utility supports the use of the following three different types of CICS storage mechanisms and MQSeries Queues for most of the sequential file it uses:

- Temporary Storage (TS) queues

TS queues are useful for files that are processed multiple times, or when recovery is not an issue. When TS queues are used as an output destination, such as a print file, exception file, and query file, they are cleared before they are used. The following three DataInterchange Utility files are the exception. They are appended, instead of cleared, when they are used as output destinations:

- Envelope TS queues

If you want the TS queue cleared, use the CLEARFILE keyword. Using this keyword clears the TS queue before a receive is processed and after a send is processed. If you do not use the CLEARFILE keyword, the DataInterchange Utility appends to the TS queue.

CICS restricts the size of a TS queue to 32 K records. Generally, DataInterchange processes only one envelope TS queue (either inbound or outbound). However, the DataInterchange Utility can process more than one inbound TS queue using the PERFORM DEENVELOPE or the PERFORM DEENVELOPE AND TRANSLATE command. When “deenvelope only” or “translate” is specified in a continuous receive profile member, the Utility automatically processes multiple incoming TS queues associated with it. For more information, see “Processing Multiple Incoming TS Queues” on page 5-4.

- Functional Acknowledgment TS queues

If you want the TS queue cleared, use the CLEARFILE keyword. Using this keyword clears the TS queue before a receive is processed and after a send is processed. If you do not use the CLEARFILE keyword, the DataInterchange Utility appends to the TS queue.

- Output application TS queues created because of a translate-to-application type command.

- ESDS VSAM files

ESDS VSAM files are useful for historical purposes and processing in the MVS environment. The information is saved in the ESDS VSAM file and cannot be easily deleted. ESDS VSAM files are always appended by the DataInterchange Utility. They are never cleared.

- Transient Data (TD) queues

The two main types of TD queues that you can use with the DataInterchange Utility are:

- Extrapartition TD queues

Extrapartition TD queues are MVS sequential data sets. They are useful output mechanisms if further processing is required in MVS. They can also be used as input for data generated in MVS. Extrapartition TD queues are always appended by the DataInterchange Utility. They are never cleared.

- Intrapartition TD queues

Recoverable intrapartition TD queues are the best choice if recovery is a high priority. If recovery is not a high priority, non-recoverable intrapartition TD queues are a good choice.

The destructive read property of intrapartition TD queues is ideal in eliminating the potential of reprocessed data. If the queues are recoverable, the input queues being read are kept in synchronization with all modified resources. This helps clearly define what resources are part of the unit of work. Restart processing is much easier with recoverable TD queues because the queues are always at a complete unit of work boundary. They are never cleared.

- MQSeries Queues

MQSeries queues are specified via DataInterchange MQSeries Queue profile member names and a type of MQ. MQSeries queues are very useful if your application needs to receive from or pass information on to other applications which are not within your CICS region. These other applications could be MVS batch applications, but they also could be applications running on other operating systems such as AIX or Windows. MQSeries allows cross-platform communications between applications. In order to use DataInterchange MQSeries support, you must have MQSeries installed and running on your CICS region.

Warning: Be careful when selecting the storage mechanism for your application's interface with DataInterchange. Data can be reprocessed or lost if the wrong storage mechanism is used.

CICS Envelope Queue Alternatives

Normally, all DataInterchange for CICS envelope files are temporary storage queues (TS queues). In other words, when DataInterchange envelopes data to be sent, the resultant file containing the envelope is normally a TS queue. And when DataInterchange is called to deenvelope incoming data, the file containing the envelope to be deenveloped is normally a TS queue.

This TS queue convention can be overridden. To do so, however, would require a user exit program that uses the Get Envelope Service (on the outbound side) and the Put Envelope Service (on the inbound side). For simple examples on how to do this and for more information, see "Get Envelope Service Example" on page D-117 and "Put Envelope Service Example" on page D-118.

As indicated, to override the envelope TS queue convention requires a user exit program. The name of this program must exist in the ADAMCTL profile. Using the Utility, the exit would get invoked when certain keywords are added to enveloping or deenveloping PERFORM commands. These keywords are IXEIT, IACCESS, and ITYPE. IEXIT is used to specify the user exit program name. IACCESS would be M, ITYPE would be M, and ITYPE would be UE. If using the API, these keywords correspond to Translator Control Block (TRCB) fields, and would be filled in as above.

Pre- and Post-Envelope Programs

It is possible to invoke a user-written program directly after an envelope has been created or after an envelope has been deenveloped. This type of program would be invoked via an EXEC CICS LINK command from DataInterchange. The DFHCOMMAREA would contain pointers to the following blocks: the Utility Control Block (UCB), the Translator Control Block (TRCB), the Trading Partner Data Block (TPPDB), and the Common Control Block (CCB). Using the Utility, this type of program would get invoked when certain keywords are added to enveloping or deenveloping PERFORM commands. These keywords are IXEIT and ITYPE. IEXIT is used to specify the program name and ITYPE would be PG. If using the API, these keywords correspond to Translator Control Block (TRCB) fields, and would be filled in as above. For simple examples on how to do this, see Inbound Envelope Program Example and Outbound Envelope Program Example.

Processing Multiple Incoming TS Queues

The DataInterchange Utility can process up to six inbound TS queues using the PERFORM DEENVELOPE or the PERFORM DEENVELOPE AND TRANSLATE command. This type of processing automatically occurs during continuous receive processing when either "deenvelope only" or "translate" is specified in the continuous receive profile member.

Table 5-1. Multiple TSQ Control Block

Name	Type	Offset	Length	Description
RESERVED	CHAR	0	16	Reserved for DataInterchange
MTSQTSQ1	CHAR	16	8	1st TS queue name
MTSQTSQ2	CHAR	24	8	2nd TS queue name
MTSQTSQ3	CHAR	32	8	3rd TS queue name
MTSQTSQ4	CHAR	40	8	4th TS queue name
MTSQTSQ5	CHAR	48	8	5th TS queue name
MTSQTSQ6	CHAR	56	8	6th TS queue name
RESERVED	CHAR	64	16	Reserved for DataInterchange

When DataInterchange detects that a continuous receive has occurred that involves more than one incoming TS queue, storage is acquired to hold the multiple TSQ control block (see Table 5-1) and it is loaded appropriately with the TSQ names. The address of the multiple TSQ control block is then placed in the Utility control block and the Utility control block multiple TSQ flag is set to Y. Similarly, a user application can invoke the Utility with a PERFORM DEENVELOPE or a PERFORM DEENVELOPE AND TRANSLATE command and, by following the same steps mentioned above for continuous receive, take advantage of multiple TSQ processing. See FFMTFLG and FFMTCBP in Table 5-3 on page 5-19.

The DataInterchange Utility, after invoking all response programs, frees the storage acquired to hold multiple TSQ control blocks. If a continuous receive is set up to not deenvelope or translate (in other words, only a continuous receive response program is to be invoked), it is the responsibility of the response program to free any storage that may have been acquired to hold a multiple TSQ control block. As is consistent with processing in the past, any incoming S@#xxxxx TSQ generated by Expedite/CICS will not be deleted by DataInterchange (or by Expedite/CICS). It is the responsibility of a user response program to delete these queues. If there are multiple TS queues, there will be multiple S@#xxxxx queues.

As previously stated, user applications can invoke the DataInterchange Utility to process multiple incoming envelope TS queues. To do this, the application would acquire storage for a multiple TSQ control block, load it appropriately, and set FFMTFLG and FFMTCBP in the Utility control block. It is important to note that if an application is to invoke the Utility in this manner, only one PERFORM DEENVELOPE or PERFORM DEENVELOPE AND TRANSLATE command should be passed into the Utility per Utility invocation. Multiple incoming envelope TS queues can be processed in this manner using HOT-DI, the UTILSRV API, or regular Utility invocation.

Ensuring Serial Processing of DataInterchange Utility Files

The DataInterchange Utility must ensure serial access to all sequential files it uses. For example, the print file, envelope file, and tracking file are sequential files. If two DataInterchange Utility transactions are executed concurrently, and both are generating interchanges to the same TS queue, the output file may have the interchange records intermixed. To avoid this potential problem, the DataInterchange Utility uses the CICS ENQ and CICS DEQ commands to serialize access to all sequential files. This occurs whenever a file is logically opened. Application programs you develop should also participate in this serialization. The standard resource naming convention for DataInterchange is: *\$EDI.resfile.restype*.

The resource name should always have a total length of 16 characters. The variable *resfile* (the name of the file) should be 8 characters long, left-justified, and padded with blanks. The variable *restype* should be 2 characters long and can have one of the following values:

TM A temporary storage queue (main storage)
TS A temporary storage queue (auxiliary storage)
TD A transient data queue
VS An ESDS VSAM file

For example, the resource name for an application file named **AP01**, which is defined as a transient data queue, would be constructed as follows:

\$EDI.AP01 .TD

Using the CICS ENQ and CICS DEQ commands with the correct resource names ensures that your application references complete information in the sequential files.

Using the previous example, suppose AP01 is an application output file generated as a result of a translation-to-application type operation. If your application issues the proper CICS ENQ command before reading the queue, DataInterchange guarantees that the TD queue contains only complete application data transactions. After reading the TD queue, your application issues the appropriate CICS DEQ command to allow other applications, such as another DataInterchange Utility transaction, access to the queue.

If the CICS ENQ and CICS DEQ commands are not used in this way, your application must handle the possibility of incomplete work.

Units of Work and Recovery

When developing a CICS application, it is important to understand what other application code (the DataInterchange Utility in this case) you will be executing considers a unit of work. In general, the term unit of work is defined as the portion of work between two synchronization points. In the CICS environment, the following are synchronization points:

- CICS Transaction Initiation

Transaction initiation is the point at which the first unit of work within a single CICS transaction begins. This unit of work might be the only unit of work within the transaction if the CICS SYNCPOINT command is not issued.

- CICS SYNCPOINT Command

When an application issues the CICS SYNCPOINT command, it marks the completion of the current unit of work and the start of the next unit of work.

- CICS Transaction Termination

Transaction termination marks the end of the last unit of work within a CICS transaction. At this point, an implicit CICS SYNCPOINT occurs. If no CICS SYNCPOINT commands were issued during transaction execution, transaction termination marks the completion of the only unit of work within the transaction.

CICS also supplies the CICS SYNCPOINT ROLLBACK command. Issuing the CICS SYNCPOINT ROLLBACK command causes the current unit of work to be backed out, as though it never existed. CICS also backs out in flight transactions that were interrupted because of an extraneous event such as cancellation of the CICS job or a power outage.

In the CICS environment, a unit of work consists of updates to recoverable resources. If the unit of work is synchronized, all modifications to recoverable resources are applied. If the unit of work is not synchronized, all modifications are backed out. When a back-out occurs, none of the updates made to recoverable resources are applied. Whether the back-out occurs after one or multiple updates, all changes are removed.

You can consider all updates to recoverable resources within a unit of work as a group. Either the entire group is applied or none are applied, depending on whether the unit of work is synchronized or backed out.

The following lists the resources that can be defined as recoverable and that the DataInterchange Utility might access during execution:

- User Files
 - Print file
 - Exception file
 - Tracking file
 - Report file
 - Envelope TS queues
 - Query file for data extract functions
 - Application input files
 - Application output files

The three recoverable resource options for these files are intrapartition TD queues, recoverable TS queues, and MQSeries Queues. ESDS VSAM files are not recoverable. If you define these files as recoverable, all changes made to them are synchronized if completion of the unit of work is successful, or backed out if completion of the unit of work is unsuccessful.

- Transaction Store
 - DB2 Tables
 - VSAM Files

DB2 tables are inherently recoverable, but the VSAM version of the Transaction Store can be defined as either recoverable or non-recoverable. DataInterchange is designed to take advantage of a recoverable VSAM Transaction Store. For both DB2 and VSAM Transaction Stores, updates can be synchronized or backed out with other recoverable resources within a unit of work.

DataInterchange bases its recovery scheme on CICS and DB2 recoverable resources and the synchronization or back-out of these resources. If a failure occurs and you are using recoverable resources, your application can re-initiate the DataInterchange Utility, passing the same control and even the application information as it was given when the failure occurred.

Note: DataInterchange recovery does not use any type of journaling or checkpointing internally. DataInterchange does not automatically pick up where it left off. It must be re-driven by your application.

The DataInterchange Utility's Unit of Work

The unit of work varies based on the function of the DataInterchange Utility being executed. If you want the DataInterchange Utility to control the unit of work, set the syncpoint interval value to 0 or 1. For more information, see the SYNCVAL field in "Format of DataInterchange Utility Control Information" on page 5-18. The following table lists all DataInterchange Utility commands and the point at which a CICS SYNCPOINT command can be issued.

Table 5-2 (Page 1 of 3). Unit of Work for Each DataInterchange Utility Command

Command	Description
Translate to Standard	Syncpoint issued after complete processing of each EDI transaction.
Translate to Application	
Retranslate to Application	

Table 5-2 (Page 2 of 3). Unit of Work for Each DataInterchange Utility Command

Command	Description
Envelope Reenvelope Translate and Envelope Deenvelope Deenvelope and Translate	Syncpoint issued after complete processing of each EDI interchange. You can override this with RECOVERY(T), forcing a syncpoint after complete processing of each EDI transaction.
Send	Syncpoint issued before communications program invocation to place transactions into SEND STARTED status. This syncpoint is for all interchanges being updated. The status of every record in the DataInterchange Utility that is part of this send is updated, and a syncpoint is issued. Another syncpoint is issued after invoking the communications program to place transactions into SEND REQUESTED or SEND REQUEST ERROR status. This set of syncpoints is repeated for each file of interchanges sent.
Translate and Send Envelope and Send Reenvelope and Send	First a syncpoint is issued after complete processing of each EDI interchange. You can override this with RECOVERY(T), forcing a syncpoint after complete processing of each EDI transaction. After all translation and enveloping is complete, a syncpoint is issued before communications program invocation to place transactions into SEND STARTED status. This syncpoint is for all interchanges being updated. Another syncpoint is issued after invoking the communications program to place transactions into SEND REQUESTED or SEND REQUEST ERROR status. This set of syncpoints is repeated for each file of interchanges sent.
Receive and Deenvelope Receive and Translate	Syncpoint is first issued after the receive is completed and management reporting statistics are recorded to the database. This syncpoint occurs even if management reporting is not active. A syncpoint is issued after complete processing of each EDI interchange. You can override this with RECOVERY(T), forcing a syncpoint after complete processing of every EDI transaction.
Receive	Syncpoint is issued after the receive is completed and management reporting statistics are recorded to the database. This syncpoint occurs even if management reporting is not active.
Purge Unpurge Hold Release	Syncpoint issued after each individual EDI transaction's status is updated (one syncpoint for each EDI transaction). Bundled transactions are an exception. When the entire bundle is updated, the syncpoint is issued.
Print commands All data extract commands Export Close Mailbox Query Custom Layout	Syncpoint is not issued during processing of these commands.
Remove Transactions	Syncpoint issued after every 100 deletes or the number of deletes (NUMDELS) is reached. If STANDALONE(Y) is given, only one syncpoint is issued at the end of the entire run.
Import	Syncpoints issued in DB2 environment only. Syncpoint issued once for each record that is added to the DB2 database.
Update Status Process Network Acks	Syncpoint issued for every interchange for which status is updated (one syncpoint for each EDI interchange updated).
Update Statistics	Syncpoint issued after every 50 updates or the number of updates (NUMUPDTS) is reached.

Table 5-2 (Page 3 of 3). Unit of Work for Each DataInterchange Utility Command

Command	Description
Remove Statistics	Syncpoint issued after every 100 deletes or the number of deletes (NUMDELS) is reached.
Mapping Migration	Syncpoint issued after successful completion of the mapping migration process (one syncpoint for each WHERE clause).

RECOVERY Utility Keyword

For DataInterchange Utility commands that include some type of interchange processing, the unit of work is controlled by the recovery level. The following commands include interchange processing:

- Envelope
- Reenvelope
- Translate and Envelope
- Deenvelope
- Deenvelope and Translate
- Receive and Deenvelope
- Receive and Translate
- Translate and Send
- Envelope and Send
- Reenvelope and Send

The recovery level is established through the RECOVERY utility keyword. The two possible values for RECOVERY are: E for interchange level recovery, and T for transaction level recovery. They are described in detail below:

- Interchange level recovery (the default in CICS)

All recoverable resources associated with the interchange are included in the unit of work scope. This includes input data read from recoverable intrapartition TD queues, Transaction Store DB2 or recoverable VSAM files, records written to the tracking file, and so on.

As discussed previously, if a CICS SYNCPOINT command is issued, the updates to all recoverable resources within the unit of work are applied. If a back-out occurs, all updates are removed from the system and all resources remain in the same state as they were before the unit of work began.

- Transaction level recovery

All recoverable resources associated with each transaction are included in the unit of work scope. This poses a problem for clean recovery because interchange and group records are added to the Transaction Store with the first EDI transaction of every interchange. Because the unit of work is at the transaction level, a CICS SYNCPOINT is issued after each EDI transaction.

If a back-out occurs in the middle of an interchange, transactions previously translated would already be applied to the Transaction Store. To avoid this condition in the CICS environment, it is best to use envelope-level recovery, discussed above.

Tips to Ensure the Unit of Work

When you perform a TRANSLATE TO STANDARD function, there are some situations where the DataInterchange Utility cannot determine the end of a unit of work until it reads application data for the next transaction. In these cases, the unit of work is not limited to one transaction but rather the current transaction and part of the next transaction. Your application can assist DataInterchange in determining the end of the unit of work as follows:

- When using C and D record formatted data, the DataInterchange Utility cannot detect the end of a transaction without reading the next transaction's C record. When you supply a Z record after the last D record in every transaction, the DataInterchange Utility does not read the next C record. This helps the transaction level recovery.

The DataInterchange Utility needs more help with envelope-level recovery. If your application recognizes that a transaction is the last to be placed in an interchange, supply a Z1 record after it. This tells the DataInterchange Utility to end the current unit of work, and forces an interchange break.

- When using raw data, supply the ending structure name in the application data format definition. This prevents the DataInterchange Utility from reading the first structure of the next transaction to determine the end of the current transaction. Unfortunately, if the end structure repeats, this is not possible. To ensure interchange level recovery, do not include interchange breaks within the same input file. By passing separate logical files for each interchange and supplying the ending structure name, the DataInterchange Utility does not perform an extra read to determine the end of interchange condition. This preserves interchange level recovery with the application file.

Making the DataInterchange Utility Part of Your Application's Unit of Work

This is an important issue if you want DataInterchange Utility changes to recoverable resources to be synchronized with changes made by your application to recoverable resources. The field SYNCVAL controls this. For more information, see the SYNCVAL field in "Format of DataInterchange Utility Control Information" on page 5-18. A value of -1 tells the DataInterchange Utility to suppress CICS SYNCPOINT commands. The application is in charge of the unit of work, except when an error occurs while accessing a recoverable resource such as a DB2 table. In this case, the DataInterchange Utility always issues the CICS SYNCPOINT ROLLBACK command regardless of the desired syncpoint interval. This ensures that all changes in the unit of work are backed out in case of error.

The following guidelines are suggested if the DataInterchange Utility is to be part of your application's unit of work:

- Do not use the RECOVERY keyword. Leave the default value E for envelope level recovery. Even though your application controls the unit of work, the DataInterchange Utility is sensitive to the recovery level. Setting the recovery level to E causes DataInterchange to hold Transaction Store updates until all CPU intensive work has completed. This greatly increases the level of concurrency which can be achieved with other CICS transactions performing DataInterchange Utility functions.
- Make all application changes to recoverable resources after the DataInterchange Utility has returned control to your application. The translation process is CPU intensive and DataInterchange Utility processing throughput is greater than an average CICS transaction. By holding updates to recoverable resources until the end of processing, your application experiences greater concurrency with other applications accessing the same recoverable resources.
- Always access your recoverable resources in the same order and consider the DataInterchange Utility a logical recoverable resource. This avoids deadlock potential.

- Do not set the syncpoint interval value to -1 and issue a CICS SYNCPOINT command from a transaction level response program. For more information, see “Response Applications” on page 5-27.
- It is recommended that you issue the CICS SYNCPOINT command in continuous receive response programs and utility termination response programs. For more information, see “Response Applications” on page 5-27.
- Your application should not issue multiple PERFORM statements to DataInterchange within a single unit of work.

Terminal Attached Applications

The throughput time of the DataInterchange Utility varies, based on the work it is requested to process. In general, throughput time is not sub-second. If your application is associated with terminal users and a sub-second response time is mandatory, the DataInterchange Utility should be executed in a separate CICS transaction. This can be accomplished by using the CICS START command or ATI.

You can use response applications to determine whether the execution of the DataInterchange Utility was successful. Response transactions can be initiated through the CICS START command with the TERMID option. This allows your response application to send a message to the associated terminal, and informs the terminal user of the status of the request made to the DataInterchange Utility.

This can be accomplished in two ways. You can:

- Use the RTERMID keyword on the CICS START command when starting transaction EDIB.
- Fill in the Response Terminal ID field in the DataInterchange Utility control information.

Running the DataInterchange Utility in a Separate CICS Region

You can execute the DataInterchange Utility in a separate region from the driving application. All three methods of invocation can be used to initiate the DataInterchange Utility. To use the LINK interface from a remote CICS region, you must use the distributed LINK function provided in CICS/ESA 3.3. Any version of CICS can be used if the DataInterchange Utility is to be executed through the CICS START or ATI commands.

Splitting the DataInterchange Utility into a separate region requires the use of remote resources, such as TD or TS queues, so your CICS systems programmer must be involved if you choose this option. It is much simpler to implement the DataInterchange Utility in the same region as the application, but the DataInterchange Utility provides the flexibility to allow the split, if necessary.

DB2 Setup Considerations

DataInterchange is shipped with a sample Resource Control Table (RCT). DB2 uses the RCT to determine, among other things, DB2 plan authorization. All DataInterchange supplied transactions have access to the DataInterchange DB2 plan. If you are developing an application that issues the CICS LINK command to program EDIFFUT to gain access to DataInterchange Utility services, the CICS systems programmer must add your CICS transaction IDs to the RCT.

CICS Startup Considerations

DataInterchange cannot run with Transaction Isolation ON. The CICS SIT parameter must be TRANISO=NO. The Temporary Storage queue EDICSDA contains a pointer to the Service Director Global Area (CSD). The CSD is shared between DataInterchange transactions EDIA and EDIT, or EDIB and EDIT.

Running DataInterchange for CICS in a HOT-DI Environment

Sub-second response time for EDI business documents has become a major requirement for many DataInterchange users. HOT-DI has been developed to meet the ever-increasing need to improve performance.

The basic concept of HOT-DI is to keep multiple DI tasks running and, thus, eliminating the performance overhead required to start and end the translator for each individual transaction. During normal translation processing, DataInterchange needs information to determine how to process data. The translator must get storage, read translation and validation tables, read profiles, read mapping instructions generated in the control string, and so on. This is all part of DataInterchange's initialization routine. Each execution of the DataInterchange Utility, whether by PERFORM commands or the API, causes the translator to repeat all of these initialization steps.

With HOT-DI, repetitive initialization is eliminated by allowing a user-written application to initialize DataInterchange once, invoke DataInterchange's translation services multiple times, and terminate DataInterchange once. The real benefit is realized when the user initializes multiple HOT-DI sessions to increase translation concurrency and overall throughput. Any commands that can be issued by PERFORM statements can be executed by HOT-DI. However, the intent of HOT-DI is to perform translation at a very high rate. HOT-DI does this particularly well when processing large volumes of small transactions being received or sent to many different trading partners.

HOT-DI can be implemented only in a DataInterchange CICS environment. The HOT-DI concept requires that DI objects, such as mapping and profiles, be read only at initialization time. This means that object updates performed while HOT-DI is executing will not be picked up until the HOT-DI tasks are terminated and re-initialized. This implementation involves using the DataInterchange Application Programming Interface (API) and Utility Service, which requires user-written applications or tasks. This can be achieved several ways. One scenario is outlined below:

Steps for Implementing HOT-DI

HOT-DI implementation requires DataInterchange for CICS and can only be used with Expedite/CICS when using the PERFORM commands to send data. It is not currently supported by environments using DataInterchange's continuous receive mailbox function.

Initialize DataInterchange

A user-written CICS task, for example (**DIINIT**), is required to initialize DataInterchange. After initialization, this task should end. Information and pointers to storage areas required by DataInterchange are kept in a storage area called the Common Control Block, or CCB. Part of the initialization process is to save this CCB so that the following translation requests can bypass the initialization phase.

To obtain an understanding of the initialization function request SYNTAX and control block parameters, see "API - Initialization" on page 3-17.

Initialization Function SYNTAX

FXXZccc(SNB,CCB,FCB,applid,sysid)

- The DataInterchange Service Name Block (SNB) field **ZSNBNAME** should be set to **ENVSERV**.
- The DataInterchange Function Control Block (FCB) field **ZFCBFUNC** should be set to **4**. This value in the ZFCBFUNC tells DataInterchange to use the shared storage option when acquiring main storage.
- The DataInterchange Common Control Block (CCB) must be different for each initialized DataInterchange task. The CCB must be used when requesting other DataInterchange utility services; for example, translation. The CCB area obtained before initialization and passed for all subsequent calls is obtained through a shared GETMAIN; for example:

```
EXEC CICS
    GETMAIN SET (CCB-POINTER) LENGTH (608) SHARED
END-EXEC
```

Considerations

The CICS task (for example, DIINIT) can initialize HOT-DI as many times as your system's storage allows (usually five or six tasks). A main storage area must be acquired for each HOT-DI to be initialized. The size of the storage is equal to the size of a CCB. The address of each CCB can be saved in a Temporary Storage Queue or a Transient Data Queue, the names of which are known by DIINIT. By placing the CCB address in a TS Queue, external access to initialized, HOT-DI CCBs is achieved. A user flag should also be associated with each CCB to show the status of processing (either AVAILABLE, indicating no current translation is being performed or BUSY, indicating a translation or other DataInterchange service is in process).

EDI Processing

A user-written CICS task (for example, GETDI) controls when to start a desired DataInterchange service (such as translation) in one of the HOT-DI sessions. To start the service, the CICS task needs to know the user service desired and the name of an AVAILABLE HOT-DI. To start multiple concurrent HOT-DI tasks, you must use separate print, exception, reporting, tracking, and query files to maintain concurrency. For more information, see "DataInterchange for CICS Considerations" on page G-3.

The CICS task (**GETDI**) will:

- Loop until an event occurs that signals EDI translation or desired user service.
- Obtain an AVAILABLE HOT-DI session (querying the user flag associated with the HOT-DI CCB address). This may involve reading a list of TS Queue names of started HOT-DIs.
- Set the user flag to BUSY for the AVAILABLE HOT-DI.
- Start a CICS task, for example (**DIPROC**), for processing using the CCB identifier (address). The user-written CICS task (**DIPROC**) will use the CCB identifier to call the Utility Service for processing.
- After completion of processing, set the user flag from BUSY to AVAILABLE.

Call Utility Services

To obtain an understanding of the utility service function request SYNTAX and control block parameters, see “API - Utility Services” on page 3-18.

Processing Function SYNTAX

FXXZccc(SNB,CCB,FCB,UTILCB)

- The DataInterchange Service Name Block **SNB** field **ZSNBNAME** should be set to **UTILSRV**.
- The DataInterchange Common Control Block **CCB** should be the CCB identifier passed from the allocate step.
- The DataInterchange Function Control Block **FCB** field **ZFCBFUNC** should be set to **2**, which tells DataInterchange to process in HOT-DI mode.
- The BATFLG flag should be set, indicating whether or not the utility is being invoked from the API. When the API is used to call the UTILSRV service (as in the case of HOT-DI), this field must be set to B.
- The DataInterchange Utility Control Information (UTILCB) should be initialized appropriately for the processing request. The format of the control block is described in the “Format of DataInterchange Utility Control Information” on page 5-18.

Considerations

Your user-written program may interrogate DataInterchange's API return codes and extended return codes, which are available in the CCB. The **ZCCBRC** field contains the API return code, which is equivalent to DataInterchange's Utility condition code (UTILSEV). The **ZCCBERC** field contains the API extended condition code, which is equivalent to DataInterchange's Utility extended code (UTILCCODE).

| For descriptions of these codes, see Appendix B, “DataInterchange Utility Condition Codes and API Return Codes.”

| The mapping global accumulators are not reset between HOT-DI invocations. If a global variable is used to conditionally execute a map switch using the DataInterchange variable DIMAPSWITCH, then you must be careful to ensure that global variables are handled appropriately in both maps.

Termination of DataInterchange

A user-written CICS task, for example (**DITERM**), is needed to terminate each active HOT-DI task.

Termination involves making a DataInterchange termination request using the CCB and then freeing the main storage area in which the CCB existed.

To obtain an understanding of the termination function request SYNTAX and control block parameters, see “API - Termination” on page 3-19.

Termination Function SYNTAX

FXXZccc(SNB,CCB,FCB)

- The DataInterchange Service Name Block **SNB** field **ZSNBNAME** should be set to **ENVSERV**.
- The DataInterchange Function Control Block **FCB** field **ZFCBFUNC** should be set to **2**, which tells DataInterchange to use shared GETMAINS.

Considerations

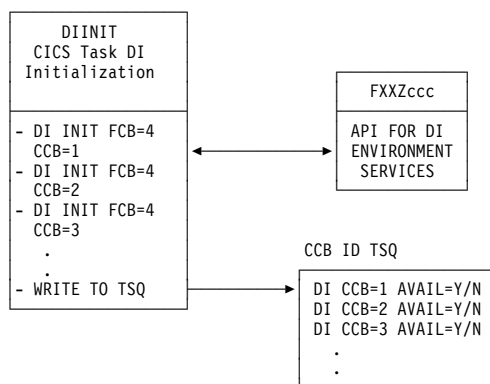
The Termination step should loop if any CCB is BUSY for a specified number of attempts on each DataInterchange until all tasks have been terminated or the specified number of attempts to terminate has been reached.

Outbound Communications

Outbound Communications can be requested using DataInterchange's Application Program Interface with Utility Services or Communication Services.

To obtain an understanding of the utility service or communication service function request SYNTAX and control block parameters, see Chapter 1, "Using the DataInterchange Utility," and "Communication Services" on page 3-105 .

HOT-DI INITIALIZATION (DIINIT)



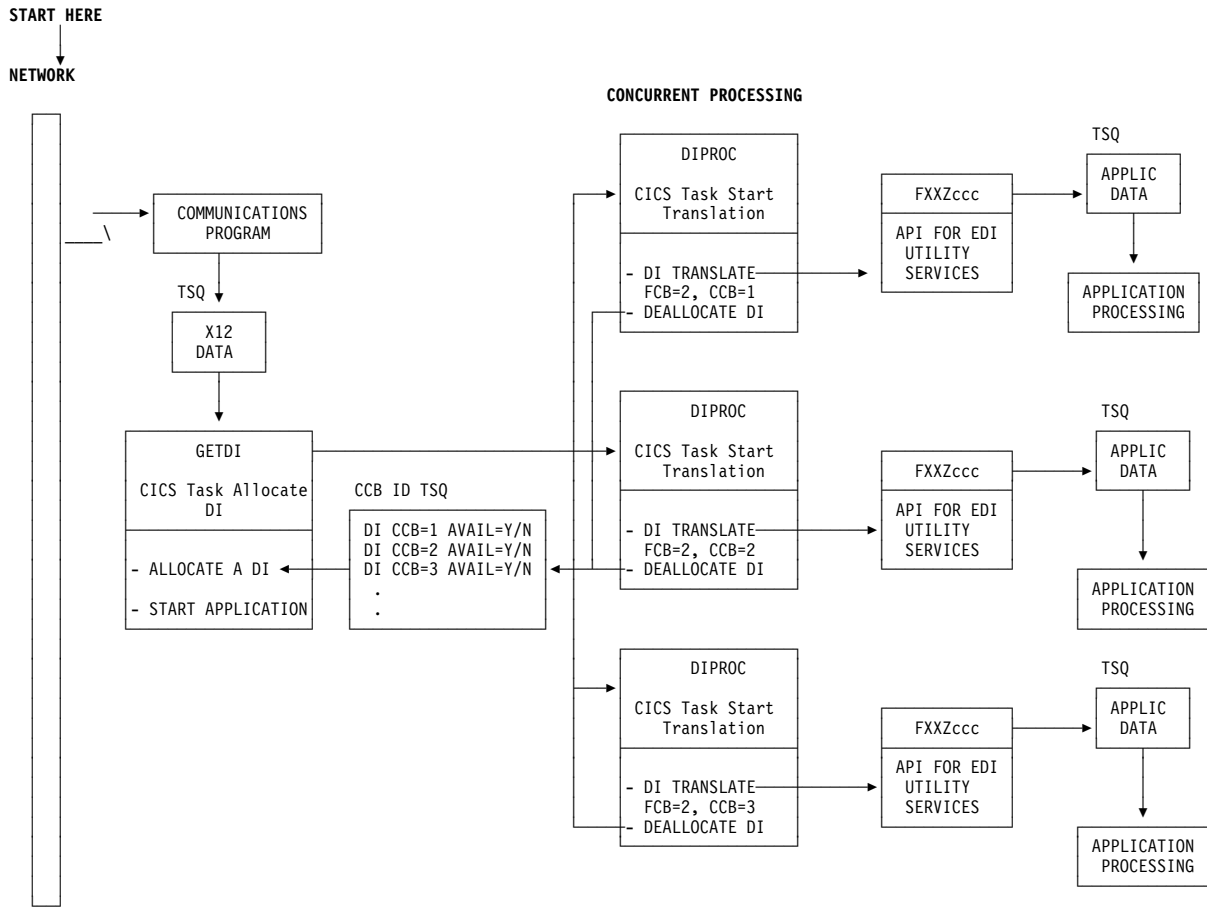
DIINIT

FXXZccc(SNB,CCB1,FCB,applid,sysid)

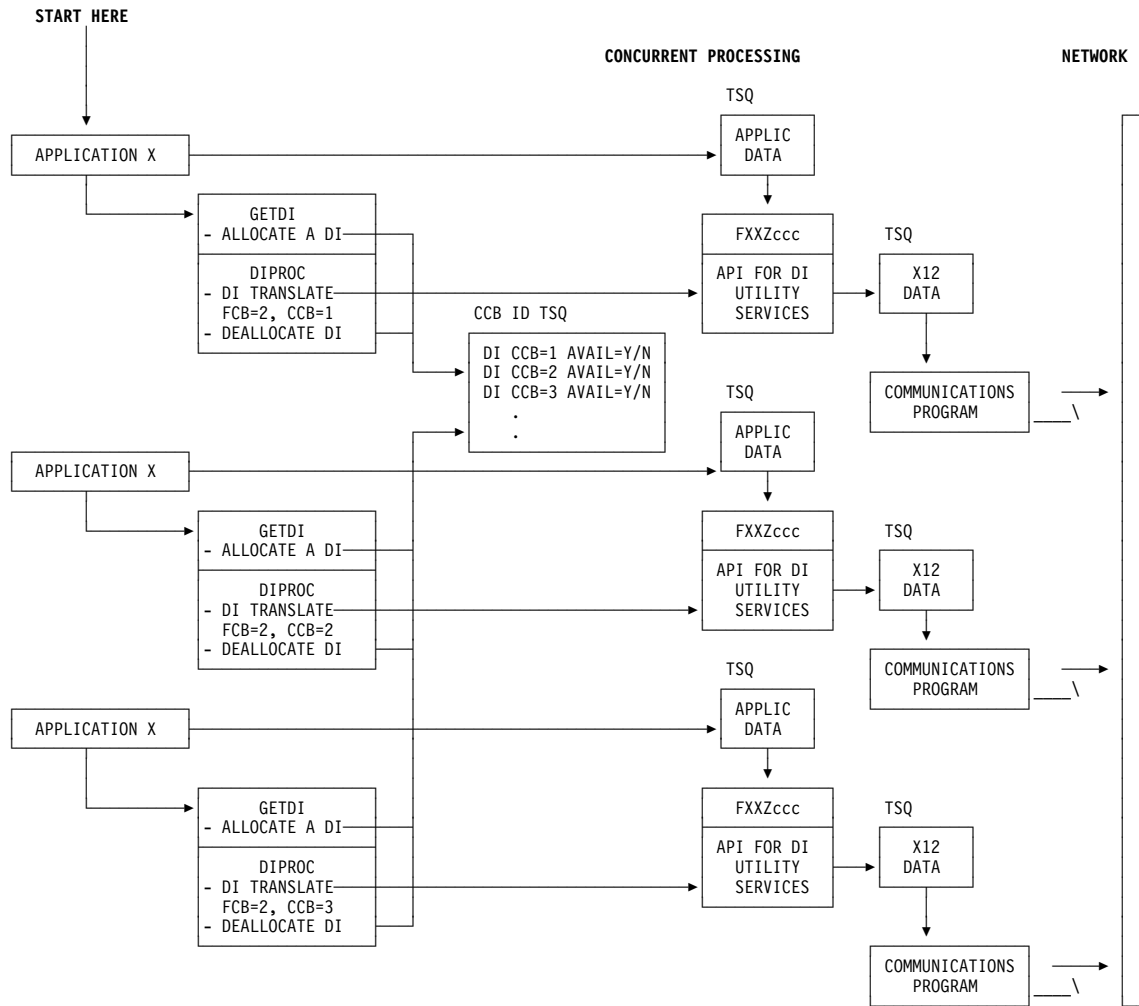
FXXZccc(SNB,CCB2,FCB,applid,sysid)

FXXZccc(SNB,CCB3,FCB,applid,sysid)

HOT-DI Non-Expedite/CICS - INBOUND DIAGRAM

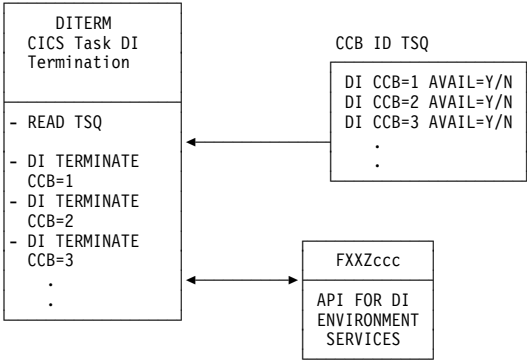


HOT DI Non-Expedite/CICS - OUTBOUND DIAGRAM



| For sample HOT-DI programs, refer to Appendix D, "Sample Programs."

HOT DI TERMINATION



DITERM
FXXZccc(SNB,CCB1,FCB)
FXXZccc(SNB,CCB2,FCB)
FXXZccc(SNB,CCB3,FCB)

Format of DataInterchange Utility Control Information

When your application invokes the DataInterchange Utility, control information must be passed. The following table describes the format of the area given to the DataInterchange Utility. The table includes the resulting control information given back to your application or passed on to your response application.

Here is a description of the columns within the table:

Column	Description
Name	The logical name of the field. The name can be referenced elsewhere within DataInterchange documentation.
Type	This field indicates whether the field is a character field or a binary field. Character fields that are not used should be initialized to blanks. Binary fields that are not used should be initialized to low values or binary 0s.
Offset	This identifies the displacement of the field within the block relative to offset 0.
Length	The length of the field in number of bytes.
Result	A value of Y indicates that the field is set by DataInterchange before control is returned to the initiating application or passed on to a response application. These fields should be inspected by the initiating program or by the response programs to determine if the execution of the DataInterchange Utility was successful. Some of these fields also contain names of TS queues that might need to be processed further. A value of X specifies that the field is used internally by DataInterchange and should be initialized with blanks (or binary zeros for binary fields) and not modified after initialization. Note: All fields will be given to your application, not just the fields set by DataInterchange.
Description	A brief description of the field. See "DataInterchange Utility Control Information Field Descriptions" on page 5-20 for a detailed description of each field.

Table 5-3 (Page 1 of 2). Format of DataInterchange Utility Control Information

Name	Type	Offset	Length	Result	Description
SYNCVAL	BIN	0	4		Units of work before syncpoint
CMDP	BIN	4	4		Command string address
CMDLEN	BIN	8	4		Command string length
CMDNAME	CHAR	12	8		Command file name
CMDTYPE	CHAR	20	2		Command file type
DELIMITER	CHAR	22	1		Command value delimiter
PRTNAME	CHAR	23	8		Print file name
PRTTYPE	CHAR	31	2		Print file type
RPTNAME	CHAR	33	8		Report file name
RPTTYPE	CHAR	41	2		Report file type
EXCPNAME	CHAR	43	8		Exception file name
EXCPTYPE	CHAR	51	2		Exception file type
TRAKNAME	CHAR	53	8		Tracking file name
TRAKTYPE	CHAR	61	2		Tracking file type
QRYNAME	CHAR	63	8		Query file name
QRYTYPE	CHAR	71	2		Query file type
APPLID	CHAR	73	8		Override application ID
LANGID	CHAR	81	6		Override language ID
RESPID	CHAR	87	8		Response program/transaction ID
RESPTYP	CHAR	95	2		Response type
RTERMID	CHAR	97	4		Response terminal ID
USRFLD	CHAR	101	16		User field to pass data
SYSID	CHAR	117	8	X	CICS system ID override
RESPFLAG	CHAR	125	1	Y	Response indicator
FILETYP	CHAR	126	2	Y	Network ack file type
ECBP	BIN	128	4		ECB address
CCBRC	BIN	132	4	Y	Severity Code (UTILSEV)
CCBERC	BIN	136	4	Y	Condition code (UTILCCODE)
FILEID	CHAR	140	8	Y	Envelope TSQ/Net ack file
APPFILE	CHAR	148	8	Y	Translated data TS Queue
ABNDPCODE	CHAR	156	4	Y	CICS Abend code if one occurred
THANDLE	CHAR	160	20	Y	Transaction handle
FFIEHDR	BIN	180	4	Y	IE long header address
FARC	BIN	184	4	Y	Functional ack return code
FAERC	BIN	188	4	Y	Functional ack ext return code
FABUILT	CHAR	192	1	Y	Functional ack built flag
FFMTFLG	CHAR	193	1		Multi-TSQ indicator flag
USERSYNC	CHAR	194	1		User syncpoint flag
RES1	CHAR	195	1	X	Reserved for DataInterchange
USERCOND	BIN	196	4	X	User condition code pointer
RES2	CHAR	200	23	X	Reserved for DataInterchange
RESPACTV	CHAR	223	1	X	Response program active flag
WORKNAME	CHAR	224	8	X	Name of workfile

Table 5-3 (Page 2 of 2). Format of DataInterchange Utility Control Information

Name	Type	Offset	Length	Result	Description
BATFLG	CHAR	232	1		Batch or UTILSRV API (HOTDI) flag
NOEXCP	CHAR	233	1	X	No exception file
INVPARM	CHAR	234	1	X	Network program invalid parm
LOGACTV	CHAR	235	1	X	Logging CE0050 active
FFUSADDR	BIN	236	4	X	Pointer to this block
CCBP	BIN	240	4	X	Pointer to the CCB
FFMTCBP	BIN	244	4		Multi-TSQ control block pointer

DataInterchange Utility Control Information Field Descriptions

Table 5-4 describes each field supplied by your application and set by DataInterchange.

Table 5-4 (Page 1 of 4). DataInterchange Utility Control Information Field Descriptions

Keyword	Description
SYNCVAL	The syncpoint value is the number of units of work before DataInterchange issues a CICS SYNCPOINT command. A value of 0 or 1 specifies that DataInterchange issues syncpoints after each unit of work (the default). A value of -1 tells DataInterchange that the application will handle syncpoints and that DataInterchange will not issue any syncpoints. Any other value can be used to decrease the number of syncpoints DataInterchange issues. See Table 5-2 on page 5-7 for a complete description of the unit of work for each command. This value must be specified as a 4-byte binary value. The PERFORM IMPORT command ignores this value.
CMDP	The command string address of the command language input in main storage. This should be used only if the application remains active while DataInterchange is running. The storage is treated as read-only, so translation to uppercase does not occur. All keywords must be in uppercase. Passing the commands in main storage improves the performance of the DataInterchange Utility.
CMDLEN	The command string length of the command language string in main storage. If the main storage option is not used, you must set this field to 0, to identify that the command file name (CMDNAME) and command type (CMDTYPE) fields should be used instead. This value must be specified as a 4-byte binary value.
CMDNAME	The command file name of a temporary storage queue, transient data queue, or VSAM entry sequenced data set that contains the command language input to be processed. Uppercase translation occurs, so the command language syntax is free form. The default value is SYSIN.
CMDTYPE	The command file type of the command file. Valid values are MQ, TD, TM, TS, and VS. The default value is TS.
DELIMITER	This command value delimiter can be used in place of the left and right parentheses to enclose values in the DataInterchange Utility command language. You must supply this value if a command contains the application control field keyword, ACFIELD, and its value contains either a left or right parenthesis.
PRTNAME	The name of the print file. For a description of this file, see "Print File (PRTFILE)" on page 2-5. The default value is PRTFILE.
PRTTYPE	The type of print file. Valid values are MQ, TD, TM, TS, and VS. The default value is TS.
RPTNAME	The name of the report file. For a description of the report file, see "Report File (RPTFILE)" on page 2-6. The default value is RPTFILE.
RPTTYPE	The type of report file. Valid values are MQ, TD, TM, TS, and VS. The default value is TS.
EXCPNAME	The name of the exception file. For a description of this file, see "Exception File (FFSEXCP)" on page 2-4. The default value is FFSEXCP.
EXCPTYPE	The type of the exception file. Valid values are MQ, TD, TM, TS, and VS. The default value is TS.
TRAKNAME	The name of the tracking file. For a description of this file, see "Tracking File (FFSTRAK)" on page 2-5. The default value is FFSTRAK.
TRAKTYPE	The type of the tracking file. Valid values are MQ, TD, TM, TS, and VS. The default value is TS.
QRYNAME	The name of the query file. For a description of this file, see "Query File (EDIQUERY)" on page 2-7. The default value is EDIQUERY.
QRYTYPE	The type of query file. Valid values are MQ, TD, TM, TS, and VS. The default value is TS.

Table 5-4 (Page 2 of 4). DataInterchange Utility Control Information Field Descriptions

Keyword	Description
APPLID	An override application ID that initialized DataInterchange. The activity log profile must contain a matching entry to define the log file used for recording errors and events pertaining to the application. The default is EDIFFS.
LANGID	The override language ID identifies which language version is being used. It can have the following value: ENU English (the default) Japanese
RESPID	The response program ID is the name of a program or transaction that DataInterchange should invoke after it has finished processing. See "Response Applications" on page 5-27 for more details.
RESPTYP	The type of response name. Valid values are: PG A program that should be linked to. TX A transaction that should be started.
RTERMID	The response terminal ID is a value that should be used with the keyword TERMID on the CICS START command if the response name is a transaction.
USRFLD	The user area used to pass 16 bytes of user data for the specific use of the application. This field is passed to each response application when it gains control. See "Response Applications" on page 5-27 for additional information.
SYSID	CICS system ID override. For DataInterchange use only.
RESPFLAG	The response indicator identifies which action caused the response program to be invoked. Valid values are: U The response program was invoked because of the completion of the DataInterchange Utility. C The response program was invoked because of the completion of a continuous receive. The ENVFILE field contains the name of the TS queue holding the interchange processed during this continuous receive. If the continuous receive was on behalf of a network acknowledgment, the ENVFILE field contains the name of the TS queue holding the network acknowledgment. T The response program was invoked because of a translation to application function. The APPFILE field contains the name of a TS queue holding data in the application format. The HANDLE field contains the transaction handle associated with the transaction.
FILETYP	This field contains the file type of the network acknowledgment file if the DataInterchange Utility command PERFORM PROCESS NETWORK ACKS was executed. Possible values are MQ, TD, TM, TS, and VS. The field FILEID contains the name of the network acknowledgment file.
ECBP	The address of an ECB that DataInterchange posts when it finishes processing. This is useful if synchronous processing is desired, but the application wants to keep the DataInterchange Utility out of its syncpoint interval. Your application will CICS START transaction EDIB and issues a CICS WAIT EVENT on the ECB. If an ECB is not used, this field must be set to 0.
CCBRC	Severity code. This field may also be referred to as UTILSEV. Possible values are: 0 No errors were detected. 4 A low-severity warning was detected, but processing was not impaired. 8 An error was detected that prevented the request from completing successfully. 12 A severe error was detected that prevented the request from completing successfully. -1 An abend occurred during DataInterchange processing.
CCBERC	DataInterchange Utility condition code. This field may also be referred to as UTILCCODE. See <i>DataInterchange Messages and Codes</i> for more information.
FILEID	The envelope file name is the temporary storage queue holding the interchange that was received with the Continuous Receive Facility. It is the response application's responsibility to delete or process the TS queue further. This field applies only when the response application is invoked at the completion of a continuous receive. If the continuous receive is set up for network acknowledgments, this field specifies the name of the TS queue that contains the network acknowledgment processed. If the TS queue contains a network acknowledgment, DataInterchange deletes the TS queue upon return from your continuous receive response application. If the PROCESS NETWORK ACKS command is used, the field contains the name of the TS queue, TD queue, or ESDS VSAM file containing the network acknowledgment. Here, the FILETYP field contains the associated type.

Table 5-4 (Page 3 of 4). DataInterchange Utility Control Information Field Descriptions

Keyword	Description
APPFIL	The application data TS queue is a temporary storage queue holding the application data in C and D format, or raw data format. This tells the response application that the TS queue contains one transaction in application format. It is the response application's responsibility to delete or process the TS queue further. This field applies only when the response application is invoked because a translation or retranslation request is being processed.
ABND	If an abend occurs during DataInterchange processing, this field is set to the CICS abend code, and the UTILSEV field is set to -1, indicating there is a value in the ABND field. If an abend occurs, DataInterchange also issues the CICS DUMP command with an associated dump code of EDI1.
THANDL	The character representation of the Transaction Store handle of the translated transaction. The handle is always given, regardless of data actually generated from the translation. Data is not generated if the error level exceeds the acceptable value for this transaction. This value is supplied only when the response program is invoked because of a translation-to-application type function. It is not supplied if the response application is invoked, because of continuous receive or DataInterchange Utility completion.
FFIEHDR	The address of the Information Exchange receive message long header given by Information Exchange to Expedite/CICS due to the continuous receive. This field is only passed when the utility is invoked due to a continuous receive. The format of the Information Exchange receive message header is described in the <i>Information Exchange Interface Programming Guide</i> . This address will only be set under the following conditions: <ul style="list-style-type: none"> If either the Translate or Deenvelope value is set to Y in the associated continuous receive profile member, the EDI1 TD queue must be defined. If EDI1 is not defined, the address will not be set. If both the Translate and Deenvelope values are set to N, the Response Type value must be set to PG in the associated continuous receive profile member. In other words, if only a response application (no other DataInterchange processing is requested) is to be invoked due to a continuous receive, the application must be a program and not a CICS transaction to receive this field.
FARC	The largest return code encountered during functional acknowledgment processing.
FAERC	The largest extended return code encountered during functional acknowledgment processing.
FABUILT	A flag indicating whether functional acknowledgments were generated. A value of Y specifies s functional acknowledgments were generated. A value of space means no functional acknowledgments were generated.
FFMTFLG	A flag set during continuous receive to indicate whether incoming data spans more than one temporary storage queue. This field may also be set by user applications to identify multiple envelope queues used in association with a PERFORM DEENVELOPE or a PERFORM DEENVELOPE AND TRANSLATE command. A value of Y indicates more than one envelope queue. This field works in tandem with FFMTCBP. See "Processing Multiple Incoming TS Queues" on page 5-4.
USERSYNC	This field must be set by response applications when the user is controlling syncpointing (SYNCVAL is -1) and the response application issues a syncpoint, a commit, or a rollback. This field tells DataInterchange that a commit has occurred. A value of Y indicates that the response application issued a commit. Otherwise, this field should be blank.
RES1	Reserved for DataInterchange
USERCOND	User condition code pointer. For DataInterchange use only.
RES2	Reserved for DataInterchange.
RESPACTV	Response program active flag. For DataInterchange use only.
WORKNAME	Name of workfile. For DataInterchange use only.
BATFLG	A flag indicating whether the utility is being invoked from the API or not. When the API is used to call the UTILSRV service (as in the case of HOTDI), this field must be set to B. Otherwise, it should be blank. If the utility program EDIFFUT or the CICS transaction EDIB is being invoked, this field should be blank. When the value in this field is B, the utility will initialize the Syncpoint service and open the print file for output, rather than extend.
NOEXCP	No exception file flag. For DataInterchange use only.
INVPARM	Invalid parameter passed to network program flag. For DataInterchange use only.
LOGACTV	Logging of CE0050 messages is active flag. For DataInterchange use only.
FFUSADDR	Pointer to this block. For DataInterchange use only.
CCBP	Pointer to the Common Control Block (CCB). For DataInterchange use only.

Table 5-4 (Page 4 of 4). DataInterchange Utility Control Information Field Descriptions

Keyword	Description
FFMTCBP	This field is set during continuous receive with the address of a multiple TSQ control block, if the incoming data spans more than one temporary storage queue. A user application can also set this field with the address of a multiple TSQ control block used in association with a PERFORM DEENVELOPE or a PERFORM DEENVELOPE AND TRANSLATE command. This field works in tandem with FFMTFLG. The value in this field must be a valid multiple TSQ control block address if FFMTFLG is set to Y. See "Processing Multiple Incoming TS Queues" on page 5-4.

Continuous Receive Considerations

The Continuous Receive Facility is a DataInterchange service that works in conjunction with Expedite/CICS and Information Exchange. It also can be used with MQSeries trigger queues. By defining members of a continuous receive profile, you can completely automate the receive process. For more details on each field contained within the profile, see the continuous receive profile definition in the *DataInterchange Administrator's Guide*. Using the Continuous Receive Facility, you can perform the following:

- Receive and deenvelope standard data
- Translate the standard data-to-application format
- Automatically initiate transaction-level response applications that process application data placed in temporary storage queues
- Automatically receive and process network acknowledgments

Because Information Exchange passes the data to the host system immediately after it enters the mailbox, DataInterchange gains control shortly after data is sent from the trading partner's system.

With EDI data and transaction level response programs, your application receives control immediately after DataInterchange generates the application data. When using continuous receive for network acknowledgments, the status of EDI transactions in your Transaction Store is kept up to date without having to initiate the PERFORM UPDATE STATUS command at different points during the day.

Continuous Receive Using MQSeries

In addition to Expedite/CICS and Information Exchange, MQSeries also triggers continuous receive processing. Much like an Information Exchange mailbox, you have your trading partners deliver envelopes to MQSeries queues. Follow these steps to use MQSeries.

1. The CICS and MQSeries administrators must enable the CICS region for MQSeries support. As part of this process, they should define an MQSeries trigger event queue, which the MQSeries-supplied transaction CKTI monitors within your region. For example, assume CICS1.TRIGGER is the name of this queue.
2. The MQSeries administrator must define one MQSeries process via the DEFINE PROCESS command to provide MQSeries with information about the DataInterchange transaction EDIQ. A sample of the DEFINE command follows:


```
DEFINE PROCESS(EDIPROCESS) APPLICID(EDIQ) APPLTYPE(CICS)
```
3. Using the MQSeries DEFINE QLOCAL command, the MQSeries administrator must define at least one MQSeries queue to receive envelopes. A sample of the DEFINE command follows:

```

| DEFINE QLOCAL(EDIRECEIVE) INITQ(CICS1.TRIGGER)
|   PROCESS(EDIPROCESS) TRIGGER TRIGTYPE(FIRST)
|   TRIGDATA('CRPROF=MQ1 MQPROF=RCV1')

```

The TRIGDATA value is critical. Within this field are two keyword-value combinations. The first is CRPROF=, which indicates the name of the DataInterchange Continuous Receive profile member to use when data is actually receive. The second is MQPOF= which relates a DataInterchange MQSeries profile member to this specific MQSeries queue. Both of these keyword-value combinations are mandatory. The order in which the two keyword-value combinations are specified is not important.

In this example only one MQSeries queue will be used for event-driven EDI but you may define any number of queues. As long as the DataInterchange and MQSeries information are set up correctly, DataInterchange will process any number of queues.

The MQSeries definitions are now complete.

4. Within DataInterchange, you must define an MQSeries queue profile member. In this example, the name of the member is RCV1. When defining the RCV1 in this example, the Full Queue Name would be set to EDIRECEIVE. Other fields would be set accordingly.
5. Within DataInterchange, you must define a Continuous Receive profile member. In this example, the name of the member is MQ1. Leave the **Requestor ID** and other **Selection** fields blank, as they are ignored. Follow the directions in “DataInterchange Processing After Data is Received” on page 5-25 to tell DataInterchange what processing is required once data is received from MQSeries.
6. Once all the previous steps have been completed successfully, DataInterchange automatically processes all data written to the MQSeries queue as soon as MQSeries dispatches the trigger event messages.

Continuous Receive Unique Selection Criteria

Based on the selection criteria, it is possible to set up one or more continuous receive members to receive and process EDI data and also set up another member to receive and process network acknowledgments, all with the same requestor ID. This allows you to send and receive data from the same mailbox and process network acknowledgments through continuous receive.

Note: When using MQSeries trigger queues, there is no real selection criteria via the DataInterchange Continuous Receive Facility. Rather, the continuous receive profile serves as a means to tell DataInterchange what to do with data once an MQSeries trigger event has occurred. So, if you are using MQSeries for continuous receive processing, skip to the next section.

You do not have to dedicate mailboxes for each direction. The same mailbox can be used in both directions. Each continuous receive profile member you have active is associated with unique continuous receive selection criteria. The active flag makes the profile member active. The following factors and corresponding profile fields make the selection criteria unique:

- REQUESTOR ID

The requestor ID indicates the requestor profile member from which you wish to receive data and that contains the Information Exchange account, user ID, and password information. You can have many continuous receive profile members with the same requestor ID, but each member must be unique. You can also have different requestor IDs in each continuous receive profile member. This makes each member unique without having to further qualify the continuous receive.

- TP NICKNAME

The trading partner nickname field indicates a trading partner from which you can receive data. When you complete this field, you must also supply the account and user ID fields for the associated trading

partner profile member. If you use this field, the continuous receive only receives and processes data for this specific trading partner. This field is optional.

- **MESSAGE USER CLASS**

The message user class is a code that you and your trading partners agree to use. If this field is supplied, only incoming data with the matching message user class is received and processed. This field is optional.

- **NETWORK ACKS ONLY**

By specifying a Y in this field, only network acknowledgments are received and processed.

DataInterchange issues a continuous receive with the sender account of *SYSTEM*, a user ID of *ERRMSG*, and a data type of A. This tells Expedite/CICS that only network acknowledgments and no EDI data is to be received. This field is optional and defaults to N.

DataInterchange Processing After Data is Received

The previous section describes how to dictate the selection criteria for each continuous receive. Once the data is received, DataInterchange must be told what processing should be performed. This is controlled by the following fields in the profile. They are listed below in order of precedence:

Field	Description
NETWORK ACKS ONLY	If the value of this field is Y, the data received is always a network acknowledgment and is processed as such. The actual processing is a CICS START and is issued for transaction EDIB (the DataInterchange Utility). The PERFORM PROCESS NETWORK ACKS command is given to the DataInterchange Utility along with the acknowledgment. If a response program is supplied in the continuous receive profile, it is given control and the name of the TS queue containing the acknowledgment is passed. Once control is returned to the DataInterchange Utility, the TS queue is deleted. A value of Y in this field overrides a value of Y in the TRANSLATE field.
TRANSLATE	If the value of this field is Y, the interchange received by Expedite/CICS and given to DataInterchange is deenveloped and translated to the application format. This process could also generate functional acknowledgments because they are created during the deenvelope process. This processing is performed within a separate CICS transaction, EDIB (the DataInterchange Utility). The PERFORM DEENVELOPE AND TRANSLATE command and the TS queue holding the interchange are passed. A value of Y in this field overrides a value of Y in the DEENVELOPE ONLY field.
DEENVELOPE ONLY	If the value of this field is Y, the interchange received by Expedite/CICS and given to DataInterchange are only deenveloped. This process could also generate functional acknowledgments because they are created during the deenvelope process. The processing is performed within a separate transaction, EDIB (the DataInterchange Utility). The command PERFORM DEENVELOPE and the TS queue holding the interchange are passed.

If you leave these fields blank, or set them to N, DataInterchange invokes only the continuous receive response program. This processing executes within the IMR1 transaction, and a separate EDIB transaction is not initiated.

Effect of Defining the EDI1 TD Queue

Use of the EDI1 TD queue is recommended, but not mandatory. If the TD queue is defined as an intra-partition TD queue, DataInterchange uses this queue to pass information from transaction IMR1 to transaction EDIB. If the queue is defined, the IMR1 transaction waits for the EDIB transaction to complete. Having IMR1 wait for EDIB to complete closes a recovery exposure and ensures all interchanges received by Expedite/CICS are processed by DataInterchange. If EDI1 is not defined, a short window exists where the interchange image is not contained in a recoverable resource. In the event of a failure (such as a power outage), interchanges could be lost. By defining EDI1, you eliminate the possibility of lost interchanges, Expedite/CICS and DataInterchange can recover appropriately. Using EDI1 is highly recommended to ensure no data is lost during continuous receive processing. It is necessary to define EDI1 as non-recoverable.

The EDI1 TD queue may also be used with DataInterchange's non-Expedite/CICS continuous receive interface.

Sent to Network Status

When Expedite/CICS Version 4.1 (or higher) is used and the IINCICS network profile communication routine name is 'VANEXPV4', transaction statuses will be updated from 'Send requested' to 'Sent to network' after the transactions have successfully been sent by Expedite/CICS. If, for some reason, a transaction was not sent successfully, its status would be changed from 'Send requested' to 'Not sent - network error'. DataInterchange and Expedite/CICS typically work in an asynchronous fashion in sending interchanges. At the time DataInterchange requests a send, Expedite/CICS will acknowledge receipt of the message but not necessarily perform the send. When the message has actually been sent to the network, Expedite/CICS (transaction IST1) will LINK back to DataInterchange so that the transaction status can be updated appropriately.

It is also possible for DataInterchange and Expedite/CICS to work synchronously in sending interchanges. The selection to work synchronously or asynchronously is established using the Expedite/CICS facility, transaction LGO1. The VANEXPV4 will update the status whether DataInterchange is working synchronously or asynchronously with Expedite/CICS. During the VANEXPV4 status update process, the:

- DataInterchange Transaction Store statuses are updated.
- Unique ID assigned by the network is inserted into the DataInterchange database and will be used as an index for processing subsequent network acknowledgments. This will improve performance for the network acknowledgment processing.

| **Note:** DataInterchange supports compress and delivery priority for Expedite/CICS.

Continuous Receive Without Expedite/CICS

Thus far, the Continuous Receive Facility has been described as it works with Expedite/CICS and Information Exchange only. You can also take advantage of the continuous receive profile for use with your own communications routines. DataInterchange provides a programming interface with which you can develop applications to interact with DataInterchange in a continuous receive mode. See "Continuous Receive Interface (CICS only)" on page 6-19 for details.

It is possible to process multiple incoming TS queues when continuous receive is used without Expedite/CICS. For more information, see "Processing Multiple Incoming TS Queues" on page 5-4. There are no special considerations, other than to make sure the TS queues are named in the correct fields within the commarea passed to EDICRIN. For more information, see "Invoking the Continuous Receive Interface" on page 6-19.

Response Applications

Response applications are user-written CICS programs or transactions that DataInterchange invokes at certain points of processing. If you are designing your application to run asynchronously with the DataInterchange Utility, or you are using the continuous receive facility, response applications are an essential element to the entire processing structure.

Methods of Invocation and Obtaining Results from DataInterchange

You can use the following methods to invoke your application:

- Specify a response type of PG to cause DataInterchange to CICS LINK to your program. When the CICS LINK command is issued, the COMMAREA and LENGTH keywords are used to pass the results. Your application then obtains the COMMAREA address and processes the results.

Note: Response programs of type PG should not handle abends. DataInterchange needs to handle all abends so that it is allowed to free ENQs and other resources. If DataInterchange is not allowed to handle abends, subsequent user tasks may hang.

- Specify a response type of TX to cause DataInterchange to CICS START your transaction. When the CICS START command is issued, the FROM and LENGTH keywords are used to pass results. Your application then obtains the results by issuing a CICS RETRIEVE.

See “Format of DataInterchange Utility Control Information” on page 5-18 for the format of the results control information given to your application.

Kinds of Response Applications

The following are the basic kinds of optional response applications that DataInterchange can invoke:

- DataInterchange Utility response application. The response indicator is U (see Table 5-4 on page 5-20 - Response Indicator field).
- Continuous receive response application. The response indicator is C.
- Transaction level response application. The response indicator is T.

It is important to be aware of the point at which your response application gains control. It is also important to know when to enter the response program name and type during product administration, because some choices might allow for greater flexibility. Response applications can be programs (type=PG) or CICS transactions (type=TX).

DataInterchange Utility Response Application - Indicator U

The DataInterchange Utility response application is only applicable when your main program invokes the DataInterchange Utility. For example, you can CICS START transaction EDIB and pass control information as specified in Table 5-3 on page 5-19. The response name and response type (PG or TX) passed in this control information are used to identify this general purpose utility response application.

It gains control after the DataInterchange Utility completes all of the processing you requested when you started EDIB. It is important to note that this DataInterchange Utility response application does not gain control at intermediate points during processing; it only gains control after all requested processing is complete.

This is an optional response application. It might not be necessary, depending on how DataInterchange was originally invoked. In the previous example, your application started transaction EDIB, and

DataInterchange processing was asynchronous with your application. It is important that you know the outcome of the function you requested. This is why you specify a DataInterchange Utility response application. The results DataInterchange passes to the DataInterchange Utility response application contain essential information that can be used for self-identification and to determine the success or failure of the requested function. See “Format of DataInterchange Utility Control Information” on page 5-18 for the format of the control information passed to the response application.

DataInterchange sets the value of the response indicator (RESPFLAG) field to specify the kind of response application being invoked. This information is also useful in determining why your application is being invoked. The user field (USRFLD) can also be used to identify the application that originally invoked DataInterchange. The user field (USRFLD) contains the same value that was supplied when the DataInterchange Utility was initiated, unless it has been modified by translation response applications. For more information, see “Transaction Response Application - Indicator T” on page 5-29.

The DataInterchange Utility response application responds to and inspects the severity code (CCBRC) and DataInterchange Utility condition code (CCBERC) fields to determine success or failure. These fields reflect the highest value encountered by DataInterchange during the entire process. Based on the outcome, your DataInterchange Utility response application might execute another internal program, or execute DataInterchange to perform any subsequent function.

A DataInterchange Utility response application can also manage your resources. You can clear the input file if the processing was successful. The DataInterchange Utility response application could complete the unit of work. If the application you used to invoke the DataInterchange Utility performs a CICS START EDIB and tells the DataInterchange Utility not to issue a CICS SYNCPOINT (SYNCVAL of -1), your DataInterchange Utility response application can update your recoverable resources and issue the CICS SYNCPOINT. This process includes all DataInterchange Utility and response application updates as part of the same unit of work, as long as your response application is a program (type PG). This scenario removes the DataInterchange Utility from your initiating application’s unit of work, but the response application you develop is still in charge of the DataInterchange Utility’s unit of work.

Continuous Receive Response Application - Indicator C

EDI interchanges and network acknowledgments are dynamically received on behalf of a specific continuous receive profile member based on the receive criteria specified therein. The name and type (PG or TX) of your continuous receive response application is specified in the continuous receive profile. For example, for EDI data, your continuous receive response program can be associated with specific interchanges that meet the criteria. Your continuous receive response application gains control after DataInterchange has completely processed the entire received envelope TS queue, and DataInterchange passes the entire envelope TS queue. Normally, the received envelope TS queue will contain only one interchange. The response application receives the resulting information from the DataInterchange Utility control information. See “Format of DataInterchange Utility Control Information” on page 5-18 for a description of the resulting information.

The response indicator (RESPFLAG) and user area (USRFLD) fields are useful in determining why your application is being invoked. DataInterchange sets the value of the response indicator field to indicate the kind of response application being invoked. The user area field contains the value specified in the USER FIELD of the continuous receive profile member, unless it has been modified by translation response applications. For more information, see “Transaction Response Application - Indicator T” on page 5-29.

The envelope TS queue name is passed in the control area. It is the application’s responsibility to further process the envelope TS queue (delete the TS queue, archive, and so on). The severity code (CCBRC) and DataInterchange Utility condition code (CCBERC) fields should be inspected to determine success or failure of the overall continuous receive process. If a transaction level response program is used, it may not be necessary to take action in this continuous receive response application for data element or

segment-level translation errors. The errors can be handled in the transaction response application. It is the responsibility of this continuous receive response application to detect higher severity errors, such as communications errors, database errors, abnormal ending, and so on.

For network acknowledgments (Process Network Acks=Y in the continuous receive profile), your continuous receive response application gains control after DataInterchange has processed the network acknowledgment and updated the status in the Transaction Store. The name of the TS queue containing the acknowledgment is in the FILEID field. A continuous receive response application is optional when receiving and processing network acknowledgments. The only reason to use a continuous receive response application is if you want to save the network acknowledgment itself. The TS queue holding the network acknowledgment is deleted before termination. If you have no reason to save the network acknowledgment, you do not need to develop a continuous receive response application.

If the continuous receive response application is a program (type PG), this is a good place to issue a CICS SYNCPOINT. Consider this option, especially if you set the Allow syncpoints field in the continuous receive profile to N. By doing this, your continuous receive response program gains control and the unit of work is still active. At this point, you can change your recoverable resources and issue the CICS SYNCPOINT command. The unit of work would then include updates made to the DataInterchange Utility and your application changes to recoverable resources.

If more granularity is needed, especially where you need control after each transaction is processed, see “Transaction Response Application - Indicator T.” You can use transaction level response programs in conjunction with continuous receive processing, assuming the translate value is Y in the continuous receive profile.

In summary, the continuous receive profile provides information that controls what is received and how it is subsequently processed.

Transaction Response Application - Indicator T

You can specify that control should pass to the transaction response application for each individual transaction that is translated to the application format. Enter the name of the response program in the Application file name field, and enter the type (PG or TX) in the Application file type field (on the Trading partner usage override for receiving panel (TP27), or the Add Data Format (TD02), Copy Data Format (TD03), and Update Data Format (TD04) panels. Details about where to specify the transaction response application are discussed later in this section.

The use of this kind of response application is important if you are implementing an event-driven EDI system. Control is passed to the response application when one of the following occurs:

- The continuous receive profile member's Translate value is set to Y, and a translation to the application format has occurred.
- One of the following DataInterchange Utility commands is invoked:
 - Perform Receive and Translate
 - Perform Deenvelope and Translate
 - Perform Translate to Application
 - Perform Retranslate to Application
- After the TRANSLATE RECEIVED TRANSACTIONS or RE-TRANSLATE RECEIVED TRANSACTIONS option is invoked from the online CICS Transaction Store facility.

The point at which transaction level response programs are invoked is based on the recovery level (DataInterchange Utility parameter RECOVERY). If the recovery level is E (envelope), then all transactions within the interchange are deenveloped and translated before the first response application is

invoked. Once the entire interchange is deenveloped and translated, all corresponding response programs are invoked, one after the other, for each transaction.

A syncpoint is taken by DataInterchange after all response programs are invoked. The entire interchange and response program invocation is one unit of work. This is the default in CICS and occurs during continuous receive processing. It is also the default when deenveloping is part of the translation process.

Processing is different when transactions are translated separately from the deenvelope process. Here, the following processes are performed:

- A transaction is translated
- The response program is invoked
- A syncpoint is taken

These steps are repeated for each transaction being translated.

The results are passed to the response application within the DataInterchange Utility control information. The name of a unique temporary storage queue that holds the translated application data and assigned by DataInterchange is passed within this control information. The field name is the APPFILE (the application data TS queue). The name of the TS queue is 8 characters long and begins with EDI. For more information, see “Format of DataInterchange Utility Control Information” on page 5-18.

The response indicator (RESPFLAG) and user area (USRFLD) fields are useful in determining why your application is being invoked. DataInterchange sets the value of the response indicator field to specify the kind of response application being invoked. When the transaction response application is invoked, the user area will contain:

- The value specified in the USER FIELD of the continuous receive profile member if translation is occurring because of a continuous receive request.
- The original user area value provided by the application that originally invoked the DataInterchange Utility.
- Blanks if the DataInterchange Utility is being invoked because a TRANSLATE RECEIVED TRANSACTIONS or RE-TRANSLATE RECEIVED TRANSACTIONS option was requested from the online CICS Transaction Store facility.
- The user area modified by a previous invocation of a translation response application. If a translation response application modifies the user area, the next invocation of a response application will receive the updated user area value, including utility response applications and continuous receive response applications.

The modified value of the user area is reset whenever the DataInterchange Utility is invoked and for each EDI interchange processed by a continuous receive request.

The updated control information also contains the severity code (CCBRC) and DataInterchange Utility condition code (CCBERC) fields. These fields should be inspected to determine success or failure of the translation. See *DataInterchange Messages and Codes* for more information. Your transaction response application gains control any time the transaction is translated, independent of the acceptable error level defined in the trading partner transaction receive usage. However, if the error is unacceptable, the APPFILE field is filled in with a TS queue name, but the queue is empty. If you attempt to issue a CICS READQ command against this TS queue, you will receive a CICS QIDERR response. DataInterchange passes in the transaction handle in the results.

If an unacceptable translation error occurs, this response application can specifically identify the transaction in error. This can be saved for the next error-handling processes. This handle is also passed by DataInterchange if no errors occur, and can be used as you choose.

Note: Transaction level response applications should not issue the CICS SYNCPOINT command if the DataInterchange Utility is running with the SYNCVAL set to -1, because this increases the potential for deadlock.

Where to Specify the Transaction Response Application

The name of this response program is entered in the Application file name field, and its type (PG or TX) is entered in the Application file type field on the Trading Partner Usage Override for Receiving panel (TP27) or the Add Data Format (TD02), Copy Data Format (TD03), and Update Data Format panels (TD04).

- | The application file type field serves one of two purposes that are mutually exclusive. If the application file type is MQ, TS, TM, TD, or VS, then no transaction level response program is invoked. Translation still takes place for each transaction, but rather than passing control to a response program, the translated data is simply appended to the application file. The transaction response application is only invoked for a transaction when the application file type identifies a program or transaction (PG or TX).
- | A transaction response program is assumed if the type is PG or TX. It might be important to note whether the file name and types fields are taken from the trading partner receive usage or the application data format. DataInterchange uses the trading partner receive usage overrides first, if they are present, then it uses the application data format fields. Specifying a transaction response program in the trading partner receive usage allows you to identify different programs based on the trading partner that sent them. If there is no need for this granularity, the application data format is sufficient to identify a transaction response application based on the application that needs to process the data.

The following are the conditions under which your response program is invoked:

- If the translator extended return code is 2 or less, your response program is invoked. If the translation is acceptable, the application file (APPPFILE) field contains the name of the TS queue containing the translated data and the THANDLE field is filled in. If the translation is unacceptable, only the THANDLE field data is valid.
- If the translator extended return code is equal to 3 and a map exists for this transaction, your program is invoked and the THANDLE field is filled in.
- If the translator extended return code is equal to 3 and a map does not exist for this transaction, your transaction level response program is not invoked.
- If the translator extended return code is greater than 3, your transaction level response program is not invoked.

If your transaction level response program is not invoked, error handling for this level of error needs to be managed in a higher-level response program, such as the continuous receive or the DataInterchange Utility termination response program, whichever is applicable.

Persistent Environment

The DataInterchange persistent environment is an optional feature that can be used with CICS/ESA systems to improve DataInterchange performance. It accomplishes this by reducing the amount of read operations needed to perform translation and other functions. As data is read from the DataInterchange database, some of it is saved in an MVS data space. Next time the data is needed, it is obtained from the data space instead of the database. The data space remains functional during the life of a DataInterchange session. A DataInterchange session is considered to start when the first DataInterchange function is requested in a CICS region and ends when DataInterchange transaction EDIT is terminated.

This process allows many DataInterchange transactions running concurrently and over a long period of time to obtain the same piece of data with only one access to the DataInterchange database.

Data saved into the data space includes control strings, receive usages, send usages, and trading partner transactions (maps).

MVS Subtasks

Though there is only one DataInterchange data space per CICS region, DataInterchange will start one to sixteen MVS subtasks to manage the DataInterchange data space. The more of these subtasks running, the more throughput the data space can deliver. This value used to determine how many of these subtasks will be started is obtained from the SYSPROF profile for the CICS region. You may tune the number of subtasks up or down to fit your requirements. The number of subtasks defined in the SYSPROF profile should not exceed the number of DataInterchange threads that may execute concurrently on the CICS region.

The Data Space

The size of the MVS data space is determined by a couple of factors. First is the maximum size specified in the SYSPROF profile for the CICS region. The data space will not be made larger than the value specified in the SYSPROF profile. The value in the profile cannot exceed the installation maximum. Unless altered by the installation, this value will be the IBM default of 239 4K byte blocks (less than 1MB). Your installation can use the installation exit IEFUSI to change the IBM default.

Secondly, DataInterchange learns based on the past and can define a data space smaller than the maximum. The format of the data space is tuned automatically based on past usage to provide the most efficient data space structure. An effort is made to structure the data space at initialization so that no restructuring will need to occur during regular DataInterchange processing.

However, depending on installation usage and maximum size of the data space, all or parts of the data space may become full during processing. When this occurs, DataInterchange will expand the data space (if defined smaller than the maximum), and/or can reorganize the structure of the data space. If necessary, older data in the data space will be removed. Ideally, you do not want any of these situations to occur since making these adjustments results in momentary delays in DataInterchange processing. DataInterchange will recognize these conditions and try to adjust for them in the next session. If these conditions persist after a few sessions, it may be desirable to increase the maximum data space size. You can monitor the DataInterchange log file for messages indicating that these scenarios have occurred. All message identifiers relating to the persistent environment begin with the qualifier "GB".

It is important to ensure the DataInterchange session is ended normally when using the persistent environment. A DataInterchange session is considered ended when transaction EDIT is removed from the system. Refer to transaction EDIT on page 5-43 for information on removing transaction EDIT from the system.

Enablement/Disablement

To enable the persistent environment, you must be running DataInterchange in a CICS/ESA region and the SYSPROF profile member for the region must indicate that you want the persistent environment established.

To disable the persistent environment, use the SYSPROF profile to indicate that you want the persistent environment disabled.

Multiple Regions

- | Certain installations may allocate a DataInterchange database for use by a single DataInterchange region.
- | In these cases, when performing online updates to database records that are kept in the DataInterchange Persistent Environment, the old copy of the record will be removed from the data space.

Transaction EDIG has been created to address installations that have multiple DataInterchange regions/environments sharing a single DataInterchange database. EDIG is a transaction that is automatically started periodically. It looks to see if an update has been made recently that may affect the persistent environment. If an update has been made, then all records similar to the updated record will be removed from the persistent environment; that is, if a control string is deleted, then all control strings will be removed from the persistent environment.

It is important in installations where multiple regions/environments share a single DataInterchange database that all online updates to the database are made from the same region/environment.

Note: CICS regions with DataInterchange VSAM files defined as data tables will not have updates reflected until the VSAM data tables are reloaded and the DataInterchange session is restarted.

Reserved Temporary Storage and Transient Data Queues

By convention, DataInterchange uses the three-character prefix **EDI** for its TS and TD queues. You should verify that no other application running in the same CICS region with DataInterchange uses this TS and TD queue naming convention.

TS Queues that Might Require Additional Processing

The first time DataInterchange initializes in a CICS region, it creates a TS queue named EDITV00. This TS queue holds frequently used translation and validation tables. See the *DataInterchange Administrator's Guide*.

If you modify any of the following tables using the DataInterchange administrative functions, you must purge EDITV00 before the change can take effect. The tables stored in EDITV00 are:

- Alphnum
- Charset
- Filename
- Foldchar
- Langprof
- Monocase
- Prgname
- Specnum
- Tfstatus
- Tptsnc
- Transyn

For every envelope standard defined to DataInterchange, there is a corresponding TS queue. For more information on envelope standards, see the *DataInterchange Administrator's Guide*. The following TS queues are used for envelope standards:

EDIE EDIFACT envelopes
EDII ICS Standard envelopes
EDIT UNTDI envelopes

EDIU UCS envelopes
EDIX X12 envelopes

Each envelope standard TS queue is created when the envelope standard is first used. If you modify an envelope standard, delete the corresponding TS queue so that the changes become effective.

The next envelope request causes the changes to be read into the TS queue. DataInterchange uses the revised standard until CICS system shutdown or until you repeat this process. This improves performance by cutting out the repeated reading of the envelope standards from DB2 or VSAM.

TD Queues Used By Export and Import

These TD queues are currently reserved by DataInterchange for use with the export and import functions:

EDIA	Application data formats
EDIB	Tables
EDIC	Control strings
EDID	Business document definitions
EDIM	Business document layouts
EDIP	Profiles
EDIS	Standards
EDIT	Trading partner transactions

For more information on export and import, see the *DataInterchange Administrator's Guide*.

TS Queues Used By Import

These TS queues are currently reserved by DataInterchange for use with the import function.

EDIATSx	Application data formats
EDIBTSx	Tables
EDICTSx	Control strings
EDIDTSx	Business document definitions
EDIMTSx	Business document layouts
EDIPTSx	Profiles
EDISTSx	Standards
EDITTSx	Trading partner transactions

Currently, the DataInterchange for CICS facility supports up to three TS queues of data per type on import. This is approximately 96 K records (or 32 K by 96 K bytes of data per type). Data to be imported must come into DataInterchange for CICS in TD queues. For more information, see "TD Queues Used By Export and Import" above. These TD queues can be defined as intra- or extra-partitioned. The DataInterchange for CICS import facility then copies the data from the TD queues into their corresponding TS queues. It is the data in the TS queues that is actually imported.

TD Queues EDI1, EDI2, and EDI3

DataInterchange reserves TD queues with the names EDI1, EDI2, and EDI3. See "Effect of Defining the EDI1 TD Queue" on page 5-26 for a complete description of the use of EDI1.

If TD queue EDI2 is defined as an intra-partition transient data queue, DataInterchange will use this queue as the Expedite/CICS administrative response file. Expedite/CICS writes out network acknowledgments to this file, and it is used as input by DataInterchange for updating network status. If EDI2 is not defined, a unique TS queue per Information Exchange mailbox is used as administrative response files. These TS queues are always appended to and never cleared. If EDI2 is defined, Expedite/CICS and

DataInterchange use it as the administrative response file for all Information Exchange mailboxes. The destructive read property of intra-partition TD queues causes the network acknowledgments to be deleted once they are processed by DataInterchange. If network acknowledgments do not have to be archived, then EDI2 should be used so that network acknowledgments are not reprocessed. Network acknowledgments are reprocessed when EDI2 is not defined and multiple update status requests are executed within an active CICS region.

Note: If network acknowledgments are being received and processed continuously through continuous receive, EDI2 is not used even if it is defined. In this case, Expedite/CICS passes DataInterchange one network acknowledgment at a time in unique TS queues. On completion, DataInterchange will delete the TS queue holding the network acknowledgment.

| In a DB2 environment, TD queue EDI3 must be defined as an intra-partition queue with trigger level one
| and associated with EDIE. Transaction EDIE (program EDIELAS) is responsible for all DB2 event log
| insertions, and is used to keep this activity out of the main commit scope. Repository module EDIELOG
| writes event log messages to EDI3, and EDIELAS picks them up from there and inserts them into the
| database. If an error occurs in EDIELAS, a message is written to the system console with message ID
| RS0002.

Interface Between DataInterchange for CICS, Expedite/CICS and Information Exchange

DataInterchange provides communication routines that interface to Expedite/CICS. EDI interchanges can be sent to and received from the Information Exchange network using this interface. Additional features include an event-driven continuous receive process and network acknowledgment reconciliation. Trading partner and requestor profiles should reference network profile IINCICS for this environment.

The communication flow involves these components:

- DataInterchange
- Expedite/CICS
- Information Exchange

All three components maintain control information to track activities, so it is essential to keep them synchronized. Expedite/CICS and Information Exchange are standalone products that can be operated independently from DataInterchange. Therefore, it is possible to execute operational commands without the participation of DataInterchange, thereby prohibiting DataInterchange from updating control information. You need to make judicious use of such commands to prevent out-of-sync conditions. There are two fundamental programming layers, or sessions, between the three components that need to be kept in sync: the low level Information exchange session on which all commands and data pass, and the higher level continuous receive session.

Information Exchange Session

The Information Exchange session, sometimes called an IE session, is a fundamental session between an application program and a particular Information Exchange user ID (mailbox). All commands and data sent out from, or received into, a mailbox must pass through this session. And you can have only one active IE session per mailbox. For example, if you are using Expedite Base/MVS, this means that you cannot have concurrent IE sessions for the same mailbox on two different Expedite/CICS regions, or one on CICS and another in batch. You can have any number of Information Exchange sessions from various environments if each session pertains to a different mailbox.

Most of the processing that occurs over the Information Exchange session is performed by Expedite/CICS and Information Exchange. DataInterchange is involved only at the application level, but it is no less

important that the session be initiated and maintained by DataInterchange to ensure proper execution of all DataInterchange network facilities. The first time DataInterchange receives a request from you to perform any network function for a given mailbox, it automatically issues the appropriate session start command to Expedite/CICS. Expedite/CICS then starts the session with Information Exchange and an access key is assigned. All three components now have a record of this particular Information Exchange session. After the Information Exchange session is started, DataInterchange performs the network function you requested and returns control to you.

For efficiency reasons, DataInterchange does not issue a session end after the requested network function is complete. So, the next time you request a network function for the same mailbox, DataInterchange will not issue another session start because it knows that an Information Exchange session is already active. There is no degradation in performance as a result of leaving an Information Exchange session active.

The Expedite/CICS product provides the necessary post-initialization PLT program, EXPOSTRT, that automatically re-enables any Information Exchange sessions that were active when the CICS region was brought down, even after an immediate shutdown. For more information on PLT processing, see “Program List Table Considerations” on page 5-41.

In theory, one Information Exchange session could be used indefinitely. However, note that for a given mailbox you can only have one active Information Exchange session, and until this Information Exchange session is ended, no other environment—not even a batch job using Expedite Base/MVS—should access this mailbox. If you have multi-environment communication needs with a particular Information Exchange mailbox, you need to end the IE session from CICS before executing communications in batch. The only time DataInterchange issues an Information Exchange session end request is when a PERFORM CLOSE MAILBOX is explicitly issued by your application. To keep DataInterchange, Expedite, and Information Exchange synchronized, you should use the DataInterchange PERFORM CLOSE MAILBOX function to end your Information Exchange sessions. If you will be using the continuous receive function of DataInterchange, executing a PERFORM CLOSE MAILBOX will implicitly end a continuous receive session. It is good practice to first end the continuous receive session and then execute a PERFORM CLOSE MAILBOX. See “Continuous Receive Session” on page 5-38 for details.

The Expedite/CICS product provides a facility to start and end Information Exchange sessions directly.

Warning: You should not use Expedite/CICS facilities to manage Information Exchange sessions used by the DataInterchange application.

The following is for your information and to prevent inadvertent execution of certain Expedite/CICS functions. The Expedite/CICS terminal transaction **LGO1** can be used to start and end Information Exchange sessions. After you enter your mailbox account and user ID, **LGO1** checks internal control information. If it finds that an Information Exchange session is already active for the mailbox you identified, it brings you to the main menu, assuming the Force User to log on user session option is *No* (in which case, it does not issue a Session Start to Information Exchange). If there is no active Information Exchange session, **LGO1** prompts you for an Information Exchange password, and if entered, Expedite issues an Information Exchange Session Start. Once the session starts, the main menu is displayed.

Accessing the mailbox from **LGO1** is one way to determine if Expedite acknowledges that there is an active Information Exchange session. You can end an Information Exchange session from **LGO1** by typing X to logoff. A confirmation panel is displayed; it will show any active network functions that must complete before the End will be issued. If one of the active functions is Continuous Receive, it will end this session implicitly and cause an out-of-sync condition between Expedite and DataInterchange. For more information, see “Continuous Receive Session” on page 5-38. If confirmed, Expedite issues an Information Exchange Session End to Information Exchange. (Expedite is unaware that DataInterchange originally started the session.)

DataInterchange is not given the opportunity to update its internal control information, which brings about an out-of-sync condition among the components. However, DataInterchange has built-in logic to re-issue a Session Start if Expedite returns the message HI421 session profile does not exist. Therefore, ending an Information Exchange session does not have serious implications, except when a Continuous Receive session is ended implicitly. If you find you inadvertently logged off and you had no Continuous Receives active, do not log back on to **LGO1** to restart the session, because DataInterchange must initiate Information Exchange sessions.

DataInterchange will automatically restart a session when you perform your next network function. As a general rule, if you enter **LGO1** for whatever reason, leave the session settings alone. If you were prompted to enter your password upon entry, log off when you leave. If you were not prompted for a password, do not log off when you leave (use PF3 or type End). For more information on using **LGO1**, refer to *Using Expedite/CICS Display Application*.

You can sign on to Information Exchange directly through a terminal connection to the IBM Global Network. The Information Exchange/SERV option, Information Exchange Administration Services, offers many useful tools and functions for managing your network activity. For example, you can directly reset a mailbox's Information Exchange session. However, you should not regularly use the Information Exchange/SERV facility to reset Information Exchange sessions used by the DataInterchange application. It may be useful in recovery situations, but should be used carefully. Resetting an Information Exchange session from Information Exchange/SERV ends the session from the Information Exchange perspective only. IE is unaware that the session was originally started by DataInterchange through Expedite/CICS. It causes an out-of-sync condition between the components. For more information regarding Information Exchange/SERV, see *Using Information Exchange Administration Services*.

Information Exchange Session Cleanup

An Information Exchange session problem may prohibit the starting or ending of an Information Exchange session. The most common symptom of an Information Exchange session error is the Expedite/CICS error SDIERR RESPONSE CODE 000008. If the error is encountered by DataInterchange, it will be embedded in a logged DataInterchange error message.

Information Exchange session problems are most often caused by a user or another program having reset the Information Exchange session. As discussed earlier, it is possible to end an Information Exchange session directly using Information Exchange/SERV. Possibly another CICS region or another application in a different environment started a session with the same mailbox. In any event, manual intervention is required. A recovery procedure follows:

- Step 1. Sign on to Information Exchange/SERV. Choose Option 1 from the main menu, Work with Profiles. Fill in the appropriate account and user ID for the mailbox in question and choose Option 8, Reset a User's Session. Does message *User acct.user ID does not have an active session* appear at the bottom of the screen?
- If the message appears, this means that your Information Exchange session was previously reset by another person or application. When the application reset your session, it ended its own. Cancel the reset request and go to Step 2 on page 5-38.
 - If the message does not appear, an Information Exchange session does exist for your mailbox. This may mean that whatever reset your session did not end its own. Enter Y to confirm the reset, then go to Step 2 on page 5-38.

Note: If you find you need to perform this procedure frequently, you may want to inspect the session trace available on Information Exchange/SERV to identify the LU name that is resetting your session.

- Step 2. Use the Expedite/CICS IDLT CICS terminal transaction to delete the Information Exchange session control information from the Expedite/CICS control file. The format of the IDLT transaction follows:

IDLTacct user ID

The account must be 8 bytes long, left-justified and concatenated to the IDLT transaction ID. If your account is shorter than 8 characters, pad the remaining positions with spaces.

Now enter your user ID. No validation is performed on this field, so it is important to key it in correctly. Go to Step 3.

- Step 3. If your Information Exchange session had any active continuous receive sessions, go to the next section, "Continuous Receive Session," or simply try the failed network function again. As discussed earlier, DataInterchange will automatically re-issue an Information Exchange Session Start.

Continuous Receive Session

The continuous receive (CR) session is a high-level application programming layer that is added on top of an Information Exchange session. A continuous receive session cannot run without an Information Exchange session. An overview of the continuous receive process follows.

When a CR session is active, EDI messages entering the mailbox that match the continuous receive criteria are automatically received. Information Exchange automatically starts Expedite/CICS and passes the EDI envelope. Expedite/CICS then writes the envelope to a unique TS queue and links to DataInterchange passing the name of this TS queue. DataInterchange processes the envelope using the processing options specified in the applicable continuous receive profile, after which DataInterchange returns control to Expedite/CICS. Expedite/CICS is then free to process the next envelope for this continuous receive session. Exactly when DataInterchange returns control to Expedite/CICS depends on whether intra-partitioned TD queue EDI1 is defined. If EDI1 is defined, DataInterchange and Expedite run synchronously. In other words, Expedite/CICS will not process the next envelope until DataInterchange returns control. Without the EDI1 TD queue, Expedite/CICS and DataInterchange run asynchronously where Expedite is free to process the next envelope before DataInterchange has finished with the first. See "Effect of Defining the EDI1 TD Queue" on page 5-26. Multiple continuous receive sessions with unique receive criteria can be concurrently active for one mailbox, in which case, all will use the same Information Exchange session. You can also have continuous receive sessions for other mailboxes.

Starting and Stopping Continuous Receive Sessions

The Continuous Receive Facility of DataInterchange provides two CICS terminal transactions: **EDIR** to start a continuous receive session, and **EDIS** to stop a continuous receive session. These two CICS transactions are complimented by the DataInterchange Utility commands PERFORM START CONTINUOUS RECEIVE and PERFORM STOP CONTINUOUS RECEIVE. Also, the DataInterchange Utility provides a way to report the statuses of your continuous receives by using the PERFORM REPORT CONTINUOUS RECEIVE STATUS command. For more information on these commands, see "Continuous Receive Functions" on page 1-65.

Note: These transactions pertain to continuous receive processing associated with Expedite/CICS and Information Exchange. Continuous receive processing driven by user-written applications is not started or stopped. See "Continuous Receive Interface (CICS only)" on page 6-19 for details.

Both transactions can apply to all continuous receive profile members or selectively to a single member. The following shows the format for entering the transaction at a terminal:

EDIR [*membername*]
EDIS [*membername*]

where *membername* identifies a member of the continuous receive profile. If *membername* is not specified, all continuous receive members are assumed, and a continuous receive start or stop occurs for each profile member.

During a continuous receive Session Start using **EDIR**, DataInterchange invokes Expedite/CICS passing the continuous receive unique selection criteria. See “Continuous Receive Unique Selection Criteria” on page 5-24 for the criteria options. Expedite/CICS assigns a key to identify the CR session, passes the criteria and key to Information Exchange, and returns control to DataInterchange. DataInterchange then sends a message back to the issuing terminal indicating success or failure of the CR session start. After a successful start, all three components (DataInterchange, Expedite/CICS, and Information Exchange) have saved the key assigned to the continuous receive session for subsequent identification. If an error occurs invoking **EDIR**, inspect the EDIFFS event log.

During a continuous receive Session Stop using **EDIS**, DataInterchange invokes Expedite/CICS passing the key for the particular continuous receive session. Expedite/CICS issues the end continuous receive request to Information Exchange. Yet, the continuous receive may not be stopped at this point. The end request is delayed if an actual receive is taking place. Expedite/CICS returns control to DataInterchange where DataInterchange and Expedite/CICS now complete asynchronously. However, DataInterchange checks Expedite/CICS control information until an indicator shows that the continuous receive session has ended. After which, DataInterchange will issue a message confirming a successful end.

The Expedite/CICS product provides a facility to start and stop continuous receive sessions directly. It is not recommended that you use Expedite/CICS facilities to regularly manage your CR sessions used by the DataInterchange application. The following is for your information and to prevent inadvertent execution of certain functions within Expedite/CICS. The Expedite/CICS terminal transaction, LGO1, can be used to start and stop any active continuous receive sessions. It is available under Option 1, Work with Receive Data. DataInterchange will not be aware of any activities performed in LGO1, so use this feature carefully.

Continuous Receive Session Cleanup

Before attempting any cleanup, you should generate a continuous receive status report. For more information, see “Reporting Continuous Receive Statuses” on page 1-66. Based on the reported status, you can initiate a reasonable cleanup.

DataInterchange supplies a CICS terminal transaction, **EDIZ**, that can be used to clean up an unrecoverable continuous receive session. It deletes internal continuous receive session control records; however, only those managed by DataInterchange. It does not affect the state of Expedite/CICS or Information Exchange. Therefore, it is essential that Expedite/CICS (using LGO1) be used to terminate all continuous receive sessions before executing **EDIZ**. If Expedite/CICS has not terminated the continuous receive and **EDIZ** is executed inadvertently, EDI envelopes will be lost because DataInterchange is unaware of the receive requests for which Expedite/CICS is now passing data.

The format of the command is similar to **EDIR** and **EDIS** described above:

EDIZ [*membername*]

if *membername* is not specified, all continuous receive members are assumed.

The following is a list of symptoms to help you identify an unrecoverable continuous receive session.

- A continuous receive session that was once working is no longer receiving data. Most likely, the Information Exchange session was lost after a continuous receive had already been successfully started.
- A continuous receive session is receiving data from the mailbox, but DataInterchange is not processing the data. It appears DataInterchange is never getting control. This is probably caused by an inadvertent or premature use of **EDIZ**.
- When **EDIS** is executed, DataInterchange will check the stop indicator for up to 2 minutes. If more than 2 minutes elapse, a timeout error is logged, VN1019, and a message is issued to the terminal. This may be caused by a delay in Expedite/CICS to end a continuous receive session.
- **EDIR** or **EDIS** will fail immediately if Expedite/CICS cannot communicate with Information Exchange because of an Information Exchange session problem. A negative response is returned and DataInterchange logs an error containing the text that an SDIERR occurred and sends a message to the terminal. This is an indication that you have an Information Exchange session that requires recovery.

If any of the above symptoms arise, use the following Continuous Receive Recovery Procedure:

Step 1. Sign on to the mailbox using LGO1.

- a. If you are not prompted to enter your password, go to Step 2.
- b. If you are prompted to enter your password, this means that an Information Exchange session does not already exist. Enter your password to access the LGO1 main menu. Expedite/CICS will start a new Information Exchange session, which may re-activate continuous receive(s) that were not receiving data previously. This is expected. Continue with Step 2.

Step 2. Choose Option 1, Work with Receive Data, and then choose Option 3, Stop Continuous Receive. If there are no entries displayed, go to Step 6. If there are entries, go to Step 3.

Step 3. Issue a stop request by all continuous receive sessions. Use the Enter key to refresh the screen.

- a. If they change from STARTED to STOPPED, go to Step 6.
- b. If they do not change to STOPPED, go to Step 4.

Step 4. Sign on to Information Exchange/SERV. Choose Option 1, Work with Profiles. Fill in the appropriate account and user ID for the mailbox in question and choose Option 8, Reset a User's Session. Enter Y to confirm the reset and go to Step 5.

Step 5. Use the Expedite/CICS IDLT CICS terminal transaction to delete the Information Exchange session control information from the Expedite/CICS control file. The format of the IDLT transaction follows:

```
IDLTACCT [user ID]
```

The account must be 8 bytes left-justified and concatenated to the IDLT transaction ID. If your account is shorter than 8, pad the rest with spaces. Follow this with the user ID. There is no validation performed on the format, so it is important to key it in correctly. Go back to Step 1 to attempt a continuous receive stop again.

Step 6. You should still be in LGO1 at this point. Type =X to log off. Confirm the session end, and go to Step 7 on page 5-41.

Step 7. Execute the DataInterchange Continuous receive cleanup transaction, **EDIZ**, only for the continuous receive profile member(s) that are experiencing a symptom, and go to Step 8.

Step 8. Restart the continuous receive session using **EDIR**.

The continuous receive session should be recovered at this point. If you executed **EDIZ** inadvertently or prematurely, envelopes will probably be lost. You can use LGO1 to determine which, if any, envelopes were not processed by DataInterchange. Choose the Receive option from the main menu, and then Option 5, View List of Completed Receives. Each EDI envelope received by Expedite/CICS will have an entry. A status of RECEIVED is normal. A status of E-HI999 means that DataInterchange did not accept the envelope. DataInterchange returns the HI999 error when a continuous receive control record is not found for a receive. Envelopes that encounter the HI999 error cannot be reprocessed through the Release option on this LGO1 panel, because DataInterchange and Expedite/CICS cannot be re-synchronized. However, you can use the View option here to obtain the interchange control number, the interchange sender, and so on. If your mailbox has archiving turned on, you can use IE/SERV to retrieve each lost envelope. An envelope retrieved from archive is receivable as if it had just been sent.

Note: When an HI999 error is returned, DataInterchange issues an error message to the EXPLOG1 data set associated with your CICS region; CR0120 DataInterchange control record missing for received data, writing data to EXPDERR file.

Program List Table Considerations

There are two post-initialization PLT programs that affect continuous receive processing. One that all customers should use is a program called EXPOSTRT. This is an Expedite/CICS PLT program that re-establishes both Information Exchange and continuous receive sessions automatically during CICS start-up, even after an immediate shutdown. The second program is an optional DataInterchange PLT program called, EDICRTS. In effect, this program issues **EDIR** to start all continuous receive profile members that have their active flag set to Y. For most customers, EDICRTS is unnecessary because EXPOSTRT will re-establish existing CR sessions for you. EDICRTS starts entirely new sessions, which is redundant if your continuous receive sessions were already active when the region was brought down.

There is one optional pre-termination PLT program provided by DataInterchange named, EDICRSP. In effect, this program issues an **EDIS** to stop all continuous receives. In addition to the **EDIS** function, this program issues a PERFORM CLOSE MAILBOX against all involved mailboxes. As you can see, the pre-termination program EDICRSP and post-initialization program EDICRTS go hand in hand. During a normal shutdown, EDICRSP will end continuous receive sessions and end all Information Exchange sessions. The purpose of this is to free up all mailboxes using continuous receive so that other communication applications requiring the same mailbox can execute while the CICS region is down. This will prevent the Information Exchange session problems discussed earlier. However, if you have no other Information Exchange communication applications, or if the other applications you have do not use the same mailbox, it is unnecessary to free up the Information Exchange session and PLT program EDICRSP is unnecessary.

If EDICRSP is used for pre-termination, then you should use EDICRTS for post-initialization to start those continuous receive sessions ended by EDICRSP. EXPOSTRT will not start these for you in this case, because they will not be active after shutdown is complete.

The following are sample post-initialization and pre-termination PLTs:

```
*-----*
* Sample post-initialization PLT including program EDICRTS,      *
* the continuous receive start program, and the necessary        *
* Expedite/CICS PLT program EXPOSTRT. Also included is a       *
* sample entry for the DB2 attachment facility, DSNCCOM1, in    *
```

```
* case you are using DB2 for your repository. The order of
* the following programs is important.
```

```
*-----*
```

```
PLTAA    DFHPLT TYPE=INITIAL,SUFFIX=AA
         DFHPLT TYPE=ENTRY,PROGRAM=DSNCCOM1
         DFHPLT TYPE=ENTRY,PROGRAM=EXPOSTRT
         DFHPLT TYPE=ENTRY,PROGRAM=EDICRTS
         DFHPLT TYPE=FINAL
         END
```

```
*-----*
```

```
* Sample pre-termination PLT including program EDICRSP, the
* continuous receive stop program. Program EDIXSOX, which
* terminates the long running DataInterchange transaction, EDIT,
* is shown along with DB2 attachment program, DSNCCOM1, to
* provide the proper order for the entires.
```

```
*-----*
```

```
PLTZZ    DFHPLT TYPE=INITIAL,SUFFIX=ZZ
```

```
         DFHPLT TYPE=ENTRY,PROGRAM=EDICRSP
         DFHPLT TYPE=ENTRY,PROGRAM=EDIXSOX
         DFHPLT TYPE=ENTRY,PROGRAM=DSNCCOM1
         DFHPLT TYPE=FINAL
         END
```

Note: If you will be using the EDICRSP PLT program, it is necessary to add an XLT entry for Expedite/CICS transaction ISC2. A sample entry follows:

```
*-----*
```

```
*                               DataInterchange for CICS
```

```
*-----*
```

```
* The following XLT (CICS Transaction List Table) entries are to be
* applied to any CICS region where DataInterchange will execute and
* where program EDICRSP is included in the pre-termination PLT.
```

```
*-----*
```

```
*-----*
```

```
XLTA    DFHXLT TYPE=INITIAL,SUFFIX=AA
        DFHXLT TYPE=ENTRY,TRANSID=ISC2
        DFHXLT TYPE=FINAL
        END
```

Processing Program Table Considerations

In CICS/ESA environments, the following programs should be defined in the Processing Program Table (PPT) with EXECKEY(CICS): EDICRIN, EDICRSP, EDICRTS, and EDIXSOX.

DataInterchange Supplied Transactions

DataInterchange provides a set of CICS transactions to perform many different functions. The following summary describes each transaction and its associated function:

Transaction	Description
-------------	-------------

EDIA	The online administrative transaction. Use this transaction to customize in the CICS environment.
------	---

EDIB	The DataInterchange Utility transaction. If the DataInterchange Utility is run asynchronously, EXEC CICS START this transaction. EDIB can be started by DataInterchange as part of continuous receive processing.
------	---

EDID	A background transaction used to update network status. EDID does not have a user interface and is used internally by DataInterchange.
------	--

EDIE	A background transaction used to insert DB2 event log entries into the database (see TD Queues EDI1, EDI2, and EDI3). EDIE keeps DB2 event log insertions out of the main commit scope. EDIE does not have a user interface and is used internally by DataInterchange.
------	--

EDIG	A transaction automatically invoked periodically when using the persistent environment.
------	---

EDIQ	The transaction which gains control from MQSeries transaction CKTI when data is received into an MQSeries Queue that has trigger processing (continuous receive) associated with it.
------	--

EDIR	The online or background transaction used to initiate continuous receive requests.
------	--

EDIS	The online or background transaction used to terminate continuous receive requests.
------	---

EDIT	A background transaction used for storage and resource management. EDIT is a long-running CICS transaction that is idle unless honoring a request on behalf of another DataInterchange transaction. EDIT can be removed from the system using one of two different methods:
------	---

- Add program EDIXSOX to the CICS system pre-termination program list table (PLT). This program automatically removes the EDIT transaction from the system during normal shutdown, and must be executed during the first phase of CICS shutdown processing.
- Type EDIT from any terminal. No parameters are necessary. All other DataInterchange/CICS activity must be quiesced before entering EDIT.

EDIT is automatically initiated when the first DataInterchange request is issued in a region. This is the beginning of the DataInterchange session. When EDIT is removed from the system, the DataInterchange session is considered terminated.

EDIV	The installation verification transaction. For more information on this transaction, see the <i>DataInterchange Installation Guide</i> .
------	--

EDIW	The DataInterchange Utility invocation transaction. For more information, see “Using EDIW to Invoke the DataInterchange Utility” on page 5-45.
------	--

EDIX	A background transaction used to perform some transaction store updates for other DataInterchange transactions. This transaction does the following:
------	--

- Gets a handle
- Deletes an envelope during a DEENVELOPE command with the DUPENV(Y) option

EDIX must be defined in the CICS Resource Control Table for DB2 installations. EDIX processes a request and remains idle in the system for up to one minute. If one minute

expires, and no other request has been issued by another DataInterchange transaction, EDIX removes itself from the system. If another request is generated in the one-minute wait period, the request is honored and the one-minute wait is reset.

Eventually, as all DataInterchange work is quiesced, EDIX removes itself from the system. If need be, EDIX can be removed from the system manually by typing EDIX at any terminal. No parameters are necessary. This transaction does two things: It is called to get a handle and to delete an envelope during a DEENVELOPE command with the DUPENV(Y) option.

EDIZ The online or background transaction used to clean up continuous receive problems.

Note: Unpredictable results will occur if DataInterchange VSAM files are closed while a DataInterchange transaction is executing.

Performance Monitor User Exit

DataInterchange for CICS can invoke a user exit during enveloping and deenveloping for performance monitoring. See Table 5-5 for definition of the COMMAREA passed to the exit. The exit will conditionally be invoked based on whether a user exit program name is specified in the APPDEFS profile. This function is valid only for DataInterchange for CICS.

For performance monitoring to be meaningful, it would be done in tandem with Expedite/CICS. Typically, DataInterchange and Expedite/CICS work asynchronously with one another (except for continuous receives). To understand resource utilization involved in data translation/transmission, the DataInterchange and the Expedite/CICS components would have to be considered together. This can be achieved by using the performance monitor user exit capabilities of both products. The COMMAREA format passed to each is the same. Table 5-5 describes which fields are filled in by DataInterchange and which fields are filled by Expedite/CICS.

On the send side, DataInterchange could invoke the user exit during enveloping and, subsequently, Expedite/CICS could invoke it after sending. These two disparate invocations could be associated using the interchange key fields:

- Sender ID
- Receiver ID
- Control number
- Direction

Likewise, on the receive side, Expedite/CICS could invoke the user exit after receiving the interchange and subsequently DataInterchange could invoke it during deenveloping. If more than one envelope is generated or deenveloped during a single execution of DataInterchange, the user exit would be invoked for each envelope.

By using these user exit capabilities, calculations could be done on the collective resources consumed by both products in handling an interchange. The user exit would most commonly be used to stamp SMF records (through EXEC CICS MONITOR command) with the information passed to it. The SMF records could then be analyzed to produce the desired performance statistics. This information would be helpful for system tuning and in capacity planning.

Table 5-5 (Page 1 of 2). Format of Performance Monitor Commarea

Name	Type	Offset	Length	Owner	Description
RESPCODE	CHAR	0	5	Exp	Response code
RESPTYPE	CHAR	5	8	DI/Exp	DI - 'ENVELOPE' or 'DENVELOP'

Table 5-5 (Page 2 of 2). Format of Performance Monitor Commarea

Name	Type	Offset	Length	Owner	Description
DIRECT	CHAR	13	1	DI/Exp	'S' or 'R'
SENDQUAL	CHAR	14	4	DI	Interchange sender qualifier
SENDID	CHAR	18	35	DI/Exp	Interchange sender ID
RECVQUAL	CHAR	53	4	DI	Interchange receiver qualifier
RECVID	CHAR	57	35	DI/Exp	Interchange receiver ID
CNTRLNO	CHAR	92	14	DI/Exp	Interchange control number
SACCT	CHAR	106	8	Exp	Sender's Information Exchange account
SUSERID	CHAR	114	8	Exp	Sender's Information Exchange user ID
ALIASTYP	CHAR	122	1	Exp	Receiver's alias table type
ALIASID	CHAR	123	3	Exp	Receiver's alias table ID
RACCT	CHAR	126	8	Exp	Receiver's Information Exchange account
RUSERID	CHAR	134	8	Exp	Receiver's Information Exchange user ID
DESTTYPE	CHAR	142	1	Exp	Receiver's destination type
MSGUCLAS	CHAR	143	8	DI	Message user class
UNIQUEID	CHAR	151	8	Exp	DI status update unique ID
DATE	CHAR	159	8	Exp	Current date - YYYYMMDD
TIME	CHAR	167	6	Exp	Current time - HHMMSS
APPLID	CHAR	173	8	Exp	CICS application ID
TRANID	CHAR	181	4	Exp	CICS transaction ID
TASKNO	CHAR	185	7	Exp	CICS task number
ENVSIZE	CHAR	192	11	DI	Envelope size
NUMTRXS	CHAR	203	11	DI	Number of transactions in envelope
MSGSIZE	CHAR	214	11	Exp	Information Exchange message size
APPREF	CHAR	225	14	DI	Interchange application reference
RESERVED	CHAR	239	261	N/A	Reserved

Using EDIW to Invoke the DataInterchange Utility

The DataInterchange CICS transaction, EDIW, can be used to invoke the DataInterchange Utility. On entering transaction EDIW, a panel will be displayed that allows Utility Control Information and Utility PERFORM commands to be entered. The fields on the EDIW panel generally relate to the fields in the Utility Control Information structure. For more information, see “Format of DataInterchange Utility Control Information” on page 5-18. EDIW is useful in testing various versions of PERFORM commands until final versions can be established. In this way, your Utility invocation programs do not need to be recompiled or your command files changed to experiment with various PERFORM commands.

To invoke the Utility, simply fill out whatever fields are appropriate on the panel and press Enter. EDIW allows the Utility to be invoked through an EXEC CICS LINK to program EDIFFUT or through an EXEC CICS START of transaction EDIB. When the EXEC CICS LINK method is selected, the Utility invocation results will be returned and displayed on the panel in the form of the severity and condition codes. When

the EXEC CICS START method is selected, the started EDIB task may run either synchronously or asynchronously with EDIW. The actual PERFORM command to be executed by the Utility can be specified directly on the EDIW panel or may be referenced in a command file.

DataInterchange for CICS Utility Invocation

Syncpoint Value.....:	Utility Response Prog:
Command File Name.....:	Utility Response Type:
Command File Type.....:	Terminal ID.....:
Command Delimiter.....:	Process Net Ack File..:
Print File Name.....:	Process Net Ack Type..:
Print File Type.....:	Multiple TSQ Mode.....:
Report File Name.....:	User Area.....:
Report File Type.....:	
Exception File Name..:	
Exception File Type..:	
Tracking File Name...:	Util Severity Code...:
Tracking File Type...:	Util Condition Code..:
Query File Name.....:	Abend Code.....:
Query File Type.....:	Func Ack Built.....:
Application ID.....:	Func Ack Ret Code.....:
Language ID.....:	Func Ack Ext Ret Code:
Command Statements....:	

F3=End F4=Dlt F7=Bwd F8=Fwd F9=Clr Link? Y Wait? N Caps? Y

Table 5-6 lists the active function keys for this panel.

Table 5-6. Active Function Keys for DataInterchange for CICS Utility Invocation		
Key (PF)	Command	Description
PF3	Exit	Ends EDIW and returns to native CICS.
PF4	Delete	Deletes up to ten previous commands from memory.
PF7	RETRIEVE backward roll	Types a previously entered command on the command line; EDIW can store and retrieve up to 10 previously entered commands, and then wraps.
PF8	RETRIEVE forward roll	Types a previously entered command on the command line; EDIW can store and retrieve up to 10 previously entered commands, and then wraps.
PF9	CLEAR command statement	Clears any data currently displayed in the command statement and in the return code fields.

Table 5-7 lists the command fields for this panel.

Table 5-7. Command Fields for DataInterchange for CICS Utility Invocation

Field	Possible Values	Description
Command Statement	PERFORM Statement	Enter any PERFORM STATEMENT. This field is optional; a command file containing PERFORM statements may be entered in the COMMAND FILE field instead.
LINK?	Y OR N	Y indicates that program EDIFFUT will be EXEC CICS LINKed. N indicates that CICS transaction EDIB will be EXEC CICS STARTed.
WAIT?	Y OR N	Only applies when LINK? N. Y indicates that this EDIW transaction should wait for the STARTed EDIB transaction to complete. EDIB will execute synchronously with EDIW. N indicates that this EDIW transaction should not wait for the STARTed EDIB transaction to complete. EDIB will execute asynchronously with EDIW.
CAPS?	Y OR N	Y indicates the command will be translated into UPPERCASE characters. N indicates the command will remain mixed case.

Chapter 6. Interfacing to Other Networks and Applications

Generalized Network	6-2
Point-to-Point Network	6-4
Activating Point-to-Point Connections	6-4
Application-to-Network Flow	6-6
Parameters Passed to the Communication Routine	6-9
Building Network Commands	6-9
Network Operation Profile	6-9
Field Descriptions	6-10
Network Operation Example	6-11
The FSUPPORT Member	6-12
Message Handler	6-13
Special Communication Routine for CICS	6-14
Network Profile Definition	6-14
Network Program Control Information	6-15
Successful Condition	6-17
No Data Received	6-17
Error Occurred	6-17
Message Handler Control Information	6-17
Successful Condition	6-18
Error Occurred	6-18
Continuous Receive Interface (CICS only)	6-19
Invoking the Continuous Receive Interface	6-19
Interfacing with the In*Touch Gateway Communications Package	6-21
Step 1. Create DataInterchange Network Profile Members	6-22
TIGD31 Profile	6-22
TIGD31 Transaction Data Queues (Outbound files)	6-23
TIGV31 Profile	6-23
TIGV31 Transaction Data Queues (Outbound files)	6-24
Step 2. Create DataInterchange Requestor & Trading Partner Profile Members	6-24
TIGD31 And TIGV31 Receive File Names (Inbound files)	6-25
How the DataInterchange Utility builds SYSIN	6-26
Step 3. Create EDINTCMD Partition Dataset Members	6-26
User Provided Parameters for the Gateway	6-27
DataInterchange Provided Parameters for the Gateway	6-29
More About Gateway Send and Receive Files	6-29
Validation of Gateway Parameters	6-29
Substitutable Tags	6-30
Step 4. Create DataInterchange Utility JCL	6-30
Interfacing DataInterchange for CICS with SDM LinkPlus Interactive	6-34
Outbound Processing Considerations	6-34
Inbound Processing Considerations	6-34
SAP Status Tracking in DataInterchange	6-34
Outbound Processing	6-35
Inbound Processing	6-35
Extracting SAP Status	6-35
Return Code	6-36
Removing SAP Status From the Database	6-36
Return Codes	6-36
SAP Status Codes Supported by DataInterchange	6-37
Interfacing DataInterchange with MQSeries	6-37

Chapter 6. Interfacing to Other Networks and Applications

This chapter describes the interfaces between your application, DataInterchange, and the network. You can use this chapter to develop programs that communicate with applications and networks.

All communication requests by DataInterchange or by applications using DataInterchange are made through the Communications Application Programming Interface (API). For more information, see "Communication Services" on page 3-105. DataInterchange includes support for multiple networks and allows you to write programs to add support for any network for which support is not included with DataInterchange. DataInterchange uses profiles that define the network (NETPROF) and profiles that define the operations supported by that network (NETOP). The communication API requests listed below represent the minimum support required from a network for DataInterchange to function properly.

- Send transactions, function code of 211 with network operations of SENDEDI, SENDUCS, SENDX12, and SENDFILE ("API - Send Transactions and Restart Send Transactions" on page 3-109).
- Send files, function code of 221 with network operation of SENDFILE ("API - Send Files" on page 3-115).
- Receive, function code of 232 with network operations of RECVEDI, RECVUCS, RECVX12 and RECVFILE ("API - Receive and Restart Receive" on page 3-117)
- Cancel, function code of 233 with network operation of CANCEL ("API - Cancel" on page 3-121).
- Return file name, function code of 300 ("API - Return Filename" on page 3-123).
- Retrieve status, function code of 252 with network operation of RECVMSG ("API - Process Network Acknowledgments" on page 3-125).

DataInterchange provides support for two types of networks:

- A generalized network with the following characteristics:
 - A network program is provided by the network provider, such as IEbase provided by the IBM Global Network and DSXMIT2 provided by the General Electric Information Services Company.
 - The interface to the network program is a file interface where commands are placed in an input file by the requestor and responses are placed in the output file by the network program. DataInterchange has no control over this interface; it is defined by the network provider.
 - The network is able to process multiple interchanges in a single file and uses information from the interchanges to determine the destination.
 - A connection is not made to any particular trading partner but rather to the network itself, which then holds data in a mailbox until requested by the trading partner.
- A point-to-point network with the following characteristics:
 - A generalized communication package is used rather than a specific program designed for a particular network.
 - The connection is made directly to the trading partner.
 - The file transmitted must contain data for a single trading partner.

Note: These interfaces may be used if you need to provide an interface to a network not directly supported by DataInterchange. Some details of each interface are provided in the following sections.

Generalized Network

Figure 6-1 on page 6-6 illustrates how an API request from an application flows into the DataInterchange communication module. The communication module reads all the necessary profiles and invokes the communication routine defined to handle the network. The name of the communication routine is provided in the *Communication rtn* field of the network profile (NETPROF). The communication routine passes information between the application and the network program and must:

- Process the API request
- Build the appropriate interface to the network program
- Interpret the results from the network program

If you are providing an interface to your own network, you must:

1. Write a message handler to process the responses generated by the network program. The logical name of the message handler is specified in the *Message handler* field of the Network profile. The physical load module name and the implementation language for the message handler are specified in the User Program Information (ADAMCTL) profile. See “Message Handler” on page 6-13 for more details.
2. Design and enter the network operations (NETOP profile entries) to build the commands required by the network program.
3. Determine which of the DataInterchange provided generalized communication routines you want to use.

There are ten logical names that may be used in the communication routine field of the network profile. The names are:

- VANINFC
- VANIINR3
- VANIINB1
- VANEXPV4
- VANICICS
- VANIMQ
- PTTOPT
- TIGCMD31
- TIGCMV31
- GEISVAN

There are very slight differences in the operation of these programs, and you should choose one that meets your needs. The variations of the programs are:

- VANEXPV4 is for CICS only and is an alternative to VANINFC. For more information, see “Sent to Network Status” on page 5-26.
- VANICICS is for CICS only and simply fills in several pre-defined data blocks and then passes control to the user-supplied network program specified in the Network program field of the network profile. For more information, see “Special Communication Routine for CICS” on page 6-14.
- PTTOPT identifies the point-to-point network. For more information, see “Point-to-Point Network” on page 6-4.
- TIGCMD31 identifies the Gateway direct connection. For more information, see “Interfacing with the In*Touch Gateway Communications Package” on page 6-21.
- TIGCMV31 identifies the Gateway VAN connection. For more information, see “Interfacing with the In*Touch Gateway Communications Package” on page 6-21
- VANIMQ is for the exclusive use of MQSeries send and receive.

- The VANIINB1 program changes a value of G in the RECVTYP field of the CMCB to a value of N. For more information, see “Communication Interface Control Block (CMCB)” on page A-34.
- If the return code from the network program is zero, VANIINB1 does not update the status of the interchanges but assumes the message handler updates the status of all interchanges. The other programs update the status based on the return code and assume the message handler changes the status if necessary.
- VANINFC updates the network sequence value once for each interchange in a file. The other programs update the sequence number once for the entire file.
- VANIINB1 and GEISVAN request status for a single requestor at a time. The other programs request status for 10 requestors at a time.
- GEISVAN uses a network operation of STATUS to request status from the network. The other programs use a network operation of RECVMSG.
- The message handler for GEISVAN only processes the status update responses from DSXMIT2 during a STATUS request. All other responses from DSXMIT2 are processed directly by GEISVAN. For the other programs, the message handler processes all responses.
- GEISVAN asks for status to be placed into a file with a ddname of GEISTAT. VANIINB1 asks for status to be placed into a file with a ddname of INB1STAT. The other programs assume the status is written in the network output file.
- The default transaction data queue for GEISVAN is GEISQ. The other programs have a default value of QDATA.
- The default network input file for GEISVAN is DSXMIPT. The other programs have a default value of INFILE.
- The default network output file for GEISVAN is SYSOUT. The other programs have a default value of OUTFILE.
- The GEISVAN does not pass any network parameters to the network program.
- The GEISVAN does not expect a return code from the network program. The other programs expect a return code with the following values and meanings:

Value	Meaning
<0	Serious error. Status is send request error (42).
0	Request processed without error. Status is send requested (48).
1 - 4	Request processed with warnings. Status is sent with errors (41).
>4	Request processed with errors. Status is send request error (42).

- GEISVAN does not assume RESTART is an option. The other programs check the FRESTART field of the FSUPPORT network operation and invoke the network program with a RESTART option if restart is supported by the network.

The network program is called using an MVS ATTACH macro by the communications routine after the commands dictated by the network operation (for example, SENDEDI, RECVFILE) have been written to the input file. The name of the input file is contained in the *Network input file* field of the network profile entry and the commands written to this file are created from directions stored in the network operation profile (NETOP) entries. The NETOP entries contain instructions for pulling data from various sources along with literals to create commands. For more information, see “Building Network Commands” on page 6-9. The network program may be written in any language with any attributes because it is accessed using an MVS ATTACH macro. The parameters passed to the network program are specified in the *Network parameters* field in the network profile entry.

A generalized communication routine in combination with network operations (NETOP) should be able to create the commands necessary for the network. Each network produces responses (written to the output file identified by the *Network output file* field in the network profile entry) that have a unique format. Therefore, if you are providing your own network you must provide a program that processes those responses to detect errors and perhaps to update the status of interchanges within DataInterchange. This program is called a message handler and the name of the message handler is contained in the *Message handler* field of the network profile entry, which must be defined in the user program information profile (ADAMCTL). For more information, see “Message Handler” on page 6-13.

Point-to-Point Network

A point-to-point network is signaled by specifying PTTOPT in the *Communication rtn* field of the network profile (NETPROF).

DataInterchange for MVS provides a communication routine called PTTOPT for sending and receiving documents through a point-to-point connection. The point-to-point connection allows you to direct transactions to a file not associated with a network.

When the communication routine (PTTOPT) receives a request to queue a transaction, the transaction is written to a system-generated file specifically for the trading partner. The file name is a concatenation of the current user ID and the trading partner nickname, with a period inserted every eight characters. For example, if the user ID is MKRUEGER and the trading partner nickname is JWILLIAMS, the file name is MKRUEGER.JWILLIAM.S. The file must already exist, and because the file is allocated dynamically, there are no JCL or CLIST requirements for the file. Transactions are appended to the end of the file; you must clear the file of transactions that are successfully sent to the trading partner.

PTTOPT forwards all other requests to your send/receive program, passing the same parameters that were passed to it. For more information, see “Parameters Passed to the Communication Routine” on page 6-9. Your program sets return codes and control block information that are expected by the requesting application.

Activating Point-to-Point Connections

To activate point-to-point communications, follow these steps:

1. Add a member to the user program information profile (ADAMCTL) for the point-to-point program.
2. Add a point-to-point member to the network profile. The following entries must exist in the new member:
 - PTTOPT for the *communication routine*
 - The logical name of your send/receive program for *network program*
3. Enter the name of the point-to-point member that you added to the network profile as the *network ID* in the trading partner profile members for the point-to-point trading partners.
4. Enter the name of the network profile member as the *network ID* in the requestor profile members for the point-to-point requestors.
5. Add a member called *netid* FSUPPORT to the network operation profile, where *netid* is the network ID in your network profile. A literal in the command field value (CMDVAL) of this member indicates functions that are supported by the point-to-point connection.

You can create the point-to-point member by copying the member IINB1 FSUPPORT and changing the network ID and CMDVAL literal. For details about the CMDVAL literal, see “The FSUPPORT Member” on page 6-12.

The flow for a point-to-point network is similar to the flow for the generalized network (described by Figure 6-1 on page 6-6) with the following exceptions:

1. The network program is called as a DataInterchange exit rather than using an MVS ATTACH. This means the program must be defined in the ADAMCTL profile and must be written in a language supported by DataInterchange. See Chapter 4, "Exit Routines" for a description of user exits. The parameters passed to the point-to-point network program are the same as those passed to the communication routine.
2. The network program receives the same parameters as the communication routine (a programming interface rather than a file interface).
3. There is no message handler for a point-to-point network. The network program is responsible for processing the request and processing the results expected by the API definition.

Application-to-Network Flow

Figure 6-1 provides an overview of the flow between an application and a network program.

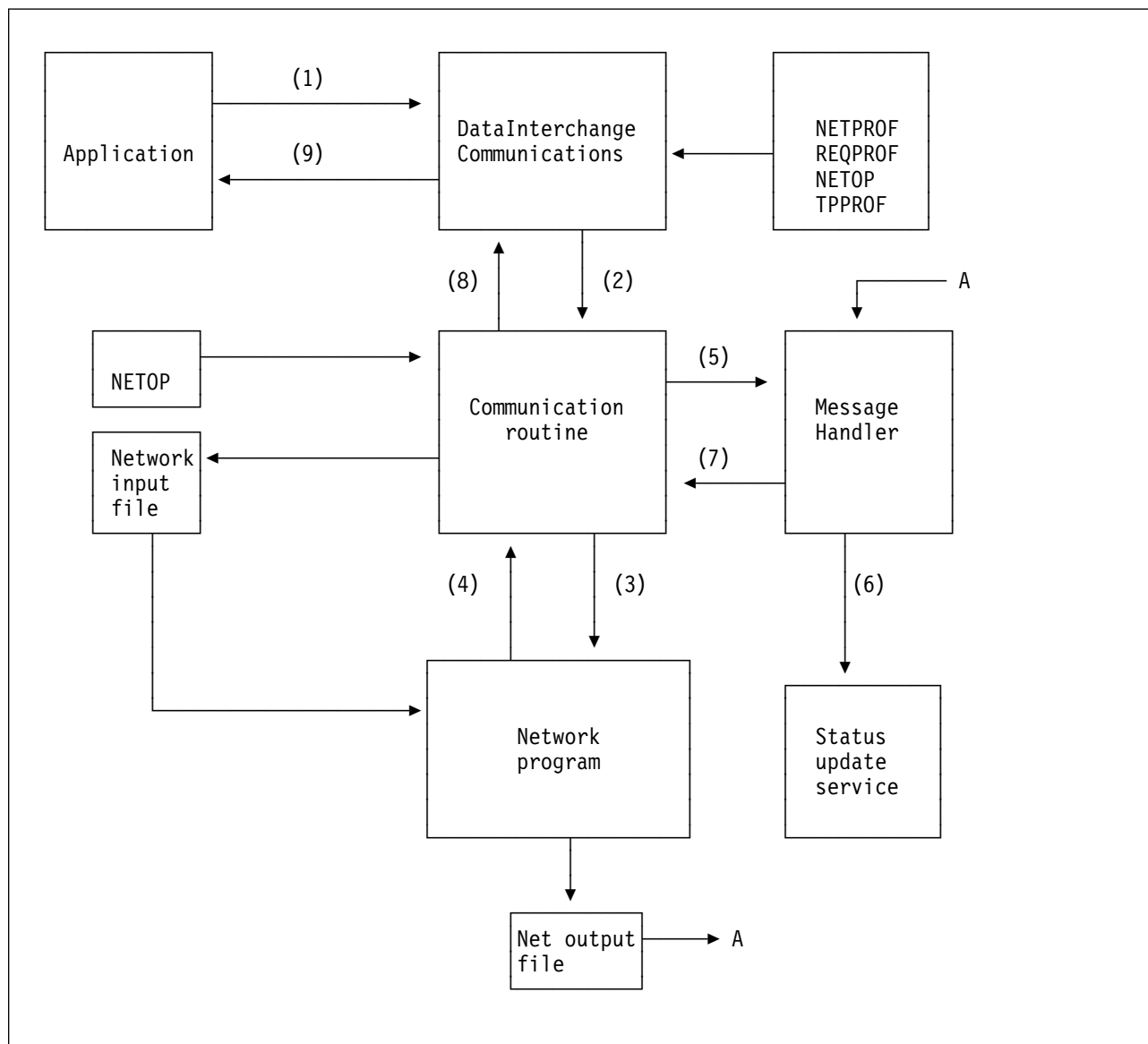


Figure 6-1. Application Interface with the Network

The numbers in the diagram refer to the following sequence of events:

Step Process

- (1) The process begins when an application program requests DataInterchange communication services through the application programming interface (API) defined in "Communication Services" on page 3-105. Communication services retrieves:
 - The requestor profile member (REQPROF) identified by the REQID field of the communications control block.
 - The trading partner profile member (TPPROF) identified by the TPNICKNM field of the communications control block. This retrieval occurs only if the trading partner data block passed to communications contains blanks.

- The network profile member (NETPROF) identified by the requestor member or the NETID field in the communications control block.
- The functions supported by the network from the FSUPPORT member in the network operation profile (NETOP).

(2) Using the *Communication rtn* field from the network profile member, communications invokes this program as a user-exit routine through the language interface routine (FXXZccc). The name of the *Communication rtn* in the network profile member is a logical name. It must be one of the following:

- VANINFC
- VANIINR3
- VANIINB1
- VANEXPV4
- VANICICS (not applicable for this flow)
- VANIMQ (not applicable for this flow)
- PTTOPT
- TIGCMD31 (not applicable for this flow)
- TIGCMV31 (not applicable for this flow)
- GEISVAN

(3) The communication routine has control. Communication routines are driven by the function requested (send, receive, or cancel) and the network operation that is specified in the NETOP field of the communications control block. The concatenation of the 8-byte network ID and the 8-byte network operation is used as a partial key to retrieve all matching entries from the network operation profile (NETOP). The NETOP members contain instructions for building commands, which are literal values combined with data from the other control blocks and profile members. The commands are written to a file as specified by the *Network input file* field from the network profile.

If transaction data is being sent (function code 211), the file of transactions is scanned and each interchange in the file is updated to have a status of send started (46). The following fields are also associated with the interchange:

- Message name (MSGNAME)
- Message user class (ENAME)
- Message sequence number (SEQNUM)
- Network acknowledgment expected flag (ACKIND)
- Send date
- Send time

When the building and writing of commands to the network input file is finished and status has been updated, the program identified in the *Network program* field is invoked using the MVS ATTACH facility. The parameters passed to the network program are specified in the *Network parameters* field of the network profile (network parameters are not passed to the network program when the communication routine is GEISVAN).

(4) This is a synchronous operation. The communication routine waits until the network program is finished. When that program returns control, it is assumed that the operation is completed or that the network program is responsible for making sure the operation completes. A return code of 0 from the network program indicates success. A return code greater than 0 but less than 5 indicates there were no serious errors. A return code of less than 0 or greater than 5 indicates the function failed completely.

Note: A return code of 0 is assumed when the communication routine is GEISVAN.

- (5) If the function requested was to send transaction data, the communication routine updates the status in the Transaction Store of each interchange in the file based on the return code from the network program. A return code of zero results in a status of send requested (48). A return code greater than 0 but less than 5 results in a status of sent with errors (41). Any other return code results in a status of send request error (42).

Note: If the communication routine is VANIINB1 and the return code from the network program is zero (0), the communication routine does not update the status. In this case, it is the responsibility of the message handler to update the status of each interchange in the file. The reason for this is that VANIINB1 was targeted for the Expedite network program and when Expedite has a return code of 0, it has written an entry for each interchange to the network output file (OUTMSG). The message handler for Expedite can read these records and update the status of each interchange.

However, if the return code from the network program is not zero, VANIINB1 initializes the status of each interchange to send request error (42), and it is the responsibility of the message handler to update the status for those interchanges that were successfully sent. With a nonzero return code, the assumption is that the network program may not have been able to (or chose not to) complete its processing. Therefore, the network output file (OUTMSG) may not contain an entries for all interchanges, but only entries for those that were successfully processed.

For example, if you are sending a file that contains four X12 interchanges but the second interchange in the file has an error that prevents the network program from processing it (such as an unknown destination), VANIINB1 will initialize the status of all four interchanges to send request error (42), because the return code from the network program should NOT be zero now.

The message handler updates the status of the first, third, and fourth interchanges to send requested (48), assuming the network program continued processing after the error. If the error in the second interchange was severe enough to cause the network to stop processing (such as an I/O error reading the file), only the status of the first transaction is updated by the message handler to be send requested (48), and the status of the other three interchanges remain send request error (42).

A message handler is driven by the responses in the network output file created by the network program. A message handler can only update the status of interchanges it has been told about via the responses of the network program. The VANIINB1 communication routine assumes that if the return code from the network program is 0, a response record of some type will have been written to the network output file for every interchange in the file so that the message handler can update the status.

The communication routine uses the *Message handler* field from the network profile member and invokes this program as a user-exit routine through the language interface routine (FXXZccc). For more information about message handler programs, see "Message Handler" on page 6-13.

- (6) If the network output file contains status information about previously sent data, the message handler should use the status update service to update the status information in the Transaction Store. For more information, see "Status Update Services" on page 3-126.
- (7) If the network output file indicates that data has been received, the message handler should set the FILERCVD flag in the communications control block to Y and put the account number and user ID of the trading partner sending the data into the trading partner data block.
- (8) The communication routine uses a reverse lookup on the trading partner profile members to locate the entry belonging to the returned account number and user ID. The trading partner nickname is returned to the application program so that it knows data was received and from what trading partner. If any errors were encountered, the communication routine writes an entry to the event log indicating the type of error that occurred. The NPSEVER and NPERRCD fields

from the communications control block are included in the VN1015 message logged by the communications routine.

The communication routine acts on the CLRFILE flag in the communication interface control block, clearing a file just sent if requested to do so.

- (9) The application regains control with the results of the operation indicated by the ZCCBRC and ZCCBERC fields of the CCB.

Parameters Passed to the Communication Routine

Communications invokes the communication routine to process each request from an application that requires network activity, such as requests to send or receive transactions. Your application program supplies the required values before calling communications as defined by the communications API. For more information, see “Communication Services” on page 3-105. The application-supplied data is combined with information from the profiles, and the following parameters are passed to the communication routine. For point-to-point networks (communication routine value of PTTOPT), these same parameters are sent to the network program.

- Service Name Block (SNB); see “Service Name Block (SNB)” on page A-1.
- Common Control Block (CCB); see “Common Control Block (CCB)” on page A-2.
- Function Control Block (FCB); see “Function Control Block (FCB)” on page A-4.
- Communication Interface Control (CMCB); see “Communication Interface Control Block (CMCB)” on page A-34.
- Trading Partner Profile (CMTPPDB); see “Trading Partner Profile Block (TPPDB)” on page A-42.
- Network Profile Block (NPDB); see “Network Profile Block (NPDB)” on page A-53.
- Requestor Profile Block (REQDB); see “Requestor Profile Block (REQDB)” on page A-56.

Building Network Commands

The communication routine (for example, VANIINB1) uses a network operation profile to build command records that it places in the network input file (for example, INMSG). The information needed to build the command records is in the control blocks passed to the communication routine. Entries in the network operation profile tell the communication routine what data from the control blocks should be used in building the commands for the network program.

Network Operation Profile

Table 6-1 on page 6-10 describes the network operation profile. Each member of the profile provides data for a certain field within a command record. The command record field is defined by the CMDLSEQ, CMDFPOS, and CMDFLEN fields of the NETOP profile member. The data for the command record field can either be a literal value (CDMFVAL) or it can be a value from one of the DataInterchange control blocks (BLKNAME and BLKPOS).

The keys to a profile member are the network ID, network operation ID, and network operation field sequence number.

Table 6-1. Definition of the Network Operation Profile

Label	Type	Length	Description
NETID	Character	8	Network ID
NETOP	Character	8	Network operation
FLDSEQ	Character	8	Network operation field sequence
BLKNAME	Character	8	Name of block from which data is taken
BLKPOS	Character	4	Block position
CMDLSEQ	Character	4	Command line sequence number
CMDFPOS	Character	4	Command field position
CMDFLEN	Character	4	Command field length
CMDFVAL	Character	58	Command field value

Field Descriptions

The following describes the network operation profile fields:

Field	Description												
NETID	The name (key) that identifies the network. The network profile must contain a member with this left-justified name. If the name has fewer than eight characters, use blanks to extend the name to eight characters.												
NETOP	The name of a network operation, such as SENDX12. These names are the same as those that can appear in the NETOP field of the communication interface control block. If the name has fewer than eight characters, use blanks to extend the name to eight characters.												
FLDSEQ	The sequential number of this entry in the network operation. For the network operations that are supplied for the IBM Global Network, the sequence number has two parts: <ul style="list-style-type: none"> • The first four bytes are the command line number. • The last four bytes are the field sequence number within the line. 												
BLKNAME	The name of the block from which data is to be taken or the name of a network operation. Valid block names are: <table> <tr> <th>Name</th><th>Description</th></tr> <tr> <td>EDICMCB</td><td>Communication interface control block; for more information, see “Communication Interface Control Block (CMCB)” on page A-34.</td></tr> <tr> <td>EDINPDB</td><td>Network profile block; for more information, see “Network Profile Block (NPDB)” on page A-53.</td></tr> <tr> <td>EDITPPDB</td><td>Trading partner profile block; for more information, see “Trading Partner Profile Block (TPPDB)” on page A-42.</td></tr> <tr> <td>EDIREQDB</td><td>Requestor profile block; for more information, see “Requestor Profile Block (REQDB)” on page A-56.</td></tr> <tr> <td>(blank)</td><td>See CMDFVAL field.</td></tr> </table>	Name	Description	EDICMCB	Communication interface control block; for more information, see “Communication Interface Control Block (CMCB)” on page A-34.	EDINPDB	Network profile block; for more information, see “Network Profile Block (NPDB)” on page A-53.	EDITPPDB	Trading partner profile block; for more information, see “Trading Partner Profile Block (TPPDB)” on page A-42.	EDIREQDB	Requestor profile block; for more information, see “Requestor Profile Block (REQDB)” on page A-56.	(blank)	See CMDFVAL field.
Name	Description												
EDICMCB	Communication interface control block; for more information, see “Communication Interface Control Block (CMCB)” on page A-34.												
EDINPDB	Network profile block; for more information, see “Network Profile Block (NPDB)” on page A-53.												
EDITPPDB	Trading partner profile block; for more information, see “Trading Partner Profile Block (TPPDB)” on page A-42.												
EDIREQDB	Requestor profile block; for more information, see “Requestor Profile Block (REQDB)” on page A-56.												
(blank)	See CMDFVAL field.												
BLKPOS	The starting position of the source data within the block indicated by BLKNAME. The position is relative to 1 (first byte of the block has a position of 1).												

CMDLSEQ	The sequence number of the command record to which the source data should be moved, or an asterisk to use the current command line.
CMDFPOS	The starting position in the command record where the source data should be moved, or an asterisk to use the current field position.
CMDFLEN	The length of data to move from the source location (BLKNAME, BLKPOS) to the command record (CMDLSEQ, CMDFPOS). The length of a command record is defined in the Input record length field of the network profile.
CMDFVAL	A literal used in the command field. This literal value is used only when BLKNAME is blank. When BLKNAME is not blank, this field can contain comments.

CMDFVAL can also contain one of these special literals:

//IMBED// to use the network operation indicated by the BLKNAME field
//STRIP// to remove trailing blanks from data moved to the command line

Network Operation Example

Table 6-2 describes the network operation entries used to build the commands for sending X12 standard transactions over the IBM Global Network. The NETID (IINR3), NETOP (SENDX12), and FLDSEQ (1-n) fields are not shown in the table.

Table 6-2. Network Operation Example

BLKNAME	BLKPOS	CMDLSEQ	CMDFPOS	CMDFLEN	CMDFVAL	Notes
		1	1	3	CSS	Start session command
EDIREQDB	38	1	4	8		Account
EDIREQDB	70	1	12	8		User ID
EDIREQDB	102	1	20	16		Old and new password
EDINPDB	148	1	36	5		Time zone
EDINPDB	153	1	41	8		System type
EDINPDB	161	1	49	4		System level
		2	1	3	CSP	Send EDI File command (part 1)
EDITPPDB	360	2	4	1		Network message class
EDITPPDB	361	2	5	1		Message charge code
EDICMCB	81	2	6	1		Acknowledgment type
EDICMCB	99	2	7	8		Message name
EDICMCB	61	2	15	5		Message sequence number
EDICMCB	80	2	20	1		Message delivery class
EDICMCB	91	2	21	8		Message user class
		3	1	3	CSP	Send EDI file command (part 2)
EDICMCB	78	3	4	1		Data type
EDICMCB	107	3	5	56		File/ddname
		4	1	3	CSE	Session end command

The network operations build the following four commands:

```

-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6
CSSacctnum userid oldpswd newpswd W05004381 R14
CSP 6B 00164 MSGCLSY 10F
CSPDQDATA
CSE

```

The FSUPPORT Member

Communications determines the support provided by a network before calling the communication routine for the network. It does this by reading the network operation profile using the key *netid* FSUPPORT. The literal in the command field value (CMDFVAL) of the member indicates which functions are supported: the first byte corresponds to FQUEUED; the second byte corresponds to FMSGGS, and so on. (See the table below.)

Table 6-3. Description of FSUPPORT Bytes

Byte Name	Description
FQUEUED	Y specifies the queued functions are supported
FMSGGS	Y specifies the free-form messages are supported
FFILE	Y specifies the free-form files are supported
FEDIX	Y specifies the ISA/IEA files are supported
FEDIE	Y specifies the UNB/UNZ files are supported
FEDIU	Y specifies the BG/EG files are supported
FEDIG	Y specifies the GS/GE files are supported
FEDII	Y specifies the ICS/ICE files are supported
FEDIT	Y specifies the STX/END files are supported
FCANCEL	Y specifies the CANCEL is supported
FCLASS	Y specifies the message class is supported
FAACK	Y specifies the network acknowledgments are supported
FSYSMSG	Y specifies the system messages are supported
FRCVBTP	Y specifies the receive by trading partner is supported
FRESTART	Y specifies the restart is supported
FNOUSERID	Y specifies the account number only
FACCTSEP	Value to separate account number and user ID

Note: Values in the FSUPPORT member must be in uppercase.

For non-BG interchange envelopes: If the value of FNOUSERID is Y, the full account number is concatenated with the full user ID. If the value is not Y:

- For UNB and STX envelopes, or if more than seven characters were entered in the account number field, then all trailing blanks are removed from the account number, the separator defined by the FACCTSEP field (must be a blank, period, or slash) is concatenated followed by the user ID. This value is then truncated to the maximum allowed by the standard.
- For ISA and ICS envelopes, where seven or fewer characters were entered in the account number field, seven bytes of the account number will be concatenated with eight bytes of the user ID.

Message Handler

The message handler program is identified in the *Message handler* field of the network profile. The message handler is architecturally the same as a user exit and, therefore, has the same language and linkage edit considerations as user exits. For more information, see Chapter 4, “Exit Routines.”

The value in the *Message handler* field is the logical name for the exit. The physical load module name and the implementation language for the message handler are defined in the user program information profile (ADAMCTL). An ADAMCTL entry is not needed for the network profile entries that are distributed by DataInterchange as these programs have been predefined in internal DataInterchange tables.

The communication routine passes the following parameters to the message handler:

- Service Name Block (SNB); see “Service Name Block (SNB)” on page A-1.

Before calling the message handler, the communication routine moves the name of the message handler from the network profile to the **ZSNBNAME** field of this block.

- Common Control Block (CCB); see “Common Control Block (CCB)” on page A-2.

The message handler returns a value of 12 in **ZCCBRC** to indicate that the message handler could not process the network output file. Any other nonzero value returned in **ZCCBRC** indicates the message handler found something wrong.

- Function Control Block (FCB); see “Function Control Block (FCB)” on page A-4.

Before calling the message handler, the communication routine moves the value 1 to the **ZFCBFUNC** field of this block.

- Communication Interface Control Block (CMCB); see “Communication Interface Control Block (CMCB)” on page A-34.

The message handler sets the FILERCVD flag. The message handler also returns a value in the NPSEVER and NPERRCD fields.

- Trading Partner Profile Block (CMTPPDB); see “Trading Partner Profile Block (TPPDB)” on page A-42.

The message handler sets the ACCTNUM and USERID fields.

- Network Profile Block (NPDB); see “Network Profile Block (NPDB)” on page A-53.

A message handler is invoked by the communication routine after the network program has returned. The message handler is expected to process the responses the network program has written to the network output file. The message handler does the following:

- Processes data in the network output file.
- Sets the FILERCVD field in the CMCB to Y and the account number and the ACCTNUM and USERID fields in the CMTPPDB to the account number and user ID of the sender if the network output file indicates that data was received. Setting the ACCTNUM and USERID fields is only required by DataInterchange programs when the Interactive Entry Facility (IEF) is being used to receive non-standard data. When standard data is received, fields from the interchange header are used to determine the sending trading partner.
- Calls the status update service to update the status in the Transaction Store, if status update messages are contained in network output file.

For more information, see “Status Update Services” on page 3-126.

- Sets the error code in the NPERRCD field and the severity in the NPSEVER field of the CMCB if errors are noted in network output file. It also sets the ZCCBRC field in the CCB to indicate that the

NPSEVER and NPERRCD fields have meaning. The communication routine uses the NPERRCD and NPSEVER value when logging a VN1015 error message.

Special Communication Routine for CICS

A special communication routine has been provided for the CICS environment. This communication routine (logical name VANICICS) uses a LINK interface to pass control to the network program and the message handler. If you do not plan to communicate with trading partners using Expedite/CICS, this special communication routine may meet your needs. It allows you to provide your own network program and message handler with a natural CICS interface. VANICICS passes control information to the network program and message handler via the CICS COMMAREA. It also expects you to overwrite the COMMAREA with resulting information and will log errors if your programs deem the execution unsuccessful. VANICICS provides the interchange queueing function. Whenever an interchange is generated through an ENVELOPE function, the translator indirectly invokes the associated communication routine to write the interchange to the appropriate TS queue. Management Reporting functions are also provided. Anytime a EDI data receive occurs, VANICICS invokes management reporting to update appropriate statistics.

Network Profile Definition

The first task in using VANICICS is to define a new network profile member. For more information on defining a network profile member, see the *DataInterchange Administrator's Guide*. The following is a sample network profile member definition using VANICICS:

```
Profile ID . . . : NETPROF

NETWORK ID . . . : LU62NET
NETWORK NAME . . : NETWORK USING VANICICS
COMMO ROUTINE . . : VANICICS
NETWORK PROGRAM . : LU62PGM
NETWORK PGM PARMS :
NET CMD INPUT FILE:
NET CMD REC LENGTH:
TRANS DATA QUEUE : EDIQDAT
TRANS DATA REC LEN:
TIME ZONE . . . . :
SYSTEM TYPE . . . :
SYSTEM LEVEL . . . :
MSG TEXT HEADER . :
NET CMD OUT FILE :
MSG PROCESSING PGM: LU62MSG
NET SEQUENCE NBR : 00000
```

The fields filled in within the sample are important and will be described. All other fields are ignored by DataInterchange and may be used as you like. Your programs receive a copy of this profile member. Based on this, your programs can use information stored in these unused fields for your own purposes.

Field	Description
NETWORK ID	This is the name DataInterchange knows this network by. Whatever name you pick, it must be used in the associated requestor and trading partner profile members.

NETWORK NAME	Use this field to briefly describe the network. This is a comment field.
COMMO ROUTINE	To use VANICICS, it is critical for you to fill this field in with VANICICS. This tells DataInterchange that anytime a function handled by a communication routine is invoked, VANICICS is given control.
NETWORK PROGRAM	This is the name of your network program. DataInterchange will CICS LINK to this program anytime a communications function is directed to VANICICS. "Network Program Control Information" defines what information your program receives when it is invoked.
TRANS DATA QUEUE	This is the default TS queue name where interchanges are written when an interchange queueing function is received by VANICICS. Your programs are not invoked during an interchange queueing function, but it is important for your network program to know where the interchanges reside. This name can then be used whenever a send function is directed to your network program.
MSG PROCESSING PGM	This is the name of your message processing program. DataInterchange will CICS LINK to this program during the processing of an UPDATE STATUS request. VANICICS does not invoke this program after other network functions, such as Send or Receive. The usual function of this program is to update the status in the Transaction Store. For more information on the function of a message handler, see "Message Handler" on page 6-13.

Note: The MSG PROCESSING PGM field is treated differently by VANICICS than all other communication routines. For VANICICS, the MSG PROCESSING PGM field contains the name of a physical load module DataInterchange invokes through CICS LINK. For all other communication routines, this field contains a logical service name. In the CICS environment, it is natural to CICS LINK to a program. This invocation method was chosen for VANICICS because it is easier to use the CICS LINK interface in CICS.

Network Program Control Information

The network program you supply receives control information via the CICS COMMAREA. The COMMAREA is made up of a set of four-byte addresses pointing to control blocks. These control blocks contain all the critical information your program needs to process the request. It is important to note that VANICICS does not use network operations profile members as does other communication routines. The interface to your network program is similar to the MVS point-to-point communication routine. Control blocks with critical information are passed to the program instead of the command being built in the COMMAREA. This method of passing control blocks gives greater flexibility to your program. The COMMAREA your program will receive should be defined as follows:

Table 6-4 (Page 1 of 2). Format of Network Program Control Information

Name	Type	Offset	Length	Description
SNBADDR	BIN	0	4	Address of SNB control block; see "Service Name Block (SNB)" on page A-1.
CCBADDR	BIN	4	4	Address of CCB control block; see "Common Control Block (CCB)" on page A-2.
FCBADDR	BIN	8	4	Address of FCB control block; see "Function Control Block (FCB)" on page A-4.
CMCBADDR	BIN	12	4	Address of CMCB control block; see "Communication Interface Control Block (CMCB)" on page A-34.

Table 6-4 (Page 2 of 2). Format of Network Program Control Information

Name	Type	Offset	Length	Description
TPPADDR	BIN	16	4	Address of CMTPPDB control block; see “Trading Partner Profile Block (TPPDB)” on page A-42. Note: This block only contains data if the TPNICKN keyword specifically refers to a trading partner. Otherwise, the TPNICKNM field in this block contains the value DEFAULTTPNICK.
NPDBADDR	BIN	20	4	Address of NPDB control block; see “Network Profile Block (NPDB)” on page A-53.
REQADDR	BIN	24	4	Address of REQDB control block; see “Requestor Profile Block (REQDB)” on page A-56.

Your network program will have to handle three main function codes. Function codes are passed within the function control block (FCB) field **ZFCBFUNC**.

Function Code Description

211	This function code indicates your network program is being requested to send a TS queue containing interchanges. The name of the TS queue to send can be found in the CMCB control block, field FILENAME . The requestor profile block (REQDB) contains information about the mailbox or user ID for which the send is to be done.
232	This function code indicates your network program is being requested to receive interchanges into a TS queue. The name of the TS queue to receive data into can be found in the CMCB control block, field FILENAME . The REQDB contains information about the mailbox or user ID for which the receive is to be done.
252	This function code indicates your network program is being requested to receive status information on behalf of an UPDATE STATUS request. You could use the CMDOUT field in the NPDB block as a place to hold the name of a TS or TD queue to write out the status information. The NPDB control block is an in-storage copy of the associated network profile member. The REQDB contains information about the mailbox or user ID for which the receive is to be done.

When your network program has completed its processing, it must return control to DataInterchange via the CICS RETURN command. DataInterchange expects your network program to indicate success or failure back to DataInterchange. Your network program does this by overwriting the incoming COMMAREA with meaningful information. The overwritten COMMAREA should be formatted as follows:

Table 6-5. Format of Returned Network Program Control Information

Name	Type	Offset	Length	Description
RESPONSE	CHAR	0	5	Response Code
SEVERITY	CHAR	5	2	Response Severity
FILLER	CHAR	7	21	Unused Filler

Successful Condition

If your network program has executed successfully, it should set the fields in the COMMAREA as follows:

- Field **RESPONSE** should be filled in with the character string HI000.
- Field **SEVERITY** should be filled in with the character string 00.
- Field **FILLER** should be filled in with blanks.

No Data Received

Your network program may issue a receive request but there may not be any data available to be received. DataInterchange does not consider this an error condition; however, this fact should be conveyed back to the original calling program. Your network program would indicate back to DataInterchange that a receive was issued but no data was returned by setting the fields in the COMMAREA as follows:

- Field **RESPONSE** should be filled in with the character string HI000.
- Field **SEVERITY** should be filled in with the character string 04.
- Field **FILLER** should be filled in with blanks.

Error Occurred

Your network program may encounter many different errors. You can indicate back to DataInterchange an error occurred and get a VN1022 message logged and an error extended return code of 1022 returned to the caller by filling in the COMMAREA as follows:

- Field **RESPONSE** should be filled in with any five-character string besides HI000. This string should indicate a specific error within your network program to aid in problem determination. This response code will be included in the VN1022 message.
- Field **SEVERITY** should be filled in with the character string 08 or 12. Again, this value is to aid you in determining the cause of the problem and will be included in the logged VN1022 message.
- Field **FILLER** should be filled in with blanks.

Message Handler Control Information

During the processing of an UPDATE STATUS request, DataInterchange will CICS LINK to the message handler. VANICICS does not start this program after other network functions, such as Send or Receive. The message handler program you supply receives control information through the CICS COMMAREA. The COMMAREA is made up of a set of four-byte addresses pointing to control blocks. These control blocks contain all the critical information your program needs to process the request. The COMMAREA your program will be receiving should be defined as follows:

Table 6-6 (Page 1 of 2). Format of Message Handler Program Control Information

Name	Type	Offset	Length	Description
SNBADDR	BIN	0	4	Address of SNB control block; see "Service Name Block (SNB)" on page A-1.
CCBADDR	BIN	4	4	Address of CCB control block; see "Common Control Block (CCB)" on page A-2.
FCBADDR	BIN	8	4	Address of FCB control block; see "Function Control Block (FCB)" on page A-4.

Table 6-6 (Page 2 of 2). Format of Message Handler Program Control Information

Name	Type	Offset	Length	Description
CMCBADDR	BIN	12	4	Address of CMCB control block; see “Communication Interface Control Block (CMCB)” on page A-34.
NPDBADDR	BIN	20	4	Address of NPDB control block; see “Network Profile Block (NPDB)” on page A-53.
TPPADDR	BIN	16	4	Address of CMTPPDB control block; see “Trading Partner Profile Block (TPPDB)” on page A-42.
REQADDR	BIN	24	4	Address of REQDB control block; see “Requestor Profile Block (REQDB)” on page A-56.

The message handler will be invoked and will always receive the same function code, that being 252. Therefore, the **ZFCBFUNC** field within the function control block (FCB) is not important. When your message handler gains control, its primary task is to update Transaction Store status. You do this using the update envelope status API documented on “Status Update Services” on page 3-126. To aid you in using this API, the SNB and CCB control blocks passed into your program are already initialized and ready to be used for this API.

When your message handler program has completed its processing, it must return control to DataInterchange via the CICS RETURN command. DataInterchange expects your message handler program to indicate success or failure back to DataInterchange. Your message handler program does this by overwriting the incoming COMMAREA with meaningful information. The overwritten COMMAREA should be formatted as follows:

Table 6-7. Format of Returned Message Handler Program Control Information

Name	Type	Offset	Length	Description
RESPONSE	CHAR	0	5	Response Code
SEVERITY	CHAR	5	2	Response Severity
FILLER	CHAR	7	21	Unused Filler

Successful Condition

If your message handler has executed successfully, it should set the fields in the COMMAREA as follows:

- Field **RESPONSE** should be filled in with the character string HI000.
- Field **SEVERITY** should be filled in with the character string 00.
- Field **FILLER** should be filled in with blanks.

Error Occurred

Your message handler may encounter many different errors. You can indicate back to DataInterchange an error occurred and get a VN1022 message logged and an error extended return code of 1022 returned to the caller by filling in the COMMAREA as follows:

- Field **RESPONSE** should be filled in with any five-character string besides HI000. This string should indicate a specific error within your message handler program to aid in problem determination. This response code will be included in the VN1022 message.

- Field **SEVERITY** should be filled in with the character string 08 or 12. Again, this value is to aid you in determining the cause of the problem and will be included in the logged VN1022 message.
- Field **FILLER** should be filled in with blanks.

Continuous Receive Interface (CICS only)

The Continuous Receive Facility was originally designed to work with Expedite/CICS and Information Exchange, but it can be used with user-written applications. The intent of this interface is to house DataInterchange processing information in the continuous receive profile, away from the receiving application's logic. The user-supplied receive application does not need to be concerned with the DataInterchange processing that should be performed. This information is contained in the continuous receive profile member. This interface is available in the CICS environment only.

There are three major differences between continuous receive processing which interacts with Expedite/CICS and processing that interacts with user-written receive applications; they are:

- Only continuous receives associated with Expedite/CICS are started and stopped with CICS transactions EDIR and EDIS or PLT programs EDICRTS and EDICRSP. Continuous receives associated with user-written receive applications are not started or stopped through DataInterchange. The user-written receive application passes the name of the continuous receive profile member to DataInterchange when a continuous receive occurs. Since this is all the control information DataInterchange needs to perform the appropriate processing, starting and stopping is not necessary. Any request to start or stop a continuous receive associated with a user-written receive application will be ignored by DataInterchange.
- The requestor profile ID in continuous receive profile members associated with user-written receive applications is optional. If one is supplied in the profile, DataInterchange will update the management reporting receive statistics database. If one is not supplied, management reporting statistics are not updated.
- The requestor profile ID is mandatory for continuous receives associated with Expedite/CICS. The continuous receive profile member ID must be a maximum of 8 characters long. Continuous receive profile member IDs associated with Expedite/CICS can have a maximum length of 16 characters.

Invoking the Continuous Receive Interface

When a user-written receive application receives data from a source (such as a leased line, different VAN, remote TD queue), it can invoke the continuous receive interface to have this EDI data processed. The EDI data must reside in a temporary storage (TS) queue for DataInterchange to process it. When the data resides in a TS queue, the user application uses the CICS LINK command to give control to program EDICRIN. When the CICS LINK command is issued for program EDICRIN, a COMMAREA must be passed. The COMMAREA must have the following format:

Table 6-8 (Page 1 of 2). Format of Control Information Given to EDICRIN

Name	Type	Offset	Length	Description
CRMEMBER	CHAR	0	8	Continuous receive profile member name
RESERVED	CHAR	8	2	Reserved for future use
ENVFILE1	CHAR	10	8	TS queue (1st 32 K records)
USRFLD	CHAR	18	16	User area
RESERVED	CHAR	34	146	Reserved for future use
ENVFILE2	CHAR	180	8	TS queue (2nd 32 K records)

Table 6-8 (Page 2 of 2). Format of Control Information Given to EDICRIN

Name	Type	Offset	Length	Description
ENVFILE3	CHAR	188	8	TS queue (3rd 32 K records)
ENVFILE4	CHAR	196	8	TS queue (4th 32 K records)
ENVFILE5	CHAR	204	8	TS queue (5th 32 K records)
ENVFILE6	CHAR	212	8	TS queue (6th 32 K records)
RESERVED	CHAR	220	36	Reserved for future use

When invoking program EDICRIN, the user-written receive application must be concerned with the following fields:

- CRMEMBER** The name of the continuous receive profile member that contains the DataInterchange processing information. As stated above, this field is limited to 8 characters in length. The profile member contains information such as whether translation is desired, the name and type of the print file, and other critical information for DataInterchange to process the incoming data. For more information on each field contained within the profile, see the continuous receive profile definition in the *DataInterchange Administrator's Guide*.
- ENVFILEx** The name of the TS queues holding the EDI data that DataInterchange is to process. The TS queues may contain multiple interchanges and processing is based on the CRMEMBER field above. For more information about processing more than one TS queue, see "Processing Multiple Incoming TS Queues" on page 5-4.
- USRFLD** Data in this field will be moved to the DataInterchange Utility Control Block USRFLD field. For more information, see Table 5-4 on page 5-20. This data would then be available to subsequent response programs that may get started.

When DataInterchange has completed its processing, it will return control to the user-written receive application. DataInterchange will update the COMMAREA indicating success or failure. The user-written receive application is informed whether or not DataInterchange accepted responsibility for the data. DataInterchange does not return any information as to whether or not the data was processed successfully. This task is to be performed by the response application supplied within the continuous receive profile. The user-written receive application should be sensitive to the returned COMMAREA. If DataInterchange was not able to accept ownership for the data, the user-written response program should be prepared to back up the data. When the problem is fixed, the data can be given back to DataInterchange to process. The first 7 characters in the returned COMMAREA will contain one of the following values:

- HI00000** DataInterchange was able to accept ownership for the data. DataInterchange processing may not have been successful, but the continuous receive response application is responsible for detecting DataInterchange processing errors.
- HI77712** The continuous receive profile member indicated translation and deenveloping was not to be performed. The only DataInterchange processing performed was the invocation of the continuous receive response program. This program returned a value of -1 in the severity code (CCBRC) field, indicating its processing was unsuccessful. The data within ENVFILEx should be backed up.
- HI88812** DataInterchange processing abended in the EDIB transaction. The data within ENVFILEx should be backed up.
- HI99912** The profile member specified in the CRMEMBER field could not be located or the active flag within the profile is set to N. The data within ENVFILEx should be backed up.

If one of the errors above is encountered, DataInterchange will attempt to log an error message to the EXPL transient data queue. EXPL could be directed to a destination where other errors are routed.

Interfacing with the In*Touch Gateway Communications Package

The In*Touch** Gateway provides a multi-dimensional routing system. Through it, messages can be routed directly to and from trading partners, to and from various VANs (for example, IBM Global Network or GEIS), and to and from local Gateway mailboxes. The individual responsible for setting up DataInterchange to use the Gateway must first be familiar with certain aspects of the Gateway product. A good understanding of Gateway entities (for example, the GMJD library and the Inish deck) is necessary. This section does not describe the Gateway product or Gateway setup procedures. It describes only how to build the necessary DataInterchange components that will allow DataInterchange to communicate with the Gateway. Figure 6-2 illustrates the DataInterchange/Gateway configuration.

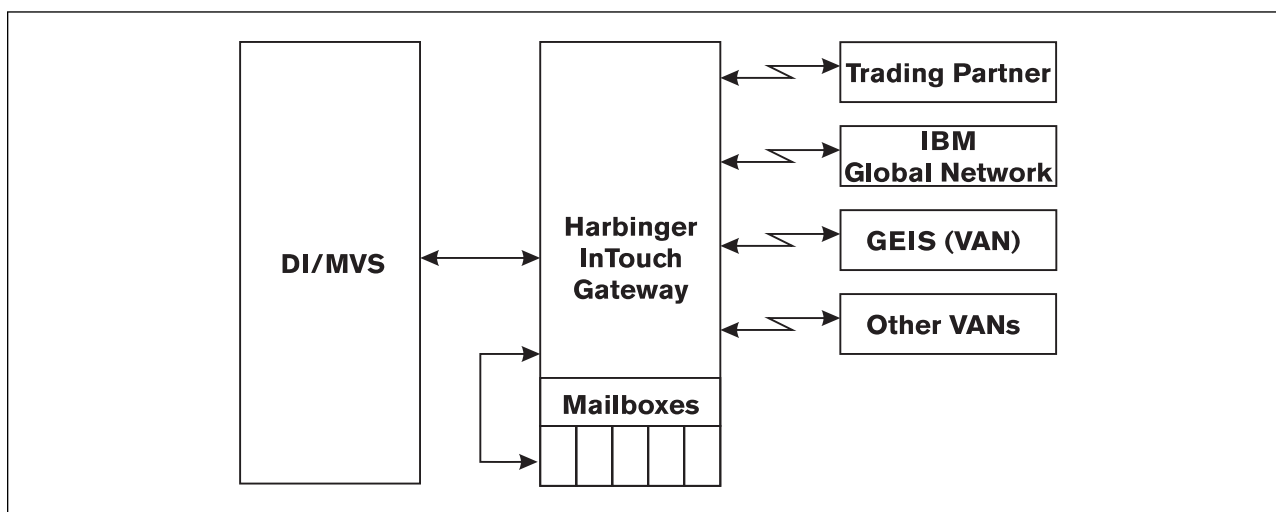


Figure 6-2. Sample DataInterchange/Gateway Configuration

The Gateway is an MVS-based, started task. Currently, only DataInterchange for MVS (not DataInterchange for CICS) can interface with it. The DataInterchange communication routine invokes the Gateway by way of the ATTACH and WAIT macros. The name of the Gateway network program is EDIBTCH. This is the name that must be specified in the network profile members used in association with the Gateway.

The following steps are required to interface DataInterchange with the Gateway:

- | 1. Create DataInterchange Network Profile Members.
- | 2. Create DataInterchange Requestor and Trading Partner.
- | 3. Create EDINTCMD Partition Dataset Members.
- | 4. Create DataInterchange Utility JCL.

Step 1. Create DataInterchange Network Profile Members

Two DataInterchange network profile members, TIGD31 and TIGV31, must be added to support the In*Touch Gateway. Both members must specify EDIBTCH as the network program name, SYSIN as the network input file name, SYSPRINT as the network output file name, and TIGNR31 as the message handler name. The first profile member (TIGD31) must specify a communication routine name of TIGCMD31. This profile member is used to go through the Gateway to connect directly to a trading partner's system or to access local Gateway mailboxes. The other profile member (TIGV31) must specify a communication routine name of TIGCMV31. This profile member is used to go through the Gateway to access mailboxes on VANs (such as, IBM Global Network or GEIS).

The default network output file name for Gateway network profile members is SYSPRINT. Any other value here is ignored. The Gateway always writes output messages to its EDILOG and to SYSPRINT (or back to the terminal if using the Transaction Store Facility). For more information, see the MSG parameter definition in "User Provided Parameters for the Gateway" on page 6-27.

Currently, the Gateway does not support the ability to track transactions. Network acknowledgments may be requested through the Gateway, but not processed back. This means that PERFORM UPDATE STATUS will not work through the Gateway. Message handler TIGNR31 is a NO-OP module. For more information about how to create network profiles members, see "Connecting to Other Networks, Adding a Network Profile Member" in the *DataInterchange Administrator's Guide*.

TIGD31 Profile

The TIGD31 profile is used to go through the Gateway to connect directly to a trading partner's system or to access local Gateway mailboxes.

```
Profile ID . . . : NETPROF

Network ID . . . : TIGD31
Network name . . : GATEWAY DIRECT CONNECTION
Communication rtn  TIGCMD31
Network program . : EDIBTCH
Network parameters
Network input file SYSIN
Input rec length
Trans data queue
Trans rec length
Time zone . . . .
System type . . .
System level . . .
Msg text header . .
Net output file . : SYSPRINT
Message handler . : TIGNR31
Network sequence
Net acks file . .
Dial connect num
```

Figure 6-3. TIGD31 Network Profile Sample

TIGD31 Transaction Data Queues (Outbound files)

For TRANSLATE AND SEND and ENVELOPE AND SEND commands:

- If a FILEID keyword override is not supplied, DataInterchange dynamically builds the physical transaction data queue names. These names are your MVS user ID and the trading partner nicknames. The Gateway is invoked for each separate trading partner. Even though DataInterchange dynamically allocates these files, the files must exist on your system.
- If a FILEID keyword override is supplied, it is used for the transaction data queue name and the Gateway is invoked once with one aggregate file. The FILEID keyword cannot be used with the REQTP keyword on ENVELOPE AND SEND type commands.
- The transaction data queue name from the network profile member is ignored (TIGWYQ is not the default).

For SEND type commands:

- The FILEID keyword override is used for the transaction data queue name if specified. Otherwise, the value from the transaction data queue field in the network profile member is used if specified. Otherwise, the default transaction data queue name is TIGWYQ.

Because of the point-to-point nature of the TIGD31 network, it is possible, and often desirable, to use the REQID and REQTP keywords with your ENVELOPE AND SEND commands. DataInterchange dynamically builds the transaction data queues and passes them on to the Gateway one at a time for each requestor/trading partner combination.

For more information on the PERFORM TRANSLATE AND SEND, the PERFORM ENVELOPE AND SEND, and the PERFORM SEND commands, see “Translating and Sending Transactions” on page 1-15, “Enveloping and Sending or Reenveloping and Sending Transaction” on page 1-12, and “Sending EDI Data” on page 1-10.

TIGV31 Profile

The TIGV31 profile goes through the Gateway to access mailboxes on VANs.

```

Profile ID . . . : NETPROF

Network ID . . . : TIGV31
Network name . . : GATEWAY VAN CONNECTION
Communication rtn  TIGCMV31
Network program . . EDIBTCH
Network parameters
Network input file  SYSIN
Input rec length
Trans data queue
Trans rec length
Time zone . . . . .
System type . . . .
System level . . .
Msg text header . .
Net output file . . SYSPRINT
Message handler . . TIGNR31
Network sequence
Net acks file . . .
Dial connect num

```

Figure 6-4. TIGV31 Network Profile Sample

TIGV31 Transaction Data Queues (Outbound files)

For the TIGV31 network, the FILEID keyword override is used for the transaction data queue name if specified. Otherwise, the value from the transaction data queue field in the network profile member is used if specified. Otherwise, the default transaction data queue name is TIGWYQ.

Step 2. Create DataInterchange Requestor & Trading Partner Profile Members

Applicable requestor and trading partner profile members must be added or updated to support the In*Touch Gateway Interface. The requestor profile fields to be considered here are: network ID, receive file name, and network commands file. The trading partner profile fields to be considered are: network ID and network commands file. Each requestor or trading partner associated with the Gateway must have a network ID of either TIGD31 or TIGV31.

The network commands file name field is critical for Gateway processing. The DataInterchange Utility uses the value specified in the trading partner profile if one is supplied. Otherwise, it uses the value specified in the requestor profile. If a valid value is not found in either profile, a VN1053 error occurs at runtime. You must enter a PDS member name in this field. The PDS must be allocated in your batch DataInterchange Utility JCL with ddname EDINTCMD. If the Transaction Store Facility is to be used, the physical name of the PDS is EDI.TIGATE.EDINTCMD. This PDS and its members must be created by the individual interfacing DataInterchange with the Gateway. For more information, see “User Provided Parameters for the Gateway” on page 6-27. These PDS members contain the parameters to be passed to the Gateway. DataInterchange reads these parameters from the PDS and writes the parameters to the file allocated with ddname SYSIN after resolving all substitutable tags.

TIGD31 And TIGV31 Receive File Names (Inbound files)

Receive file names must be specified in the requestor profiles associated with the TIGD31 and TIGV31 networks. There is no default receive file name. The receive file name must match a ddname in your batch DataInterchange Utility JCL or EDI CLIST. In either case, the file can only be allocated DISP=SHR and cannot be allocated with any other disposition. The FILEID keyword overrides the receive file name from the requestor profile in DataInterchange Utility receive commands. This is the only exception where blank requestor profile receive file names do not cause a VN1053 error at runtime.

Before the Gateway places received data in the receive file, the Gateway always clears the file. If received data is to be accumulated and at some later time deenveloped, a temporary receive file must be used and passed to the Gateway. Then a subsequent jobstep could append the data from the temporary file to the more permanent receive file. Here is an example of such a jobstep:

```
//GENER    EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=EDI.TEMP.RECVDATA,DISP=SHR
//SYSUT2   DD DSN=EDI.PERM.RECVDATA,DISP=MOD
//SYSIN    DD DUMMY
```

In the above example, EDI.TEMP.RECVDATA would be the name of the physical file passed to the Gateway in the SYSIN file (it must be DISP=SHR). EDI.PERM.RECVDATA is the file holding received data that has not yet been deenveloped by DataInterchange.

How the DataInterchange Utility builds SYSIN

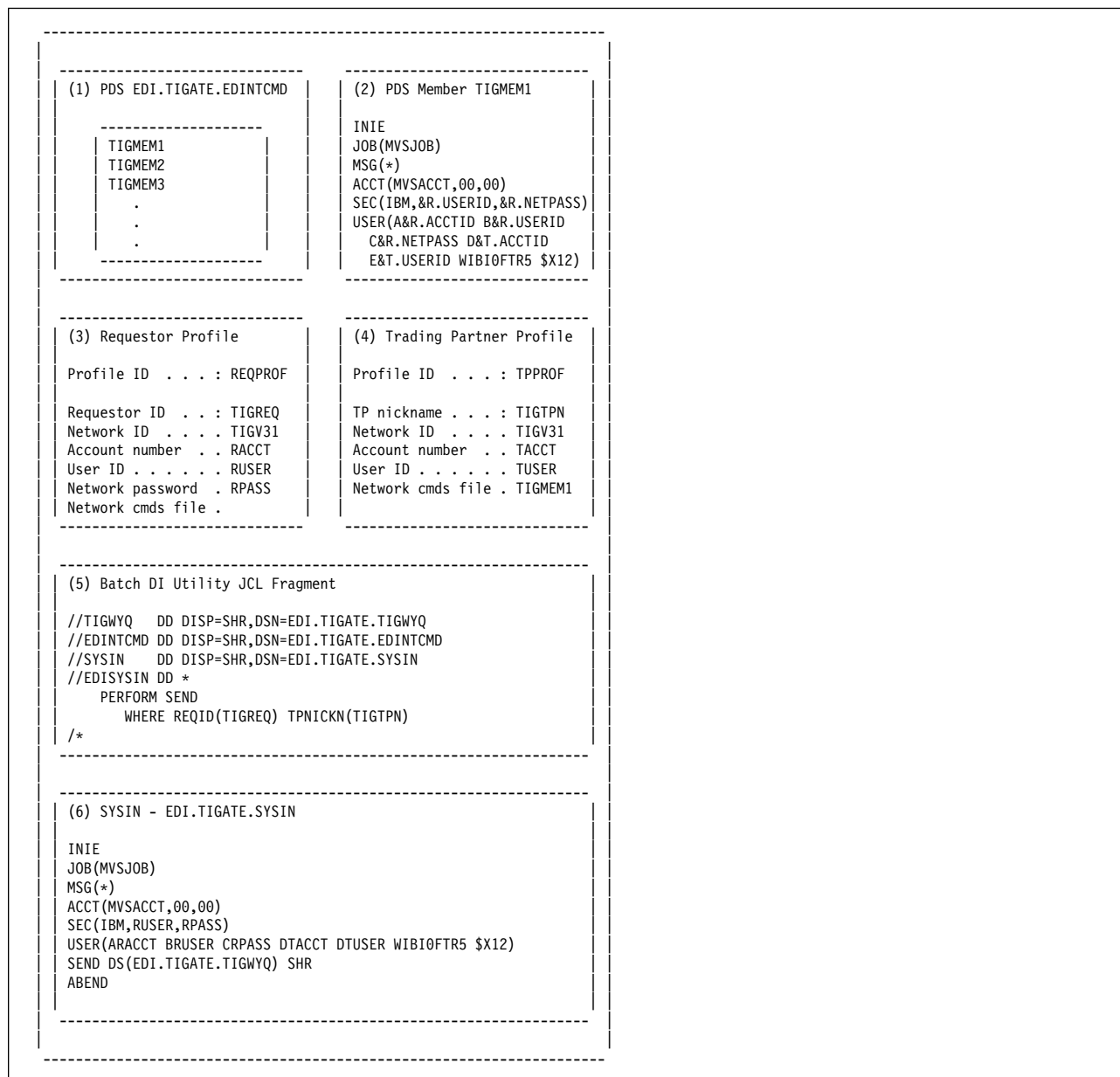


Figure 6-5. Building SYSIN

Step 3. Create EDINTCMD Partition Dataset Members

The following information is needed by the Gateway:

- gmjdlbmember or "Q"
- JOB(jobname)
- MSG(logname)
- ACCT(name, division, costcenter)
- SEC(pdnname or "QUE," user ID, password)
- USER(Aparameter, Bparameter, ... ,Zparameter, %parameter, ...)
- SEND DS(physical.file.name) disp RECEIVE DS(physical.file.name) disp
- KEY(queuekeyid)

For more information about the parameters listed above, see the *Harbinger Gateway User Guide*. This information is needed by the Gateway, not by DataInterchange. Therefore, full descriptions are not provided here. Except for SEND and RECEIVE, DataInterchange merely passes the information onto the Gateway “as is” or after resolving substitutable tags. All the parameters listed above except for SEND and RECEIVE should be provided by the user in a member of the EDINTCMD PDS.

Note: Quotation marks shown as part of a Gateway parameter are used only to indicate that the values inside the quotes are literal values (to be typed exactly as shown). The quotes themselves are not part of the parameter.

User Provided Parameters for the Gateway

The following user-provided parameters are required for the Gateway product:

- gmjdlbmember or “Q”
- JOB(jobname)
- MSG(logname)
- ACCT(name, division, costcenter)
- SEC(pdname or “QUE,” userid, password)
- USER(Aparameter, Bparameter, ... ,Zparameter, %parameter, ...)
- KEY(queuekeyid)

These parameters are to be entered in members of a special partitioned data set (PDS). This PDS must have the ddname EDINTCMD in your batch DataInterchange Utility JCL. The default EDINTCMD PDS name specified in the EDI CLIST is EDI.TIGATE.EDINTCMD. Normally, these parameters are entered into the PDS members by the individual responsible for enabling the DataInterchange/Gateway interface. Generally, a complete set is required in each PDS member. At runtime, DataInterchange reads the appropriate PDS member by the name specified in the network commands file name field in the trading partner profile or in the requestor profile. A valid EDINTCMD PDS member must be provided in either or both profiles. DataInterchange uses the name in the trading partner profile if available. Otherwise, the name from the requestor profile will be used. At runtime, DataInterchange reads the EDINTCMD PDS member specified in one of the profiles and copies the parameters into the file allocated with ddname SYSIN after resolving substitutable tags. Except for tag resolution, DataInterchange copies these parameters “as is” into the SYSIN file and adds the SEND DS or RECEIVE DS parameter. It is the SYSIN file that actually gets passed to the Gateway.

A description of each parameter is below:

- gmjdlbmember or “Q”

One of these options must be chosen. The Q parameter indicates whether communication through the Gateway occurs directly with a trading partner's system, with a VAN, or with a local Gateway mailbox. The value here is the name of a member of the Gateway's Generic Master Job Definition (GMJD) library.

For more information, see the *Harbinger Gateway User Guide*. Examples are: GENERIC, INIE, and QUE. If data is to be sent to or received from a local Gateway mailbox, a special member named QUE is used. It is possible for some jobs to not use any GMJD member, but rather to specify all needed information inline. Such jobs would use the literal Q. If communication is to occur directly with a trading partner's system, GENERIC is used.

The gmjdlbmember parameter is the name of a PDS member from your GMJD library listed in your Gateway Inish deck in a DYNALLOC card with the ddname GMJDLIB. Contained in these members are two parameters that simply get copied into your SYSIN by the Gateway before processing the

entire SYSIN. The two parameters that are contained in these members are what is most important. They are the GRP (groupname) and DAP (dapname) parameters. These parameters tell the Gateway which group of lines to schedule the transaction on (the groupname must be a valid GROUP= parameter on an EDILINE card in your Inish deck). The DAP parameter selects the program that processes your USER parameters correctly and schedules the proper communications emulator for the line type chosen by the GRP parameter.

- JOB (jobname)

Jobname is a 1-8 character name that the user chooses to have associated with the EDI transmission, which can then be used to assist with searches of the Gateway EDILOG. It does not have anything to do with the JCL jobcard.

- MSG (logname)

Logname indicates where the user would like the output from the Gateway routed. Valid options are NONE (send output to the Gateway EDILOG only) and an asterisk (send output to the Gateway EDILOG and back to the user's terminal or SYSPRINT for batch jobs).

- ACCT (name, division, cost center)

Contains information that is used by the Gateway's billing routine on the SMF record created for the job. Typically, the name is character format while the division and cost center are numeric codes. This information must always be provided, even though SMF records are not produced by the Gateway when processing QUE jobs.

- SEC (pdnname or QUE, user ID, password)

This is a security parameter that tells the Gateway which PDN card to use when contacting your trading partner. The pdnname is the name on a PDN card in your Gateway Inish deck. In your Gateway setup, you should have defined PDN cards for your trading partners. The PDN card ties the name of the trading partner to the telephone number or application ID that the Gateway uses to connect to the trading partner. Obtain the pdnname from the proper PDN card in your Gateway Inish deck. If the data is to be sent to or received from a local Gateway mailbox, the literal QUE is used as the pdnname. The user ID and password identify the user requesting the Gateway service. These values are passed to a Gateway security exit.

Because each trading partner with whom you will be connecting directly has a unique phone number, there must be a separate pdnname (and a separate corresponding EDINTCMD PDS member) for each. If, however, a file of transactions bound for different trading partners is to be sent through a VAN that will parse the file after receiving it from the Gateway, a single pdnname (and a single corresponding EDINTCMD PDS member) can be used to pass the file to the network.

- USER (Aparameter, Bparameter, ... ,Zparameter, %parameter, ...)

This parameter is a list of network specific processing options. It is necessary to consult the *Harbinger Gateway User Guide* to know what values to place here. The USER parameter is at the center of the flexibility allowed by the Gateway. Each argument should be looked at as a separate parameter in and of itself. These arguments are not positional. They are identified by the one-character ID that immediately precedes each argument. Each network (or DAP) defines its own IDs. Therefore, an F might mean one thing when used in association with the IBM Global Network VAN and mean something else when used in association with the GEIS VAN.

- KEY (queuekeyid)

The Queuekeyid specifies what local Gateway mailbox to use and, therefore, only applies to QUE jobs. Gateway mailboxes are actually MVS sequential data sets. Their names follow a definite convention. An important part of the data set name is the send ID and receive ID. These IDs come from the Gateway's Trading Partner Database. The queuekeyid typically serves as a key into the Trading Partner Database to obtain this information.

When data is sent to a local Gateway mailbox, the queuekeyid identifies to whom the data is intended. That database record is retrieved and its send ID becomes part of the data set name where the data is stored. Likewise, when data is to be received from a local Gateway mailbox, the queuekeyid typically identifies the trading partner from whom data is to be received. In this case, that database record is retrieved and its receive ID is used to identify those data sets that will be passed back to the requestor.

Typically, the queuekeyid is the Gateway's trading partner user ID suffixed with the test/production indicator (or data type) from the Gateway Trading Partner Database. A valid combination must exist. For more information on other queuekeyid formats, see the *Harbinger Gateway User Guide*. The Gateway considers the test/production indicator part of the key to its Trading Partner Database. Therefore, two separate entries must exist in the Gateway Trading Partner Database if both test and production data is to be exchanged with the same trading partner.

DataInterchange Provided Parameters for the Gateway

DataInterchange builds the correct SEND DS or RECEIVE DS parameters. The user should not provide these parameters in the EDINTCMD PDS members. For more information about how DataInterchange determines these names, see "TIGD31 Transaction Data Queues (Outbound files)" on page 6-23, "TIGV31 Transaction Data Queues (Outbound files)" on page 6-24, and "TIGD31 And TIGV31 Receive File Names (Inbound files)" on page 6-25.

This is an example of these parameters:

```
SEND DS(physical.file.name) disp  
RECEIVE DS(physical.file.name) disp
```

More About Gateway Send and Receive Files

DataInterchange may interface with the Gateway either through the Transaction Store Facility or batch DI Utility JCL. The EDI CLIST allocates the following data sets for the Gateway support:

```
//TIGWYQ DD DSN=&SYSUID..EDI.TIGWYQ,DISP=SHR  
//SYSIN DD DSN=&SYSUID..EDI.SYSIN,DISP=SHR  
//EDINTCMD DD DSN=EDI.TIGATE.EDINTCMD,DISP=SHR
```

If the Transaction Store Facility is going to be used to send and receive data through the Gateway, any new transaction data queue names and receive file names (other than TIGWYQ) must be added to the EDI CLIST. Send and receive files can only be allocated DISP=SHR in the job that invokes the Gateway. This is true whether the allocation is in the EDI CLIST or in batch DataInterchange Utility JCL. These files should not be deleted or created in the same CLIST or JCL that invokes the Gateway. DataInterchange always defines these files in SYSIN with DISP=SHR.

Validation of Gateway Parameters

DataInterchange does not attempt to validate Gateway parameters. This approach provides maximum user control and flexibility. When Gateway requirements and capabilities change (such as support for additional VANs), the DataInterchange user is not dependent on DataInterchange to make corresponding changes to accommodate the modified Gateway.

Substitutable Tags

Although the user controls much of the content of the information passed to the Gateway directly, it may be convenient to use some of the information already available in DataInterchange. A set of substitutable variable tags is provided to allow the user to tell DataInterchange to pull some of the information needed by the Gateway from DataInterchange profiles, control blocks, and so on, at runtime.

You must start each tag with an ampersand (&). In addition, following the ampersand, you should use a two-character identifier (from the list below) and then the rest of the tag. All tags must be in uppercase.

"C." retrieve a value from the communication control block
"N." retrieve a value from the network profile
"R." retrieve a value from the requestor profile
"T." retrieve a value from the trading partner profile

Step 4. Create DataInterchange Utility JCL

Your DataInterchange Utility JCL has to be modified only slightly to use the Gateway. If you followed the above steps, you encountered reference to the necessary changes. Take note of the EDINTCMD, SYSIN, EDISYSIN, and TIGWYQ ddnames. The dataset associated with ddname EDINTCMD should be the PDS containing your network commands files. For more information about EDINTCMD, see “Step 3. Create EDINTCMD Partition Dataset Members” on page 6-26. DataInterchange copies (after resolving substitutable tags) the contents of the appropriate PDS member into the SYSIN dataset. Also, DataInterchange adds the SEND DS or RECEIVE DS parameter. It is the SYSIN data set that is passed to the Gateway. EDISYSIN can then be used as input to the DataInterchange Utility.

TIGWYQ is the default outbound file ddname on SEND type commands using the TIGD31 network ID and is always the default outbound file ddname using the TIGV31 network ID. For more information about outbound data sets, see “TIGD31 Transaction Data Queues (Outbound files)” on page 6-23 and “TIGV31 Transaction Data Queues (Outbound files)” on page 6-24. For inbound jobs, attention should be made to the inbound file ddname and file disposition. For more information about inbound data sets, see “TIGD31 And TIGV31 Receive File Names (Inbound files)” on page 6-25.

Table 6-9 (Page 1 of 2). Network Command Tags for Fields of the Network profile

Network Command Tag	Utility Keyword Name	Panel Field Name	Description
CMDINDD		Network input file	Network pgm cmd input file name
CMDOUTDD		Network output file	Network pgm cmd output file name
CMDRECLN		Input record length	Network Command record length
COMMRTN		Communication rtn	Communication Routine name
MSGHNDLR		Message handler	Program to process messages
MSGTXTH		Message text header	Message text header character
NETDIAL		Dial connect num	Dial connection phone number
NETID	netid	Network ID	Network Identification (key)
NETNAME	netname	Network name	Network Name
NETPARM		Network parameters	Network program parameters
NETPROG		Network program	Network send/receive program name
NETSEQ		Network sequence	Sequence number for network
NETSTAT		Net status file	File for network status messages
QFILEDD	fileid or funackfile	Trans data queue	Transaction data queue file name
QRECLN		Trans rec length	Transaction data record length
SYSLVL		System level	System Level
SYSTYPE		System type	System Type

Table 6-9 (Page 2 of 2). Network Command Tags for Fields of the Network profile

Network Command Tag	Utility Keyword Name	Panel Field Name	Description
TIMEZONE		Time zone	Time Zone

Notes:

- All Network command tags must be preceded with &N.
- Utility keyword name and Panel field name are provided for reference only.
- Most Network command tags match Utility keyword names where a Utility keyword name exists.

Table 6-10. Network Command Tags for Fields of the Requestor profile

Network Command Tag	Utility Keyword Name	Panel Field Name	Description
ACCTID	acctid	Account number	Network Account Number
ALTDIAL		Alt net dial num	Alternate number to dial to connect to the network
MSGCLS		Message user class	Network Message User Class
NETACK		Net acknowledgment	Network Acknowledgment code
NETCHG		Network charges	Network Charges Code
DDNETCMD		Network cmds file	Commands for the network from the user
NETCLS		Network msg class	Network Classification
NETEDIO		EDI receive option	EDI processing option
NETEDIP		EDI proc override	EDI processing override
NETID	netid	Network ID	Network Identification
NETPASS		Password	Network Password (current & new)
NETRETN		Retention period	Message retention
NETVCHK		Destination verif	Validate destination
RCVFILE	fileid	Receive file name	DD name for transaction receive
REMODEMH		Remote status pgm	Program to process status messages from remote network
REQID	reqid	Requestor ID	Requestor ID (key)
STGFMT		Storage format	Storage Format
STGFMTOV		Format override	Storage Format override
TIMEOUT		Command line timeout	Command line timeout
USERID	userid	User ID	Network User ID

Notes:

- All Network command tags must be preceded with &R.
- Utility keyword name and Panel field name are provided for reference only.
- Most Network command tags match Utility keyword names where a Utility keyword name exists.

Table 6-11 (Page 1 of 2). Network Command Tags for Fields of the Trading Partner profile

Network Command Tag	Utility Keyword Name	Panel Field Name	Description
ACCTID	acctid	Account number	Trading partner net acct number
ADDRLN1	addrln1	Address line 1	Company address line 1
ADDRLN2	addrln2	Address line 2	Company address line 2
CMMTLN1	cmmtn1	Comment line 1	Comment line 1
CMMTLN2	cmmtn2	Comment line 2	Comment line 2
COMPYNM	cmpynm	Company name	Company name
CNTCTNM	cntctnm	Contact name	Contact name
CNTCTPH	cntctph	Contact phone	Contact phone number
DDNETCMD		Network cmds file	Commands for the network from the user
DECNOT		Decimal notation	Decimal Notation
DEDLN		Data element delim	Trnsctn Data Element Delimiter
EOTID		End of text msg	End of Text/Message Delimiter
FNCGRP		Functional group	Functional group wanted Y or N

Table 6-11 (Page 2 of 2). Network Command Tags for Fields of the Trading Partner profile

Network Command Tag	Utility Keyword Name	Panel Field Name	Description
GRPCTLNO	grpctlno	Group control	Functional Group control number
INTCTLNO	intctlno	Interchange control	Interchange control number
INTID	intid	Interchange ID	Interchange ID
INTQUAL		Interchange qualif	Interchange qualifier
LOGENV		Log standard data	Log envelope data Y or N
MACHTYPE		Machine type	Machine type
NETACK		Net acknowledgment	Network Acknowledgment code
NETCHG		Network charges	Network Charges Code
NETCLS		Network msg class	Network Classification
NETEDIO		EDI receive option	EDI processing option
NETEDIP		EDI proc override	EDI processing override
NETID	netid	Network ID	Network Identification
NETRETN		Retention period	Message retention
NETVCHK		Destination verif	Validate destination
RCVPASS		Interchange recv PW	Password for receiving
RLSCHAR		Release character	Release Character
SECPROF		Security ID	Security profile member
SEDLM		Subelement delimiter	Trnsctn Sub Element Delimeter
SEGDLM		Segment delimiter	Trnsctn Data Segment Delimeter
SEGSEP		Seg ID separator	Trnsctn segment separator
SNDPASS		Interchange send PW	Password for sending
STGFMT		Storage format	Storage Format
STGFMTOV		Format override	Storage Format Override
SYSID		System ID	System ID
SYSQUAL		System qualifier	System qualifier
TIMEOUT		Command line timeout	Command line timeout
TPDIAL		TP data phone num	Data (not voice) phone number
TPNICKN	tpnickn	TP nickname	Trading partner nickname (key)
TRXCTLNO	trxctlno	Transaction contrl	Transaction Set control number
USERID	userid	User ID	Trading partner network user ID

Notes:

- All Network command tags must be preceded with &T.
- Utility keyword name and Panel field name are provided for reference only.
- Most Network command tags match Utility keyword names where a Utility keyword name exists.

Table 6-12 (Page 1 of 2). Network Command Tags for Fields of the Common control block

Network Command Tag	Utility Keyword Name	Panel Field Name	Description
ACCTYP			Account type for snd/rcv/can
ACKIND			Acknowledgment request coden
ADMTYPE			Admin response file type
BLKLEN			Length (bytes) of this block
BLKNME			Dump eyecatcher 'EDICMCB'
BLKTYPE			= 'H' if HUGE blocks
CANED			Cancellation end date
CANET			Cancellation end time
CANSN			Cancellation start date
CANST			Cancellation start time
CLRFILE			Clear file after send

Table 6-12 (Page 2 of 2). Network Command Tags for Fields of the Common control block

Network Command Tag	Utility Keyword Name	Panel Field Name	Description
CONTRCV			Continuous receive flag
DATAFMT			Format of data being sent
DATATYP			Data type for send/receive
DCIND			Priority delivery class code
DDCOLON			DD: string used for Base/MVS
ENAME			Envelope name
FACCTSEP			Value to separate acct user ID
FAK			Y specifies if the network ack supported
FCANCEL			Y specifies if CANCEL is OK
FCLASS			Y specifies if the msg class supported
FEDIE			Y specifies if the EDI UNB UNZ files OK
FEDIG			Y specifies if the EDI GS GE files OK
FEDII			Y specifies if the EDI ICS ICE files OK
FEDIT			Y specifies if the EDI STX END files OK
FEDIU			Y specifies if the EDI BG/EG files OK
FEDIX			Y specifies if the EDI X12/ISA files OK
FFILE			Y specifies if the free form files OK
FILENAME			Internal file or dd name
FILERCV			File was received (Y or N)
FMSG			Y specifies if the free form messages OK
FNOUSERID			Y specifies if the account number OK
FQUEUED			Y specifies if the queued functions OK
FRCVBTP			Y specifies if the receive by trading partner
FRESTART			Y specifies if the restart supported
FSYSMSG			Y specifies if the system msgs are supported
FTYPE			File type (TS or PG)
MODEM			Modem type
MSGNAME			Message name identification
NETID			Network identification
NETOP			Network operation
NPERRCD			Error code
NPESCDE			End session response code
NPSEVER			Error code severity
NPSSCDE			Start session response code
RECVTYP			Type of receive
REQID			Requestor identification
RESRECL			Resolution of rec len disparity
SEQNUM			msg/file/trans. seq. number
TIMEZONE			Time zone for time interval
TPNICKN			Trading partner identification
UNIQID			Unique ID returned by EXP

Notes:

- All Network command tags must be preceded with &C.
- Utility keyword name and Panel field name are provided for reference only.
- Most Network command tags match Utility keyword names where a Utility keyword name exists.

Interfacing DataInterchange for CICS with SDM LinkPlus Interactive

The Software Development and Maintenance (SDM) LinkPlus** Interactive product is a CICS-based file delivery system used to manage the exchange of data between trading partners. This product offers features such as flexible connectivity, mailboxing, dial-out, and dial-in security. DataInterchange can be set up to interface with LinkPlus. The interface currently in effect involves using VANICICS. For more information, see “Special Communication Routine for CICS” on page 6-14 and “Invoking the Continuous Receive Interface” on page 6-19.

Outbound Processing Considerations

LinkPlus uses scripts to know how to process requests. These scripts are part of the LinkPlus product and are not part of DataInterchange. If a script name is known and can be associated with a DataInterchange network profile member, that name can be entered in the Script field of the network profile. This script name will then be passed to LinkPlus. It is optional and can be overridden by using the SCRIPT keyword on PERFORM SEND type commands. Network acknowledgment processing is not currently supported through this interface. To use the LinkPlus interface, create a network profile member specifying VANICICS as the Communication routine name, LKP7560 as the Network program name, and EDIVNOP as the Message handler name. Requestor and Trading Partner profile members would then have to indicate the network profile ID created above. LinkPlus will return information to DataInterchange through the COMMAREA. For more information, see “Special Communication Routine for CICS” on page 6-14. If a return code other than HI000 is returned, DataInterchange will log a VN1022 error message. If this occurs, consult the LinkPlus error log.

Inbound Processing Considerations

LinkPlus will CICS LINK to the DataInterchange program EDICRIN. For more information, see “Invoking the Continuous Receive Interface” on page 6-19. Within this COMMAREA is the name of a continuous receive profile member that will dictate processing; for example, DEENVELOPE or DEENVELOPE AND TRANSLATE. These continuous receive profile members should specify N in the Allow syncpoints field and should specify the name of a response program in the Response name field. The EDICRIN commarea USERAREA value will be moved into the Utility Control Block field USRFLD, where it will be available to the response program. The response program will conditionally do two things. If it receives a zero return code from DataInterchange (indicating a successful translation), the response program will start LinkPlus so that the SDM repository records can be marked as extracted. It will also issue an EXEC CICS SYNCPOINT. If a nonzero return code is passed to the response program, it is the responsibility of the response program to issue an EXEC CICS SYNCPOINT ROLLBACK. It will always be the responsibility of the response program to delete the data TS queues.

SAP Status Tracking in DataInterchange

- | SAP is a client/server application which supports business processes that include sales, materials management, and distribution. SAP was developed in Germany and supports an interface to an EDI subsystem. SAP has both mainframe and UNIX solutions.
- | SAP generates application data in the SAP Intermittent Document (IDOC) layout. The file is sent to the EDI subsystem (or translator) using a file transfer product such as File Transfer Protocol (FTP) or Transmission Control Protocol/Internet Protocol (TCP/IP) from UNIX to mainframe.

The mapping required to perform the translation of the inbound and outbound IDOCs is the user's responsibility. To assist in mapping the IDOC, a new mapping literal keyword called &THANDLE is

provided to enable mapping of the DataInterchange archive key to the SAP IDOC for inbound processing. A new special DataInterchange variable name called DISAPSEQ allows saving of the IDOC record sequence number on the first error during outbound processing. The variable DISAPSEQ will be captured in the SAP status record to indicate the first record in error.

DataInterchange provides the capability to capture the SAP status information during different phases of the EDI process by specifying a new keyword, SAPUPDT, on the utility PERFORM commands. The SAP status tracking is only supported with the DataInterchange Utility.

A utility PERFORM command allows you to extract or remove the SAP status records from the database based on selection criteria, and write them (in SAP EDI_DS record format) to a sequential file for transfer to the SAP system. DataInterchange currently supports the SAP status EDI_DS record at IDOC releases 2, 3, and 4.

Outbound Processing

The outbound processing of the SAP status information is under the user control through the *SAPUPDT()* keyword on the utility perform statements. However, if the *SAPUPDT(Y)* keyword is specified, the SAP status along with the DataInterchange archive key is stored in a new database called EDIVSSTK. The status is then updated at various key points during EDI processing.

The *SAPUPDT(Y)* keyword is required on the following DataInterchange Utility PERFORM commands:

- TRANSLATE TO STANDARD
- TRANSLATE AND ENVELOPE
- TRANSLATE AND SEND
- ENVELOPE
- REENVELOPE
- ENVELOPE AND SEND
- REENVELOPE AND SEND
- SEND

Inbound Processing

The inbound processing of the SAP status information for functional acknowledgments is under the user control through the *SAPUPDT()* keyword on the utility perform statements. However, if the *SAPUPDT(Y)* keyword is specified, the functional acknowledgment status is collected in the database. The SAP status can be collected only when the Transaction Store is active.

The *SAPUPDT(Y)* keyword is required on the following DataInterchange Utility PERFORM commands:

- DEENVELOPE
- DEENVELOPE AND TRANSLATE

Extracting SAP Status

To extract SAP status records from the DataInterchange database and write them to the file specified in the keywords *OUTFILE* and *OUTTYPE*, enter:

```
PERFORM SAP STATUS EXTRACT  
WHERE OUTFILE() OUTTYPE() CLIENT() SAPSTAT() TO ( ) DAYS() TO()
```

Where:

OUTFILE

Specifies the file name to which the extracted SAP status records are written. The default is SAPOUT. The output file should be defined with a record length that is at least as large as the SAP Status EDI_DS record.

OUTTYPE

Specifies the file type, for example TD, TS, TM, or VE. The default is TD. For CICS environment only.

CLIENT

Specifies the extract records by the client ID. The default is all.

SAPSTAT

Specifies the SAP status value to extract or uses the TO() to obtain a range of values to extract. The acceptable values are 04 - 22. The default is all status.

DAYS

Specifies the date of the records to extract or uses the TO() to obtain a range of values to extract. The range is inclusive. The default is all days.

Return Code

The return codes for extracting SAP status from the database are:

Return Code	Description
792	No records met the selection criteria or the records were truncated in the output file.
796	Error occurred during processing.

Removing SAP Status From the Database

To remove the SAP status from the database, enter:

```
PERFORM SAP STATUS REMOVE  
WHERE CLIENT() PRIORTO() SAPSTAT() TO()
```

Where:

CLIENT

Specifies to remove records by the client ID. The default is all.

SAPSTAT

Specifies the SAP status value to remove or uses the TO() to obtain a range of values to remove. The acceptable values are 04 - 22. The default is all status.

PRIORTO

Specifies the date prior to when the records are to be removed. The default is all dates.

Return Codes

The return code for removing SAP status from the database is:

Return Code	Description
796	Error occurred during processing.

SAP Status Codes Supported by DataInterchange

The SAP status codes supported by DataInterchange are:

Table 6-13. SAP Status Codes Supported by DataInterchange

Status Code	Description
04	Error within control information of EDI subsystem
05	Error during translation process
06	Translation successful
09	Error during interchange handling
10	Interchange handling successful
11	Error during dispatch
12	Dispatch successful
16	Functional Acknowledgment positive
17	Functional Acknowledgment negative
22	Dispatch successful, acknowledgment still due

The mapping required to perform the translation of the inbound and outbound IDOCs is a user responsibility. To assist in mapping the IDOC, a new mapping literal keyword *&THANDLE* is provided to enable mapping of the DataInterchange archive key to the SAP IDOC for inbound processing, and a new special DataInterchange variable name DISAPSEQ is provided to allow saving of the IDOC record sequence number on the first error during outbound processing. The variable DISAPSEQ is collected in the SAP status record to indicate the first record in error.

Interfacing DataInterchange with MQSeries

You can exchange data with your trading partners using MQSeries queues in network profiles. DataInterchange prohibits the use of MQSeries queues as the target of the envelope process or as input to the deenvelope process. MQSeries queues can be used as part of a logical network where enveloped data is sent to and received from MQSeries queues. DataInterchange provides a sample network profile member named **MQSAMP** to assist you with the setup of this local network, as described in the following steps:

1. Have your MQSeries Administrator define the MQSeries queue you plan on using.
2. Define DataInterchange MQSeries Queue profile members on behalf of the queues your MQSeries administrator has created. Refer to Chapter 4 of the *DataInterchange Administrator's Guide* for more information.
3. In the View Profile Member Panel (PM11), complete the fields as follows. If you need additional information about the use of a specified field, place the cursor in that field and press F1 (Help).

In This Field:	Enter:
Network ID	A unique name to identify the network, such as MQSAMP. This value is referenced by the requestor profile and the trading partner profile. Use the same ID throughout DataInterchange to refer to this network.
Network name	A descriptive name of the network, such as Sample MQSeries Network. This field is optional.

In This Field:	Enter:
Communication rtn	VANIMQ, which is the name of the communication routine that builds network commands and invokes the network's send and receive program to process the commands.
Network program	EDIMQSR, which is the physical name of the send and receive program. This program is invoked by the communication routine to process requests.
Network parameters	The parameters required by the network program. Change the Network parameters field to match the DataInterchange MQSeries Queue profile member names created in step 2. The SENDMQ= keyword should be followed by the DataInterchange MQSeries Queue profile member name where you send data. The RECEIVEMQ= keyword should be followed by the DataInterchange MQSeries Queue profile member name where you receive data. The two parameters must be blank space delimited. If you are performing one-way communications with your trading partner, only the keyword-value combination of that direction is required.
Network input file	The data definition name (ddname) that contains the commands written for the network program to process. For IINB41, the name is INMSG. This field is not used for point-to-point connections or CICS.
Input rec length	The length of records in the network input file. For the IBM Global Network, the record length is 80. For a point-to-point connection or CICS, leave this field blank.
Trans data queue	The ddname of the file that will hold the enveloped transactions waiting to be sent to your trading partner. This file is also used when queuing or sending transactions. If you leave this field blank, the default is QDATA.
Trans rec length	The length of records in the transaction data queue. The communication routine provided by DataInterchange (VANIINB41) ignores this field and uses the logical record length you allocated for the file. For DataInterchange for CICS, the maximum usable record length that can be used in a Temporary Storage Queue (TSQ) is 28000. To utilize all 28000 bytes in each record, a value of zero or blank should be entered in the TRANS REC LENGTH field. Otherwise, the maximum number that can be entered in this 4-character field is '9999'.
Time zone	The code for your location's time zone. The network specifies the allowable codes. See the <i>DataInterchange Administrator's Guide</i> for the list of codes allowed by IBM Global Network.

In This Field:	Enter:																																												
System type	This field is optional. If used, the code describes the software you use to interface with IN. Valid values are:																																												
	<table> <tr> <th>Code</th><th>Description</th></tr> <tr><td>01</td><td>Unknown system type</td></tr> <tr><td>10</td><td>Expedite/PC</td></tr> <tr><td>11</td><td>Expedite Base/2</td></tr> <tr><td>12</td><td>Expedite Base/AIX</td></tr> <tr><td>14</td><td>Expedite Base for SCO UNIX</td></tr> <tr><td>15</td><td>Expedite Base/DOS</td></tr> <tr><td>16</td><td>Expedite Base for SCO XENIX</td></tr> <tr><td>17</td><td>Expedite Base for Windows</td></tr> <tr><td>19</td><td>Expedite for Windows</td></tr> <tr><td>20</td><td>Expedite/MVS Host</td></tr> <tr><td>21</td><td>Expedite Base/MVS</td></tr> <tr><td>22</td><td>TCP/IP FTP Gateway</td></tr> <tr><td>30</td><td>Mail Exchange</td></tr> <tr><td>31</td><td>Expedite Base/VM</td></tr> <tr><td>33</td><td>X.400 Gateway</td></tr> <tr><td>40</td><td>Expedite/Direct</td></tr> <tr><td>44</td><td>EDI VAN Interconnect</td></tr> <tr><td>71</td><td>Expedite Base/400</td></tr> <tr><td>80</td><td>Expedite/CICS</td></tr> <tr><td>90</td><td>Information Exchange Administration Services</td></tr> <tr><td>91</td><td>Expedite ASYNC</td></tr> </table>	Code	Description	01	Unknown system type	10	Expedite/PC	11	Expedite Base/2	12	Expedite Base/AIX	14	Expedite Base for SCO UNIX	15	Expedite Base/DOS	16	Expedite Base for SCO XENIX	17	Expedite Base for Windows	19	Expedite for Windows	20	Expedite/MVS Host	21	Expedite Base/MVS	22	TCP/IP FTP Gateway	30	Mail Exchange	31	Expedite Base/VM	33	X.400 Gateway	40	Expedite/Direct	44	EDI VAN Interconnect	71	Expedite Base/400	80	Expedite/CICS	90	Information Exchange Administration Services	91	Expedite ASYNC
Code	Description																																												
01	Unknown system type																																												
10	Expedite/PC																																												
11	Expedite Base/2																																												
12	Expedite Base/AIX																																												
14	Expedite Base for SCO UNIX																																												
15	Expedite Base/DOS																																												
16	Expedite Base for SCO XENIX																																												
17	Expedite Base for Windows																																												
19	Expedite for Windows																																												
20	Expedite/MVS Host																																												
21	Expedite Base/MVS																																												
22	TCP/IP FTP Gateway																																												
30	Mail Exchange																																												
31	Expedite Base/VM																																												
33	X.400 Gateway																																												
40	Expedite/Direct																																												
44	EDI VAN Interconnect																																												
71	Expedite Base/400																																												
80	Expedite/CICS																																												
90	Information Exchange Administration Services																																												
91	Expedite ASYNC																																												
System level	This field is optional. If used, it represents a value that indicates to your trading partner the level of your system, for example, B41. If you do not know what to type in this field, leave it blank.																																												
Msg text header	The character that indicates the beginning of text for a message. For the IBM Global Network, the character is T.																																												
Net output file	The logical name of the file containing the network's responses to the command input file. For example, for IINB41 the name is OUTMSG. For a point-to-point connection, leave this field blank. This field is not used in DataInterchange for CICS.																																												
Message handler	The name of the program that processes messages and network acknowledgments from the network. When you send a transaction, if your network returns a message to indicate the success or failure of your send request, the message handler program interprets the returned message and updates the status of the interchange. For example, for IINB1, IINB41, and IINB42, the name is INB1MSG; for IINR3, the name is INMSGHL; and for GEIS, the name is GEMSGHL. For a point-to-point connection, type NONE .																																												
Network sequence	A number that DataInterchange increments and assigns to all outbound documents. You can set or reset the value at which sequential numbering begins. 00000 is the default.																																												
Net acks file	The name of a file (MVS ddname) where you would like the network to write network acknowledgments when you request a status update. The network acknowledgments are read and evaluated by the message handler program.																																												
	Note: This feature will be enabled in a future release of DataInterchange.																																												

In This Field:	Enter:
Dial connect num	The phone number to dial to connect to your network.
Script name	Optional. The name of a set of instructions that the communication software can use to process requests associated with this network profile member. The set of instructions would be part of the communication software package and not part of DataInterchange. See “Interfacing DataInterchange for CICS with SDM LinkPlus Interactive” on page 6-34.
	Note: You can override the script name in the utility commands for sending transactions.

4. Your new network profile member can be used within trading partner profiles and requestor profiles. Please refer to documentation on those profiles for more information.
5. If you are using continuous receive, see “Continuous Receive Using MQSeries” on page 5-23 for more information. You may also perform the following tasks, depending on how you are using MQSeries queue:
 - To use MQSeries queues with an application file, you must identify the associated MQSeries Queue profile member in the application data format (specifically, the **Application file name** and **Application file type** fields).
 - When executing a translation, you can read data to or write data from MQSeries queues (for print, report, exception, and so on).
 - To use an MQSeries queue for your application data file, you must use the specific MQSeries Queue profile member name and the MQ type in at least one of the following places:
 - Application file name and file type in the application data format
 - Application file name and file type in the Receive Usage panel
 - Any PERFORM command that uses the APPFILE and APPTYPE keywords

Appendixes

Appendix A. DataInterchange Control Blocks	A-1
Service Name Block (SNB)	A-1
Common Control Block (CCB)	A-2
Function Control Block (FCB)	A-4
Translator Control Block (TRCB)	A-6
Translator Input Data Block (TRIDB)	A-29
Translator Output Data Block (TRODB)	A-31
Communication Interface Control Block (CMCB)	A-34
Trading Partner Profile Block (TPPDB)	A-42
Communication Data Block (DATBLK)	A-52
Network Profile Block (NPDB)	A-53
Requestor Profile Block (REQDB)	A-56
 Appendix B. DataInterchange Utility Condition Codes and API Return Codes	 B-1
Return Codes	B-1
DataInterchange Utility Condition Codes	B-2
Translation Condition Codes	B-4
Communications Condition Codes	B-14
Combination Command Condition Codes	B-16
Application Programming Interface Return Codes	B-17
Communication Return Codes	B-27
 Appendix C. Copy Books and Include Files	 C-1
Copy Books for COBOL	C-2
Include Files for PL/I	C-13
Include Files for C	C-25
Copy Books for ASSEMBLER	C-36
 Appendix D. Sample Programs	 D-1
Creating Tagged Import Files from Fixed Flat Files	D-1
Initializing and Terminating DataInterchange	D-1
Query the Transaction Store using COBOL	D-9
Query the Transaction Store in PL/I	D-11
Translate and Queue for Send	D-12
Send Queued Data	D-15
End Translation	D-18
Receive From the Network	D-19
Translate Received Data	D-22
End DataInterchange	D-26
Data Extract Report Generator (EDISAMR1)	D-27
Data Extract Report Generator (EDISAMS1)	D-35
Network Activity Report Generator (EDISAMT1)	D-44
Initializing, Invoking, and Terminating HOT-DI	D-49
Invoking Response Programs	D-56
Field Exit Program	D-57
Test for Filter Type	D-65
Filtration Exit Examples	D-68
Authentication Examples	D-85
Encryption Examples	D-95
Get Envelope Service Example	D-117

Put Envelope Service Example	D-118
Inbound Envelope Program Example	D-120
Outbound Envelope Program Example	D-121
VANICICS Network Program Example	D-122
Appendix E. Using Sample JCL	E-1
DataInterchange Utility (EDIUTILV) JCL	E-1
Utility Data Sets Required	E-17
Archive VSAM event log entries (EDIELARV)	E-18
Archive DB2 event log entries (EDIELARD)	E-20
Appendix F. Space Calculation Examples	F-1
Space Requirements for DataInterchange Tables and Files	F-1
DataInterchange Allocation Tables	F-24
Space Calculation Scenario	F-29
Worksheets	F-37
Appendix G. Performance Considerations	G-1
DataInterchange Optimization	G-1

Appendix A. DataInterchange Control Blocks

This Appendix provides DataInterchange control block information.

Service Name Block (SNB)

In DataInterchange architecture, the service name block (SNB) enables all services to be accessed by using a logical name. At the time of execution, the association is made between a logical name and a physical load module name. Services provided by DataInterchange use predefined names (see the **ZSNBNAME** field description). The association of these names and load modules is accomplished with a static table within DataInterchange. The same interface used to access DataInterchange API services is used when DataInterchange invokes a user-written exit program. User exits are defined and given logical names during the customization and mapping process. The ADAMCTL profile is then updated so that the physical load module and the implementation language can be associated with the logical name. ADAMCTL profile entries are automatically added to the service table at execution time as user exits are requested.

Using the same SNB when requesting the same service improves performance. Once the SNB has been used and the physical and logical association made, this association is saved in the **ZSNBNDX** field, eliminating the table search on the next request. The SNB is modified by DataInterchange. If you want a reentrant program, the storage for the SNB must not come from static storage.

Table A-1 shows the layout of the SNB.

Table A-1. SNB Definition. Layout of the SNB and Descriptions of Fields

Name	Type	Offset	Length	Description
ZSNBLL	Binary	0	2	SNB length
ZSNBID	Binary	2	2	Reserved
ZSNBEYE	Character	4	8	Dump eye catcher
ZSNBNAME	Character	12	8	Service name
ZSNBNDX	Binary	20	4	Service index
ZSNBPC	Binary	24	2	Parameter count
ZSNBFLG0	Character	26	1	First flag byte
ZSNBFLG1	Character	27	1	Second flag byte
ZSNBFANC	Binary	28	4	First anchor pointer

SNB Field Descriptions

The following list describes the SNB fields.

- ZSNBLL** Length of SNB. A 2-byte binary field that contains the length of the SNB control block. An SNB is 32 bytes.
- ZSNBID** Reserved. This field is not currently used.
- ZSNBEYE** Eyecatcher. The first time an SNB is used, DataInterchange initializes this field with a value of ****ZSNB****. When you look at virtual storage dumps, this field helps identify those sections of storage that contain an SNB.

ZSNBNAME Service name. This field indicates to DataInterchange what service is being requested. Each service within DataInterchange is assigned a logical name that must be placed in this field. DataInterchange has an internal table that associates a logical name with a physical load module that processes the request. User exits are given logical names at various points in the customization process. The association of a user-defined logical name with a physical load module name is done by the ADAMCTL profile. The service names for the API services provided by DataInterchange are:

Name	Description
ENVSERV	Environmental services
TRANPROC	Translation services
TRANPROC	Enveloping services
TRANPROC	Data extraction services
COMM	Communications services
TRANSSRV	Update status services
SYNCSERV	SYNCPPOINT services

The service name must be left-justified and padded with blanks within the **ZSNBNAME** field.

ZSNBNDX Service index. This field is used internally by DataInterchange to record the offset into the internal table that defines the services. The first time an SNB is used, DataInterchange searches through an internal table that associates a logical name (**ZSNBNAME**) with a physical load module that processes the request. This search is eliminated on subsequent calls that use the same SNB, because once the correct entry is found, the index of that entry is saved in the **ZSNBNDX** field. Making repeated calls for the same service and then using the same SNB for all calls to that service improves performance slightly.

ZSNBPC Parameter count. The service being requested is identified in the **ZSNBNAME** field. The function within that service is indicated by the **ZFCBFUNC** field of the function control block (FCB). Not all functions within the service require the same number of parameters. The **ZSNBPC** field must be set to indicate the number of parameters that are being provided in the FXXZC, FXXZCBL, FXXZPLI or FXXZASM call. Failure to set the correct parameter count yields unpredictable results.

- | **ZSNBFLG0** First flag byte. For DataInterchange use only.
- | **ZSNBFLG1** Second flag byte. For DataInterchange use only.
- | **ZSNBFANC** First anchor pointer. For DataInterchange use only.

Common Control Block (CCB)

DataInterchange uses the CCB to maintain status information about the current DataInterchange session. A DataInterchange session is anything that happens between an initialization service request and a terminate service request. The CCB used on the initialization request must be the same CCB used for all future requests. This CCB is provided to all DataInterchange programs and to all user-defined exit programs that are invoked during the session.

Most of the fields in this control block are for use by DataInterchange. However, the return code (ZCCBRC) and extended return code (ZCCBERC) are used to communicate to the calling program the results of the request that was just made. The CCB must be at least 608 bytes, the length DataInterchange requires to maintain status. However, the length of the CCB can exceed 608 bytes if data that an application program wants to provide to a user-defined exit program is invoked in processing an application request. Anything defined beyond the **ZCCBRV** field is for application use only and is not

altered by DataInterchange. The CCB is modified during execution. If you want a reentrant program, the storage for the CCB must not come from a static storage area. Table A-2 on page A-3 shows layout of the CCB.

Table A-2. CCB Definition. Layout of the CCB and Descriptions of Fields

Name	Type	Offset	Length	Description
ZCCBLL	Binary	0	2	CCB length
ZCCBID	Binary	2	2	Reserved
ZCCBEYE	Character	4	8	Dump eye catcher
ZCCBRC	Binary	12	4	Return code
ZCCBERC	Binary	16	4	Extended return code
ZCCBSID	Character	20	8	System ID
ZCCBUID	Character	28	8	User ID
ZCCBAID	Character	36	8	Application ID
ZCCBCID	Character	44	8	Error module ID
ZCCBXFID	Binary	52	2	Error function ID
ZCCBLPID	Character	54	6	Language profile ID
ZCCBCPID	Binary	60	4	Code page ID
ZCCBRSV	Binary	54	4	Reserved for DataInterchange
ZCCBCCXP	Binary	68	4	Pointer to CCB extension
ZCCBCABP	Binary	72	4	Pointer to common area block
ZCCBDBID	Character	76	4	MVS DB2 subsystem ID
ZCCBDBPL	Character	80	8	MVS DB2 plan / AIX DB2 alias
ZCCBDBUI	Character	88	8	AIX DB2 user ID
ZCCBDBPW	Character	96	18	AIX DB2 password
ZCCBRSV1	Character	114	26	Reserved for DataInterchange
ZCCBRSV2	Binary	140	468	Reserved for DataInterchange

CCB Field Descriptions

The following list describes the CCB fields.

ZCCBLL	Length of CCB. A 2-byte binary field that contains the length of the CCB control block. A CCB is 608 bytes. The 608 bytes of storage must be allocated for this block.
ZCCBID	This field is not currently used.
ZCCBEYE	Eyecatcher that is useful for identifying this control block when looking at a storage dump. During the DataInterchange initialization call, DataInterchange initializes this field with the text **ZCCB** . The calling program is not affected by this field.
ZCCBRC	Return code. A signed integer that indicates the result of your last request. A value of 0 indicates successful completion. A negative value indicates an error getting to the service you have requested. A positive value indicates an error returned from the service requested. For more information on the explanation of the return codes for the various services, see Appendix B, "DataInterchange Utility Condition Codes and API Return Codes."

ZCCBERC	Extended return code. A signed integer that further defines the results of your request. The return code field usually indicates the severity of the error, and the extended return code specifies exactly what error occurred. For more information on the return codes and extended return codes that can be returned by DataInterchange API services, see Appendix B, “DataInterchange Utility Condition Codes and API Return Codes.”
ZCCBSID	System ID. DataInterchange provides the system ID field during the initialization service API request. The system ID has a static value of EDIV00.
ZCCBUID	User ID. DataInterchange activates the user ID field during the initialization service API request. For a TSO user, this is the TSO user ID. For an MVS batch job, this is the user ID provided in the //JOB card. For a CICS system, this is the CICS signon user ID, the terminal ID, or the application ID of the CICS region.
ZCCBAID	Application ID. The application ID field is moved into the CCB during the initialization service API request and is one of the parameters that must be supplied during the initialization request. The application ID identifies the APPDEFS profile entry that should be used for this DataInterchange session.
ZCCBCID	Error module ID. This field is used internally by DataInterchange for logging of errors, but it can prove useful in problem determination.
ZCCBXFID	Function ID. This field is used internally by DataInterchange for logging of errors, but it can prove useful in problem determination.
ZCCBLPID	Language profile ID. A value for the language profile ID must be placed in the CCB before the DataInterchange initialization request. During the initialization request, DataInterchange verifies that a LANGPROF entry exists with a key value equal to the value in this field. If a LANGPROF entry does not exist, initialization fails.
ZCCBCPID	Code page ID. This field is currently not used.
ZCCBRSV	Reserved for DataInterchange. Do not alter after initialization.
ZCCBCCXP	Pointer to CCB extension. Do not alter after initialization.
ZCCBCABP	Pointer to common area block. Do not alter after initialization.
ZCCBDBID	MVS DB2 subsystem ID. Do not alter after initialization.
ZCCBDBPL	MVS DB2 plan / AIX DB2 alias. Do not alter after initialization.
ZCCBDBUI	AIX DB2 user ID.
ZCCBDBPW	AIX DB2 password.
ZCCBRSV1	Reserved for DataInterchange. Do not alter after initialization.
ZCCBRSV2	Reserved for DataInterchange. Do not alter after initialization.

Function Control Block (FCB)

The function control block (FCB) identifies the function within the service identified by the SNB that is being requested. See the description of the **ZFCBFUNC** field for more details.

Table A-3. FCB Definition. Layout of the FCB and Descriptions of Fields

Name	Type	Offset	Length	Description
ZFCBLL	Binary	0	2	FCB length
ZFCBFUNC	Binary	2	2	Function code

FCB Field Descriptions

ZFCBLL	Length of FCB. A 2-byte binary field that contains the length of the FCB control block. An FCB is 4 bytes.
ZFCBFUNC	<p>Function code. The function within the service (defined by the SNB control block) that is being requested. A service represents a logical grouping of function. The ZFCBFUNC field is used to indicate exactly which function of the service is being requested. The function codes associated with DataInterchange API services are:</p> <ul style="list-style-type: none">• Environmental services - ENVSERV<ul style="list-style-type: none">1 Initialize DataInterchange2 Terminate DataInterchange5 Switch APPLID• Translation services - TRANPROC<ul style="list-style-type: none">131 Production translate to standard111 Test translate to standard212 Production deenvelope and translate to application211 Test deenvelope and translate to application213 Translate a specific transaction to application1000 End translation• Enveloping services - TRANPROC<ul style="list-style-type: none">1 Return interchange header2 Return group header3 Return transaction header215 Envelope transactions214 Deenvelope transactions990 Close and queue current envelope991 Issue COMMIT request• Data extraction services - TRANPROC<ul style="list-style-type: none">216 Retrieve detailed data217 Retrieve transaction image218 Retrieve functional acknowledgment image219 Retrieve transaction acknowledgment image• Communications services - COMM<ul style="list-style-type: none">211 Send transactions221 Send files232 Receive233 Cancel252 Process network acknowledgment300 Return file name110 Queue transaction data

- Update Status Services - TRANSSRV
 - 210 Update using envelope key (account number, user ID)
 - 211 Update with transaction handle
 - 212 Update using alternate key (account number, user ID)
 - 213 Update using envelope key (qualifier, receiver ID)
 - 214 Update using alternate key (qualifier, receiver ID)
 - 215 Update using envelope key (trading partner nickname)
- SYNCPOINT services - SYNCSEV
 - 1 Initialize SYNCPOINT services
 - 2 Request a COMMIT
 - 3 Request a ROLLBACK

Translator Control Block (TRCB)

The translator control block (TRCB) is the primary control block used to convey information between an application and DataInterchange when using translation, enveloping, and data extraction services. For more information, see “Translation Services” on page 3-20, “Enveloping Services” on page 3-69, and “Data Extraction Services” on page 3-101.

Table A-4 defines the functions of the TRCB fields.

Table A-4 (Page 1 of 5). TRCB Definition. Layout of the Translator Control Block (TRCB) and Descriptions of Fields

Name	Type	Offset	Length	Description
BLKLEN	Binary	0	2	Block length
RSRVD1	Binary	2	2	Reserved
BLKNME	Character	4	8	Block name
REQID	Character	12	16	Requestor ID
APPFIL	Character	28	8	Application file name
ATFID	Character	36	16	Application data format ID
ATSID	Character	52	16	Structure ID
EJECT	Character	68	1	Processing flag
INTPID	Character	69	35	Internal trading partner ID
APPCTNUM	Character	104	32	Application control value
TRNID	Character	136	16	Standard transaction ID
TEST	Character	152	1	Usage indicator
IHCTL	Character	153	9	Interchange control number
GHCTL	Character	162	9	Group control number
THCTL	Character	171	9	Transaction control number
ENVTYPE	Character	180	1	Envelope type
BLKTYPE	Character	181	1	Data block type
DUPTRAN	Character	182	1	Duplicate flag

Table A-4 (Page 2 of 5). TRCB Definition. Layout of the Translator Control Block (TRCB) and Descriptions of Fields

Name	Type	Offset	Length	Description
ITPBREAK	Character	183	1	New interchange on INTPID change
REQSIZE	Binary	184	4	Required size
XPANDED	Character	188	1	Expanded flag
NEWENV	Character	189	1	New interchange flag
NEWGRP	Character	190	1	New group flag
NEWTRN	Character	191	1	New transaction flag
QSIZE	Binary	192	4	Interchange size
ESIZE	Binary	196	4	Bytes of interchange processed
GRPNUM	Binary	200	4	Groups processed so far
TRNNUM	Binary	204	4	Transactions in interchange processed
SEGNUM	Binary	208	4	Segment in interchange processed
TRNGRP	Binary	212	4	Transactions in group processed
SEGTRN	Binary	216	4	Segment count for transaction
ERRNUM	Binary	220	4	Error counter
QBT	Character	224	8	Interchange size
IHXCTL	Character	232	14	Interchange control number
ISYNTAXID	Character	246	4	Interchange syntax ID (E,T env)
ISYNTAXVER	Character	250	1	Interchange syntax VER (E,T env)
ISIDQUAL	Character	251	4	Interchange sender ID qualifier
ISID	Character	255	35	Interchange sender ID
ISENDNAME	Character	290	14	Interchange sender Name (T env) Application sender Code (U env)
IREVROUT	Character	304	14	Interchange reverse routing (E env)
IRIDQUAL	Character	318	4	Interchange receiver ID qualifier
IRID	Character	322	35	Interchange receiver ID
IRECVNAME	Character	357	14	Receiver name (U,T env)
IROUTEADDR	Character	371	14	Routing address (E env)
IDATE	Character	385	6	Interchange date
ITIME	Character	391	6	Interchange time
IVERREL	Character	397	5	Interchange version/release
IGT	Character	402	6	Interchange group total
ITT	Character	408	6	Interchange transaction total
IST	Character	414	10	Interchange segment total
IBT	Character	424	8	Interchange byte total
ISPW	Character	432	14	Interchange password
IAPREF	Character	446	14	Application reference
ISTDID	Character	460	4	Interchange std ID (I, X env)

Table A-4 (Page 3 of 5). TRCB Definition. Layout of the Translator Control Block (TRCB) and Descriptions of Fields

Name	Type	Offset	Length	Description
RSRVD2	Character	464	1	Reserved
IPRIOR	Character	465	1	Priority (E,T env)
ICOMMAGREE	Character	466	35	Communication agree (E env)
GHXCTL	Character	501	14	Group control number
GFGID	Character	515	6	Group functional group ID
GSIDQUAL	Character	521	4	Group sender qualifier (E env)
GSID	Character	525	35	Group application sender ID
GRIDQUAL	Character	560	4	Group receiver qualifier (E env)
GRID	Character	564	35	Group application receiver ID
GDATE	Character	599	6	Group date
GTIME	Character	605	6	Group time
GVER	Character	611	12	Group version
GREL	Character	623	12	Group release
GTT	Character	635	6	Group transaction total
GAPW	Character	641	14	Group password
GRESAGENCY	Character	655	2	Group responsible agency
RSRVD3	Character	657	12	Reserved
THXCTL	Character	669	14	Transaction control number
TTC	Character	683	6	Transaction ID
TVER	Character	689	6	Transaction version
TREL	Character	695	6	Transaction release
TST	Character	701	10	Transaction segment total
LASTINENV	Character	711	1	Last transaction in interchange
XACFIELD	Character	712	35	Application control number
FABUILT	Character	747	1	Functional acknowledgment built
QGTNUM	Binary	748	4	Total groups in interchange
QTTNUM	Binary	752	4	Total transactions in interchange
QSTNUM	Binary	756	4	Total segments in interchange
QGT	Character	760	6	Total groups in interchange
QTT	Character	766	6	Total transactions in interchange
QST	Character	772	10	Total segments in interchange
FASPM	Character	782	8	FA standard profile member
ENVCHK	Character	790	1	Envelope status check
TRNSTAT	Character	791	1	RAWDATA transaction status
APTYPE	Character	792	2	Application file type
TSKEY	Packed	794	10	Packed transaction handle
TSKEYU	Character	804	20	Unpacked transaction handle

Table A-4 (Page 4 of 5). TRCB Definition. Layout of the Translator Control Block (TRCB) and Descriptions of Fields

Name	Type	Offset	Length	Description
MAPKEY	Character	824	16	Trading partner transaction ID
BATCHID	Character	840	8	Batch ID
ENVLDATE	Character	848	8	Earliest envelope date
TRXLIFE	Binary	856	2	Transaction life span
IMGLIFE	Binary	858	2	Transaction image life span
HOLDFLAG	Character	860	1	Hold flag
BNDLFLAG	Character	861	1	Bundle flag
RAWDATA	Character	862	1	RAWDATA flag
ENVLDELAY	Character	863	1	Delayed enveloping flag
TRXACCEPT	Character	864	1	Transaction acceptable flag
TRABORT	Character	865	1	Translator abort flag
FILEID	Character	866	8	User-defined file
DSNAME	Character	874	56	Physical data set name
QNETID	Character	930	8	Network ID
QPTTOPT	Character	938	1	Point-to-point network flag
QSRPGM	Character	939	1	Send/receive program flag
QDDNAME	Character	940	8	DDNAME for transaction file
FUNACKFLE	Character	948	8	DDNAME for FA file
QTPNICK	Character	956	16	Trading partner nickname
QRC	Binary	972	2	Queuing return code
QERC	Binary	974	2	Queuing extended return code
ERRCDES	Binary	976	20	First 10 error codes
INMEMTRANS	Binary	996	2	In-storage transactions
NOCOMMIT	Character	998	1	Commit flag
SCOPE	Character	999	1	Recovery scope
TPNICK	Character	1000	16	Trading partner nickname
CONCATENATE	Character	1016	1	Concatenation flag
ASSERTLVL	Character	1017	1	Assertion level
RAWDATAOUT	Character	1018	1	RAWDATA wanted for output
FIXEDTRX	Character	1019	1	Fixed-to-fixed mapping flag
MRREQID	Character	1020	16	Management reporting requestor ID
ERRFILTER	Character	1036	80	Initial error filter
FFILEID	Character	1116	8	File ID for fixed translations
VARTRACE	Character	1124	1	Variable trace wanted flag
EXPTRACE	Character	1125	1	Expression trace wanted flag
SSEGVAL	Character	1126	1	Service segment validation flag
TRXFACODE	Character	1127	1	FA code generated for transaction

Table A-4 (Page 5 of 5). TRCB Definition. Layout of the Translator Control Block (TRCB) and Descriptions of Fields

Name	Type	Offset	Length	Description
IUSEREXIT	Character	1128	8	User exit for envelopes
IUSERAREA	Binary	1136	4	User area for IUSEREXIT
IUSERACCESS	Character	1140	1	User exit access type
IUSERTYPE	Character	1141	2	User exit program type
MAPCHAIN	Character	1143	1	Mapchain active flag
FORCETEST	Character	1144	1	Forced test received translate
ENVPRBRK	Character	1145	1	Envelope profile name change ISA break
SUSBLKF	Character	1146	1	CICS suspend block flag
CLRERRS	Character	1147	1	Clear ERRCODES array flag
FARC	Binary	1148	4	FA return code
FAERC	Binary	1152	4	FA extended return code
SAPUPDT	Character	1156	1	SAP updates requested
SAPTRX	Character	1157	1	SAP transaction indicator
SAPCLIENT	Character	1158	3	SAP client
SAPDOCNUM	Character	1161	16	SAP document number
SAPSEQ	Character	1177	6	SAP error record seq number
ROUTCODE	Character	1183	3	Generic send usage routing code
FAEREQ	Character	1186	1	FA envelope file required
BOUNDARY	Character	1187	1	Incremental translation flag
SUSBLKP	Binary	1188	4	CICS SUSPEND block pointer
CUSERDATA	Character	1192	256	User data area
APPLTPID	Character	1448	15	Application trading partner ID
EXTENDC	Character	1464	1	Extend the C record flag
VAXFLAG	Character	1465	1	Pageable translation flag
GDATE8	Character	1466	8	Group envelope 8-byte data
RECOVBAD	Character	1474	1	Recover from bad standard data
RSRVD4	Character	1475	61	Reserved for DataInterchange

TRCB Field Descriptions

The following list describes the TRCB fields.

BLKLEN	Length of TRCB. A 2-byte binary field that contains the length of the TRCB control block. A TRCB is 1536 bytes.
RSRVD1	Reserved.
BLKNME	The name of the TRCB, which is EDITRCB.
REQID	The requestor ID for a member in the requestor profile, which is REQPROF. This field might be required for the translate file and deenvelope requests, because the RECEIVE FILE NAME field in the requestor profile contains the ddname for the file to be processed.

For more information on the translate and deenvelope functions, see “API - Translate File” on page 3-57 and “Deenvelope Function” on page 3-86.

If a value for FILEID is provided, this field is not required.

APPFIL	<p>The name of a file for writing application records. This field is returned by the translator during operations that translate standard data to an application format. The value is taken from the application data format definition, unless a value is supplied in the trading partner receive usage record.</p> <p>In MVS, APPFILE is a ddname for a file. In CICS, the storage mechanism represented by APPFILE is indicated in the APPTYPE field. For more information, see “API - Translate File” on page 3-57, “API - Translate Specific” on page 3-49, and “Deenvelope Function” on page 3-86.</p>
ATFID	<p>The application data format ID. This is a required input field for any operation that translates application data to a standard format, because it is part of the primary key used to determine the map for translating the data. For more information, see “API - Translate to Standard” on page 3-27.</p> <p>This field is an output field for operations that translate standard data to an application format. For more information, see “API - Translate File” on page 3-57, “API - Translate Specific” on page 3-49, and “Deenvelope Function” on page 3-86. It is also an output field for the enveloping function. For more information on the enveloping process, see “Envelope Function” on page 3-72.</p>
ATSID	<p>The name of the structure that describes the format of the application data. Unless RAWDATA is used, this is a required input field for functions that translate data from application to standard format. For more information, see “API - Translate to Standard” on page 3-27.</p> <p>This is also an output field for functions that translate standard data to an application format. For more information, see “API - Translate File” on page 3-57 and “API - Translate Specific” on page 3-49.</p>
EJECT	<p>This field is used by almost all functions, as both an input and an output field. Refer to the detailed descriptions of the functions you are using for the specific allowable values for the EJECT field. In general, this field is used to indicate when all data associated with a transaction has been provided, and when an interchange has been written to the file associated with the network.</p>
INTPID	<p>Identifies the internal trading partner ID associated with a transaction. The internal trading partner ID is the name of the trading partner known to the application and is often referred to as a customer or vendor number.</p> <p>This is a required input field for functions that translate application data to a standard format, because it is part of the key used to locate the map used to translate the data. For more information, see “API - Translate to Standard” on page 3-27.</p> <p>If RAWDATA is used, INTPID is not a required input field, but becomes the output field. It is also the output field for those functions that translate from standard format to application format, and for the enveloping function. For more information, see “API - Translate File” on page 3-57, “API - Translate Specific” on page 3-49, “Deenvelope Function” on page 3-86, and “Envelope Function” on page 3-72.</p>
APPCTLNUM	<p>A 32-byte version of XACFIELD.</p>
TRNID	<p>The standard transaction or message ID that is being translated, enveloped, and deenveloped. This is always a returned field. Example values are 850 for an X12 purchase order and INV01CE for an EDIFACT invoice.</p>

TEST The test status of the transaction. This is a required input field when data is translated from an application format to a standard format, because it is part of the key used to locate the map used to translate the data. For more information, see “API - Translate to Standard” on page 3-27.

Valid values are:

Value	Description
-------	-------------

I	Specifies that this is an information transaction and that an information usage should be used, if one exists. If an information usage does not exist, a production usage will be used instead. Even when a production usage is used, the transaction is flagged as an information transaction.
P	Specifies that this is a production transaction and that only a production usage should be used. This is the default.
T	Specifies that this is a test transaction and that a test usage should be used, if one exists. If a test usage does not exist, a production usage will be used instead. Even when a production usage is used, the transaction is flagged as a test transaction.
U	Specifies that the translator should determine if the transaction is test, information, or production based on the usage found. If a test usage exists, the transaction is a test transaction, and a value of T is returned. If an information usage exists, the transaction is an information transaction, and a value of I is returned. If only a production usage exists, the transaction is a production transaction, and a value of P is returned.

TEST is an output field when translating from standard format to application format, deenveloping, or enveloping. The interchange contains a test indicator that sets the test status of all transactions within the interchange.

IHCTL A 9-character version of **IHXCTL**.

GHCTL A 9-character version of **GHXCTL**.

THCTL A 9-character version of **THXCTL**.

ENVTYPE The type of interchange enveloping used. This is a returned value for all translation, enveloping, and deenveloping functions. Valid values are:

Value	Description
-------	-------------

E	UNB/UNZ
I	ICS/ICE
T	STX/END
U	BG/EG
X	ISA/IEA

BLKTYPE Indicates whether the 2- or 4-byte length data blocks are being used for the TRIDB and TRODB structures. The formats for TRIDB and TRODB are always the same. In a previous revision, when the translator was limited to 32 K interchanges and structures, the TRIDB and TRODB structures used 2-byte lengths. After the 32 K limitation was removed, new TRIDB and TRODB formats were introduced that used 4-byte lengths. Valid values are:

Value	Description
-------	-------------

H	4-byte length blocks
(other)	2-byte length blocks

DUPTRAN	<p>This is usually an output field for those functions that translate data from standard format to application format or deenvelope functions. For more information, see “API - Translate File” on page 3-57, “API - Translate Specific” on page 3-49, and “Deenvelope Function” on page 3-86. A value of Y indicates that a transaction was part of a duplicate interchange (an interchange that has been received more than once). A value of N indicates the transaction was not part of a duplicate interchange.</p> <p>This is an input field for the translate file and the deenvelope function. For more information on these functions, see “API - Translate File” on page 3-57, and “Deenvelope Function” on page 3-86. When you set the DUPTRAN field to N on the first request, duplicate interchanges are considered errors. Any value other than N indicates that duplicate interchanges should not be considered errors.</p>						
ITPBREAK	<p>Specifies whether a change in value of the internal trading partner (INTPID field) should create a new interchange. Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Specifies that a change in internal trading partner should cause a new interchange</td></tr> <tr> <td>(other)</td><td>Specifies that a change in internal trading partner should not cause an interchange break, unless the change in internal trading partner results in a new trading partner nickname</td></tr> </table> <p>This field applies only to enveloping situations. For more information, see the “API - Translate to Standard” on page 3-27 and “Envelope Function” on page 3-72.</p>	Value	Description	Y	Specifies that a change in internal trading partner should cause a new interchange	(other)	Specifies that a change in internal trading partner should not cause an interchange break, unless the change in internal trading partner results in a new trading partner nickname
Value	Description						
Y	Specifies that a change in internal trading partner should cause a new interchange						
(other)	Specifies that a change in internal trading partner should not cause an interchange break, unless the change in internal trading partner results in a new trading partner nickname						
REQSIZE	<p>Indicates the size of the structure being returned in the TRODB. This field applies only when data is being translated from a standard format to an application format. For more information on translation, see “API - Translate File” on page 3-57, and “API - Translate Specific” on page 3-49. This field contains a value only when the TRODB is not large enough to contain the entire structure. It is provided for informational purposes only, because all the data can be retrieved as partial structures. For more information on partial structures, see “Partial Structures (TA)” on page 3-68.</p>						
XPANDED	<p>This flag was created for migration from Release 1 to Release 2 of DataInterchange. A value of Y indicates that a post Release 1 API program is executing, not a Release 1 API program. New API programs should always set this flag to Y.</p> <p>Note: Many of the following fields in the TRCB were introduced after Release 1.</p>						
NEWENV	<p>This is a returned field for all except the translate-specific functions (function code 213). A Y indicates that the current transaction is the first transaction of an interchange.</p>						
NEWGRP	<p>This is a returned field for all except the translate-specific functions (function code 213). A Y indicates that the current transaction is the first transaction of a group.</p>						
NEWTRN	<p>This is a returned field for all translation, enveloping, and deenveloping functions. A Y indicates that a new transaction was started.</p>						
QSIZE	<p>The total number of bytes in the interchange when queued using an EJECT value of Q during “API - Translate to Standard” on page 3-27 or “Envelope Function” on page 3-72. The total number of bytes in the interchange when received (NEWENV has a value of Y during “API - Translate to Standard” on page 3-27 or “Deenvelope Function” on page 3-86). QBT is a related field.</p>						
ESIZE	<p>If an interchange is being processed, ESIZE represents the amount of the interchange processed so far. For send type operations (translate-to-standard), ESIZE represents the current size of the interchange. For receive type operations (translate-to-application),</p>						

ESIZE represents the total number of bytes of the interchange that have been processed. **IBT** is a related field.

GRPNUM	If an interchange is being processed, GRPNUM represents the number of groups in the interchange processed so far. For send type operations (translate-to-standard), GRPNUM represents the current number of groups in the interchange. For receive type operations (translate-to-application), GRPNUM represents the total number of groups of the interchange that have been processed. IGT is a related field.
TRNNUM	If an interchange is being processed, TRNNUM represents the number of transactions in the interchange processed so far. For send type operations (translate-to-standard), TRNNUM represents the current number of transactions in the interchange. For receive type operations (translate-to-application), TRNNUM represents the total number of transactions of the interchange that have been processed. ITT is a related field.
SEGNUM	If an interchange is being processed, SEGNUM represents the number of segments in the interchange processed so far. For send type operations (translate-to-standard), SEGNUM represents the current number of segments in the interchange. For receive type operations (translate-to-application), SEGNUM represents the total number of segments of the interchange that have been processed. IST is a related field.
TRNGRP	If an interchange is being processed, TRNGRP represents the number of transactions in the group processed so far. For send type operations (translate-to-standard), TRNGRP represents the current number of transactions in the group. For receive type operations (translate-to-application), TRNGRP represents the total number of transactions of the group that have been processed. GGT is a related field.
SEGTRN	The total number of segments in the current transaction. TST is a related field.
ERRNUM	The total number of errors in the current transaction.
QBT	The character representation of QSIZE .
IHXCTL	If an interchange is active, IHXCTL is the interchange control number extracted from the interchange header field with a CN data type.
ISYNTAXID	If an EDIFACT or UNTDI interchange is active, ISYNTAXID is the interchange syntax identifier extracted from the interchange header field UNB01 or STX01. For enveloping functions, ISYNTAXID is also an input field. It provides the syntax identifier to be used when building the interchange header segment.
ISYNTAXVER	If an EDIFACT or UNTDI interchange is active, ISYNTAXVER is the interchange syntax version extracted from the interchange header field UNB01 or STX01. For enveloping functions, ISYNTAXVER is also an input field. It provides the syntax identifier to be used when building the interchange header segment.
ISIDQUAL	If an EDIFACT, ICS, or ISA interchange is active, ISIDQUAL is the interchange sender ID qualifier extracted from the interchange header fields UNB04, ICS04, or ISA05. For enveloping functions, ISIDQUAL is also an input field. It provides the sender ID qualifier to be used when building the interchange header segment.
ISID	If an interchange is active, ISID is the interchange sender ID extracted from the interchange header field with an IS data type. For enveloping functions, ISID is also an input field. It provides the interchange sender ID to be used when building the interchange header segment. If this value differs from the current interchange sender ID value, a new interchange is created. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.
ISENDNAME	If a UNTDI interchange is active, ISENDNAME is the interchange sender name extracted from the interchange header field STX04. For enveloping functions, ISENDNAME is also an input field. It provides the sender name to be used when building the interchange

	header segment. If a UCS interchange is active, ISENDNAME is the application sender code extracted from the interchange header field UCS03 if this field is not an IS data type. For enveloping functions, ISENDNAME is also an input field. It provides the application sender code to be used when building the interchange header segment if this field is not an IS data type.
IREVROUT	If an EDIFACT interchange is active, IREVROUT is the interchange reverse routing extracted from the interchange header field UNB05. For enveloping functions, IREVROUT is also an input field. It provides the reverse routing to be used when building the interchange header segment.
IRIDQUAL	If an EDIFACT, ICS, or ISA interchange is active, IRIDQUAL is the interchange receiver ID qualifier extracted from the interchange header fields UNB04, ICS04, or ISA05. For enveloping functions, IRIDQUAL is also an input field. It provides the receiver ID qualifier to be used when building the interchange header segment.
IRID	If an interchange is active, IRID is the interchange receiver ID extracted from the interchange header field with an IR data type. For enveloping functions, IRID is also an input field. It provides the interchange receiver ID to be used when building the interchange header segment. If this value differs from the current interchange receiver ID value, a new interchange is created. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.
IrecvNAME	If a UNTDI interchange is active, IRECVNAME is the interchange receiver name extracted from the interchange header field STX06. For UNTDI enveloping functions, IRECVNAME is also an input field. It provides the receiver name to be used when building the interchange header segment. If a UCS interchange is active, IRECVNAME is the application receiver code extracted from the interchange header field UCS04 if this field is not an IR data type. For UCS enveloping functions, IRECVNAME is also an input field. It provides the application receiver code to be used when building the interchange header segment if this field is not an IS data type.
IRouteADDR	If an EDIFACT interchange is active, IROUTEADDR is the interchange routing address extracted from the interchange header field UNB08. For enveloping functions, IROUTEADDR is also an input field. It provides the routing address to be used when building the interchange header segment.
IDATE	If an interchange is active, IDATE is the interchange date extracted from the interchange header field with a DT data type.
ITIME	If an interchange is active, ITIME is the interchange time extracted from the interchange header field with a TM data type.
IVERREL	If an interchange is active, IVERREL is the interchange version or release extracted from the interchange header field with a VR or LV data type.
IGT	Character representation of GRPNUM .
ITT	Character representation of TRNNUM .
IST	Character representation of SEGNUM .
IBT	Character representation of ESIZE .
ISPW	If an interchange is active, ISPW is the interchange password extracted from the interchange header field with a PW data type. For enveloping functions, ISPW is also an input field. It provides the interchange password to be used when building the interchange header segment. If this value differs from the current interchange password value, a new interchange is created. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.

IAPREF	If an interchange is active, IAPREF is the interchange application reference extracted from the interchange header field with an AP data type. For enveloping functions, IAPREF is also an input field. It provides the interchange application reference to be used when building the interchange header segment. If this value differs from the current interchange application reference value, a new interchange is created. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.
ISTDID	If an X12 or ICS interchange is active, ISTDID is the interchange standard ID extracted from the interchange header fields ISA11 or ICS02. For enveloping functions, ISTDID is also an input field. It provides the standard ID to be used when building the interchange header segment.
RSRVD2	Reserved
IPRIOR	If an EDIFACT or UNTDI interchange is active, IPRIOR is the interchange processing priority extracted from the interchange header fields UNB15 or STX12. For enveloping functions, IPRIOR is also an input field. It provides the processing priority to be used when building the interchange header segment.
ICOMMAGREE	If an EDIFACT interchange is active, ICOMMAGREE is the interchange communication agreement extracted from the interchange header fields UNB17. For enveloping functions, IPRIOR is also an input field. It provides the processing priority to be used when building the interchange header segment.
GHXCTL	If a group is active, GHXCTL is the group control number extracted from the group header field with a CN data type.
GFGID	If a group is active, GFGID is the functional group ID value associated with the group.
GSIDQUAL	If an EDIFACT group is active, GSIDQUAL is the group sender ID qualifier extracted from the group header field UNG03. For enveloping functions, GSIDQUAL is also an input field. It provides the sender ID qualifier to be used when building the group header segment.
GSID	If a group is active, GSID is the group application sender ID extracted from the group header field with an AS data type. For enveloping functions, GSID is also an input field. It provides the group application sender ID to be used when building the group header segment. If this value differs from the current group application sender ID value, a new group is created. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.
GRIDQUAL	If an EDIFACT group is active, GRIDQUAL is the group receiver ID qualifier extracted from the group header field UNG05. For enveloping functions, GRIDQUAL is also an input field. It provides the receiver ID qualifier to be used when building the group header segment.
GRID	If a group is active, GRID is the group application receiver ID extracted from the group header field with an AR data type. For enveloping functions, GRID is also an input field. It provides the group application receiver ID that should be used when building the group header segment. If this value differs from the current group application receiver ID value, a new group is created. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.
GDATE	If a group is active, GDATE is the group date extracted from the group header field with a DT data type.
GTIME	If a group is active, GTIME is the group time extracted from the group header field with a TM data type.

GVER	If a group is active, GVER is the group version extracted from the group header field with a VR data type. For enveloping functions, GVER is also an input field. It provides the group version that should be used when building the group header segment. If this value differs from the current version value, a new group is created. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.
GREL	If a group is active, GREL is the group release extracted from the group header field with an LV data type. For enveloping functions, GREL is also an input field. It provides the group release that should be used when building the group header segment. If this value differs from the current group release value, a new group is created. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.
GTT	Character representation of TRNGRP .
GAPW	<p>If a group is active, GAPW is the group application password extracted from the group header field with a PW data type. For enveloping functions, GAPW is also an input field. It provides the group application password that should be used when building the group header segment.</p> <p>If this value differs from the current group application password value, a new group is created. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.</p>
GRESAGENCY	If an EDIFACT group is active, GRESPAGENCY is the group controlling agency extracted from the group header field UNG09. For enveloping functions, GRESPAGENCY is also an input field. It provides the controlling agency to be used when building the group header segment. If an ICS, UCS, or X12 group is active, GRESPAGENCY is the group responsible agency extracted from the group header field GS07. For enveloping functions, GRESPAGENCY is also an input field. It provides the responsible agency to be used when building the group header segment.
RSRVD3	Reserved
THXCTL	If a transaction is active, THXCTL is the transaction control number extracted from the transaction header field with a CN data type.
TTC	If a transaction is active, TTC is the transaction or message ID extracted from the transaction header field with a TC data type.
TVER	If a transaction is active, TVER is the transaction version extracted from the transaction header field with a VR data type. For enveloping functions, TVER is also an input field. It provides the transaction version that should be used when building the transaction header segment. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.
TREL	If a transaction is active, TREL is the transaction release extracted from the transaction header field with an LV data type. For enveloping functions, TREL is also an input field. It provides the transaction release that should be used when building the transaction header segment. For more information on enveloping functions, see “Envelope Function” on page 3-72 and “API - Translate to Standard” on page 3-27.
TST	Character representation of SEGNUM .
LASTINENV	If the current transaction is the last transaction of the interchange, this field is set to Y.
XACFIELD	The application control number is a returned value for all translation and enveloping functions. The field that contains the application control number is indicated in the application data format with an AC data type. This field can be overridden by providing up

to 8 fields that can be concatenated to form an application control number in the trading partner transaction ID (map). This field is the value that uniquely identifies the transaction to the application.

FABUILT During a deenvelope process, functional acknowledgments can be built. If they are, **FABUILT** contains the envelope type for the functional acknowledgment. The values in this field are the same as the values for **ENVTYPE** with the following two additions:

Value Description

- G Indicates that GS/GE enveloping was used for the functional acknowledgment.
- S Indicates that the functional acknowledgments were added to the Transaction Store, but were not enveloped.

The **ENVLDELAY** field controls enveloping of functional acknowledgments. For more information on the deenvelope process, see “Deenvelope Function” on page 3-86 and “API - Translate File” on page 3-57.

QGTNUM The total number of groups in the interchange, when queued (**EJECT** value of Q during “API - Translate to Standard” on page 3-27 or “Envelope Function” on page 3-72). The total number of groups in the interchange, when received (**NEWENV** value of Y during “API - Translate to Standard” on page 3-27 or “Deenvelope Function” on page 3-86). **QGT** is a related field.

QTTNUM The total number of transactions in the interchange, when queued (**EJECT** value of Q during “API - Translate to Standard” on page 3-27 or “Envelope Function” on page 3-72). The total number of transactions in the interchange, when received (**NEWENV** value of Y during “API - Translate to Standard” on page 3-27 or “Deenvelope Function” on page 3-86). **QTT** is a related field.

QSTNUM The total number of segments in the interchange, when queued (**EJECT** value of Q during “API - Translate to Standard” on page 3-27 or “Envelope Function” on page 3-72). The total number of segments in the interchange, when received (**NEWENV** value of Y during “API - Translate to Standard” on page 3-27 or “Deenvelope Function” on page 3-86). **QST** is a related field.

QGT Character representation of **QGTNUM**.

QTT Character representation of **QTTNUM**.

QST Character representation of **QSTNUM**.

FASPM The standard profile member used in building the service segments for the functional acknowledgment. This field is for the translator’s use only.

ENVCHK This field can be used to ask the translator to verify the transaction status during an envelope function for the intent of the envelope operation. For more information on the enveloping process, see “Envelope Function” on page 3-72. Valid values are:

Value Description

- 1 The intent is to envelope transactions, and the translator should reject this transaction if it has been enveloped before.
- 2 The intent is to reenvelope transactions, and the translator should reject this transaction if it has not been enveloped before.

(other) The status of the transaction is not checked.

This field becomes important if multiple jobs can be enveloping concurrently and you want to ensure that a transaction is not enveloped and sent twice.

TRNSTAT	<p>This flag indicates the status of a RAWDATA transaction. It is a single character field used in conjunction with the RAWDATA flag and can be used by the application to determine the status of a RAWDATA transaction. It can also be used by the application to determine the next action expected from the application. Valid values are:</p>		
	<table> <tr> <th data-bbox="373 325 454 357">Value</th><th data-bbox="454 325 1471 357">Description</th></tr> </table>	Value	Description
Value	Description		
I	<p>This is set if the application data format has defined a structure that starts a transaction and the structure has not been received. Data is ignored until the starting transaction structure is recognized.</p>		
S	<p>This is set when the starting transaction structure has been received. If a starting structure is not defined, this is set when the first structure is received.</p>		
C	<p>This is set if a transaction is in progress and the current structure passed did not mark a significant milestone. The structure was neither the starting nor the ending structure.</p>		
R	<p>If a transaction is in progress and a structure that defines the start of a transaction is received, this flag is set and the data is ignored. The application must first call the translator with an EJECT flag value of Y to process the current transaction, and then calls the translator again with the same data to start a new transaction.</p>		
Y	<p>This is set when a transaction is in progress and a structure recognized as the ending structure is received. This indicates the transaction is ready to be processed and a call to the translator is needed with an EJECT value of Y.</p>		
APTTYPE	<p>The type of application file containing the values that apply only to the CICS environment. Valid values are:</p>		
	<table> <tr> <th data-bbox="373 1060 454 1092">Type</th><th data-bbox="454 1060 1471 1092">Description</th></tr> </table>	Type	Description
Type	Description		
TD	<p>Application data should be written to the transient data queue identified by the first 4 characters of APPFILE.</p>		
TM	<p>Application data should be written to the temporary storage queue (main) identified by APPFILE.</p>		
TS	<p>Application data should be written to the temporary storage queue (auxiliary) identified by APPFILE.</p>		
VS	<p>Application data should be written to the VSAM ESDS file identified by APPFILE.</p>		
TX	<p>Application data should be passed to the transaction code identified by the first 4 characters of APPFILE.</p>		
PG	<p>Application data should be passed to the program identified by APPFILE.</p>		
TSKEY	<p>The Transaction Store handle value for a transaction, in packed format. This is an input field for the translate-specific and envelope functions. For more information on these functions, see “API - Translate Specific” on page 3-49 and “Envelope Function” on page 3-72.</p> <p>This field indicates the transaction to be translated or enveloped. For all other functions, TSKEY is an output field that indicates the transaction key value just processed.</p> <p>Note: If handling the key in a packed format is a problem to your program, TSKEYU contains the key value in an unpacked format. TSKEYU can also be used as the input field. You indicate this by making the TSKEY value all blanks or all binary zeros.</p>		

TSKEYU	<p>The Transaction Store handle value for a transaction in unpacked format. If the TSKEY field value is blanks or binary zeros, this becomes the input field for the translate-specific and envelope functions. For more information on these functions, see “API - Translate Specific” on page 3-49 and “Envelope Function” on page 3-72.</p> <p>This field indicates the transaction to be translated or enveloped. TSKEYU is a returned value for all translation, enveloping, and deenveloping functions. If TSKEYU is used as an input field, its packed value is returned in TSKEY.</p>						
MAPKEY	A returned value that provides the trading partner transaction (map) ID value used to translate the data.						
BATCHID	An input field for all translation functions. For more information on translation functions, see “API - Translate to Standard” on page 3-27, “API - Translate File” on page 3-57, and “API - Translate Specific” on page 3-49. BATCHID is an indexed field within the transaction store database. It provides an efficient method for retrieving transactions from the store. It can also be used to identify all the transactions that were processed during a particular execution of a job. If a value is not supplied, the system date and time are used to create a default value of DDHHMMSS.						
ENVLDATE	The earliest date that a transaction is eligible to be enveloped and sent. If a date is not provided, the transaction is eligible immediately, unless other conditions indicate that it is not eligible (such as a transaction in held status). This is an input field for translate-to-standard only (“API - Translate to Standard” on page 3-27).						
TRXLIFE	The number of days the transaction should remain in the Transaction Store before it is eligible to be purged. If this field contains zero, a default value of 30 days is used. The default value is returned in the TRXLIFE field. This is an input field for all translating and deenveloping functions. For more information on the translating and deenveloping functions, see “API - Translate to Standard” on page 3-27, “API - Translate File” on page 3-57, and “Deenvelope Function” on page 3-86.						
IMGLIFE	<p>The number of days the transaction image should remain in the Transaction Store before it is eligible to be purged. If this field contains zero, a default value of 30 days is used. The default value is returned in the IMGLIFE field. This is an input field for all translating and deenveloping functions. For more information on the translating and deenveloping functions, see “API - Translate to Standard” on page 3-27, “API - Translate File” on page 3-57, and “Deenvelope Function” on page 3-86.</p> <p>Note: Currently, a transaction’s life span (TRXLIFE) and the transactions image life span (IMGLIFE) are always the same. However, IMGLIFE should still be set.</p>						
HOLDFLAG	<p>Specifies if a transaction should be held. Valid values are :</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>The transaction is given a HELD status.</td></tr> <tr> <td>(other)</td><td>The transaction is given an active status.</td></tr> </table> <p>This is an input field for functions that initially add a transaction to the Transaction Store. For more information, see “API - Translate to Standard” on page 3-27, “API - Translate File” on page 3-57, and “Deenvelope Function” on page 3-86.</p>	Value	Description	Y	The transaction is given a HELD status.	(other)	The transaction is given an active status.
Value	Description						
Y	The transaction is given a HELD status.						
(other)	The transaction is given an active status.						

BNDLFLAG Specifies the bundle status for transaction. Valid values are:

Value	Description
-------	-------------

- | | |
|---------|---|
| Y | Specifies the start of a bundle, which is a group of related transactions that must be acted on together. This transaction becomes the controlling transaction for the bundle. All transactions that follow become part of the bundle until: <ul style="list-style-type: none">• Another transaction ends the bundle or starts a new bundle.• The input data forces a new group or envelope.• The translation process ends. |
| N | Specifies the last transaction in the bundle. |
| (blank) | Specifies to continue as before. |

This is an input field for translate-to-standard. For more information, see “API - Translate to Standard” on page 3-27. If the current trading partner is not using functional groups (a functional group field value in the trading partner profile of N), changes in data that would generally cause a new group to be created are ignored, and the bundles are not terminated.

Note: The bundled transactions are also referred to as *clustered transactions*.

RAWDATA Indicates if RAWDATA processing is being used. Valid values for translate-to-standard functions are:

Value	Description
-------	-------------

- | | |
|---------|---|
| Y | The RAWDATA interface is being used. For more information on this interface, see “Special Considerations (TS)” on page 3-40. |
| (other) | C and D record type interface is being used (application knows the format of the data and is communicating the type of data provided with the ATSID field). |

For more information on translate-to-standard functions, see “API - Translate to Standard” on page 3-27.

Valid values for translate-to-application functions are:

Value	Description
-------	-------------

- | | |
|---------|--|
| Y | RAWDATA processing is requested. If RAWDATA specifications have been provided in the application data format ID, the translator automatically moves in the record ID values, and the internal trading partner ID is moved to the specified field. The RAWDATA flag is returned with a value of Y.

If RAWDATA specifications have not been supplied in the application data format, the RAWDATA flag is set to N and RAWDATA processing does not occur. The application data format ID is returned in the ATFID field. |
| (other) | RAWDATA processing is not requested. |

For more information on translate-to-application functions, see “API - Translate File” on page 3-57, and “API - Translate Specific” on page 3-49.

Note: If RAWDATA processing has been requested and RAWDATA processing was possible, the DataInterchange Utility writes the data to the application file in RAWDATA format. However, if RAWDATA processing was not requested or was not possible, C and D records are written to the application file. Your application program can write the records in any desired format, regardless of the setting of the RAWDATA flag.

ENVLDELAY	<p>For the translate-to-standard function, a value of Y indicates that the transaction should not be enveloped at the same time it is translated. For more information, see “API - Translate to Standard” on page 3-27. Any other value indicates that enveloping should take place at the same time as translation. In any case, the transaction and its image is saved in the Transaction Store.</p> <p>For the deenvelope and translate file functions, a value of Y indicates that any functional acknowledgments created should not be enveloped at this time. For more information on the deenvelope and translate functions, see “Deenvelope Function” on page 3-86 and “API - Translate File” on page 3-57. Any other value indicates that functional acknowledgments should be enveloped at the same time they are created. See the FUNACKFLE field for related information. Whatever the value, the functional acknowledgment transaction and image are saved in the Transaction Store.</p> <p>The value set for ENVLDELAY on the first call of a session is the value that is used for the entire session.</p>						
TRXACCEPT	<p>Indicates whether the transaction just processed had an acceptable error level. Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Indicates that the transaction translated with an acceptable error level (less than or equal to the error level specified in the trading partner usage)</td></tr> <tr> <td>N</td><td>Indicates that the transaction had an unacceptable error level</td></tr> </table>	Value	Description	Y	Indicates that the transaction translated with an acceptable error level (less than or equal to the error level specified in the trading partner usage)	N	Indicates that the transaction had an unacceptable error level
Value	Description						
Y	Indicates that the transaction translated with an acceptable error level (less than or equal to the error level specified in the trading partner usage)						
N	Indicates that the transaction had an unacceptable error level						
TRABORT	<p>A value of Y indicates that while processing the last request, the translator had an error that was considered so serious that the translator did not continue. Y indicates that the translator has ended of its own accord. A value of N indicates the translator has not ended (did not have a terminating error).</p>						
FILEID	<p>This is the ddname (temporary storage queue name in CICS) to which an interchange should be written during a translate-to-standard (without delayed enveloping) or enveloping function. For more information on translate-to-standard and enveloping functions, see “API - Translate to Standard” on page 3-27 and “Envelope Function” on page 3-72. This field overrides the default file name in the <i>Trans data queue</i> field in the network profile.</p> <p>This is the ddname (temporary storage queue name in CICS) from which transactions should be read during a translate file or deenvelope function. For more information on the translate file and deenvelope functions, see “API - Translate File” on page 3-57 and “Deenvelope Function” on page 3-86. It also overrides the default file name in the <i>Receive file name</i> field in the requestor profile.</p>						
DSNAME	<p>The name of the data set to which transactions were written (when the EJECT field value is Q), or from which transaction data is being read (NEWENV field value of Y). In CICS, the DSNAME field has the same value as the QDDNAME field.</p>						
QNETID	<p>The network ID associated with the interchange that was written (EJECT field value of Q).</p>						
QPTTOPT	<p>Indicates whether the network associated with the interchange that was written (EJECT field value of Q) is a point-to-point network. Valid values are Y for yes, and N for no.</p>						
QSRPGM	<p>Indicates whether the network associated with the interchange that was written (EJECT field value of Q) has a send/receive program defined. Valid values are Y for yes, and N for no.</p>						
QDDNAME	<p>Indicates the ddname of the file to which the interchange was written (EJECT field value of Q).</p>						

FUNACKFLE	This is the ddname (temporary storage queue name in CICS) to which a functional acknowledgment interchange should be written if enveloping is not being delayed (ENVLDELAY field). This applies only during a translate file or deenvelope function. For more information on translate file and deenvelope functions, see “API - Translate File” on page 3-57 and “Deenvelope Function” on page 3-86. This field overrides the default file name in the <i>Trans data queue</i> field in the network profile.
QTPNICK	The trading partner nickname associated with an interchange just written (EJECT value of Q), or the trading partner nickname associated with a transaction just received.
QRC	Provides the return code on the attempt to write an interchange. This field has meaning only when EJECT has a value of E, indicating there was an error writing the interchange.
QERC	Provides the extended return code on the attempt to write an interchange. This field has meaning only when EJECT has a value of E, indicating there was an error writing the interchange.
ERRCDES	An array of 10 binary values of 2 bytes each. Use the CLRERRS flag located in this control block to reset the array. These values indicate the errors found while processing this transaction. See Table A-5 on page A-28 for the possible values and associated message IDs.
INMEMTRANS	<p>A 2-byte binary value that indicates the number of transactions that should be maintained in storage before database updates are attempted. This field has meaning only if interchange level recovery is in effect. See also the SCOPE field. This value affects the amount of concurrency you can achieve if you have multiple processes executing at the same time.</p> <p>Keeping transactions in storage delays the time the database lock is obtained and reduces the length of the database lock. The amount of storage used by each transaction depends on the function being performed:</p> <p>ENVELOPE 1508 bytes per transaction. If encryption is taking place, you must add the size of an average transaction image. For more information on the enveloping process, see “Envelope Function” on page 3-72.</p> <p>TRANSLATE AND ENVELOPE 1820 bytes per transaction. You can subtract 240 from this value, if overrides from the C record are not being used. If encryption is taking place, you must add the size of an average transaction image. For more information, see “API - Translate to Standard” on page 3-27.</p> <p>DEENVELOPE 1508 bytes per transaction. For more information, see “Deenvelope Function” on page 3-86.</p> <p>DEENVELOPE AND TRANSLATE 1580 bytes per transaction. For more information, see “API - Translate File” on page 3-57.</p> <p>If interchange level recovery is being used and a value for INMEMTRANS is not supplied, a default value of 100 is used. For more information, see “Send Recovery Scope” on page 3-42.</p>
NOCOMMIT	A value of Y indicates that the translator should not issue a COMMIT request, even when it is time based on the recovery scope. This field can be used to delay the COMMIT request until the application databases have been updated. When they are updated, an API request to commit should be issued. For more information, see “API - Issue Commit” on page 3-98.

SCOPE	Indicates the recovery scope that should be in place for this session. Valid values are:
Value	Description
T	Indicates that a transaction recovery scope should be used. The translator issues a COMMIT request at the end of every transaction during translate-to-standard or enveloping functions, and at the beginning of every transaction during translate-to-application or deenveloping functions.
E	Indicates that an interchange recovery scope should be used. The translator issues a COMMIT request only after an interchange has been written during translate-to-standard or enveloping functions, and on the next request after the last transaction of an interchange (LASTINENV field) during translate-to-application or deenveloping functions.
	<p>INMEMTRANS is a related field when an interchange recovery scope is being used. An interchange recovery scope is not recommended for interchanges that contain a large number of transactions because:</p> <ul style="list-style-type: none"> • The number of locks that can be obtained by a process is limited in DB2. If this value is exceeded, DB2 automatically issues a ROLLBACK. • If the number of transactions in an interchange exceeds INMEMTRANS, concurrent processes are blocked, from the time INMEMTRANS is exceeded until the interchange has been completely processed.
TPNICK	A returned value indicating the trading partner nickname associated with the current transaction.
CONCATENATE	This field applies only to data extraction requests and indicates whether the data extraction records should be concatenated in the output data block. Valid values are Y for yes, and N for no. For more information on data extraction requests, see “Data Extraction Services” on page 3-101.
ASSERTLVL	During the mapping process, the &ASSERTn special literal could be used to establish assertions about the transaction. For example, the total amount of this transaction does not exceed 1 million dollars. The n in &ASSERTn is the assertion level. ASSERTLVL provides the assertion level that should be active for this translation. Only &ASSERTn special literals with an n value greater than or equal to the value of ASSERTLVL is executed. If a value for the field is not supplied, a default assertion level of 0 is used, indicating that all ASSERTIONS apply.
RAWDATAOUT	Set to Y if raw data format is wanted for output data resulting from a Fixed-to-Fixed translation.
FIXEDTRX	Return value that indicates the current transaction has used a Fixed-to-Fixed mapping.
MRREQID	During a translate file or deenvelope function, it is possible to provide the name of a requestor ID in the MRREQID field. For more information on translate file and deenvelope functions, see “Deenvelope Function” on page 3-86 and “API - Translate File” on page 3-57. If this value is supplied, the management reporting component of DataInterchange is notified of the number of bytes in the interchanges processed in the session. Do this only if the interchanges being processed have not already been counted. If the data was received using DataInterchange communications functions, the data was counted at that time, and therefore a MRREQID value should not be provided.
ERRFILTER	The initial list of errors that should be filtered during this translation session. The list of errors provided here will be the list of errors that is active at the start of each transaction and provides the initial values for the DIERRFILTER named variable. The field contains a

list of unique error code values (see Table A-5 on page A-28) that should be filtered and therefore not produce an error message. Each entry in the list should be separated by a blank or a comma and a range of codes may be specified by using a dash (–) between the low and high values. For example, to filter all warning error codes plus error code 103 and 105 the specification would be:

0–99,103,105

The following errors may not be filtered and any attempt to do so will be ignored.

106–110,117–118

204,207–210

301,303,305,308–334

401,403,405–411

501,505–509

601,602,604

900–999

The following errors have special meaning if filtered. If they are filtered, then the error is ignored and a transaction, group or interchange will be processed even when an inconsistency exists.

302,304 (Transaction header/trailer inconsistent)

402,404 (Group header/trailer inconsistent)

502,503 (Interchange header/trailer inconsistent)

Under certain circumstances where the error occurs before or after the mapping is processed, error codes 3 through 6 (messages TR0403 through TR0406) cannot be filtered using the DIERRFILTER in the mapping. These errors can be filtered using the DIERRFILTER keyword on the utility Perform statement.

FFILEID	This is the ddname (temporary storage queue name in CICS) to which the result of a Fixed-to-Fixed translation should be written during a translate-to-standard (without delayed enveloping) or envelope function. This field overrides the default file name that is the concatenation of <i>Application file name</i> from the target application data format and the <i>File suffix</i> from the trading partner profile.
VARTRACE	A value of Y specifies that a variable level translator trace is wanted (for DataInterchange use only).
EXPTRACE	A value of Y specifies that an expression level translator trace is wanted (for DataInterchange use only).
SSEGVAL	<p>A flag that indicates which level of service segment validation should take place. The service segments are the segments used when a transaction is enveloped (ISA, GS, ST, UNB, UNH, UNT, etc.). No validation will occur unless SSEGVAL has one of the values below. Utility uses set this value with the SERVICESEGVAL keyword on the PERFORM command.</p> <ol style="list-style-type: none">1. Indicates the service segments should be validated for SYNTAX only. This includes checking for mandatory data that is missing, as well as data elements that are too large or too small.

2. Indicates that in addition to level 1 checking, the values in the service segment data elements should be validated according to their types (only dates and times are validated), and if a validation table has been specified, then the value of the data element will be checked against the validation table.

TRXFACODE A flag that indicates the type of functional acknowledgment that was generated for this transaction.

Value	Description
A	Accepted
R	Rejected
E	Accepted with errors

IUSEREXIT The name of a user exit that will be given to each interchange as it is created by DataInterchange during SEND processing; or the name of a user exit that will provide the interchange to be processed by DataInterchange during RECEIVE processing.

IUSERAREA 4 bytes of information that will be returned to the program identified by IUSEREXIT.

IUSERACCESS

A flag that indicates how the interchange should be presented to the IUSEREXIT program.

Value	Description
M	A value of M indicates that the interchange should be given to the exit in virtual storage. This option only applies when the IUSERTYPE is UE (user exit).
F	A value of F indicates that the interchange should be given to the exit in a file. With an IUSERACCESS value of F, the interchange is first written to the transaction data queue file, and then the IUSEREXIT program is invoked.

IUSERTYPE The type of program specified in IUSEREXIT.

Value	Description
PG	A type of PG indicates that IUSEREXIT is a program that should be linked to (EXEC CICS LINK in CICS).
UE	A type of UE indicates that IUSEREXIT is a DataInterchange user exit program defined in the ADAMCTL profile

MAPCHAIN A flag to indicate whether mapchaining is in effect.

Value	Description
Y	Indicates that the CURRENT transaction will be translated again rather than the NEXT transaction being translated.
(other)	The next transaction will be translated

FORCETEST A flag that indicates whether the deenvelope and/or translate to application process is to be forced to select only a test usage regardless of the value of the test indicator in the envelope:

Value	Description
Y	Force the process to test mode and select only a test usage if it is defined. If a test usage is not found, an error is generated and the transaction is rejected.
	If FORCETEST(Y) was used on the deenvelope process, then it also must be used on the Translate-to-Application or Retranslate-to-Application to select those transactions stored by the deenvelope.

(other) Use the test indicator from the envelope to determine the usage to select (default). An envelope without a test indicator is always considered a production envelope.

ENVPRBRK Specifies whether a change in the standard envelope member name should create a new interchange envelope or a new group envelope.

Valid values are:

Value	Description
-------	-------------

Y	A change in the standard envelope profile member name should create a new interchange envelope.
---	---

(other)	A change in the standard envelope profile member name should create a new group envelope. This is the general operational mode.
---------	---

SUSBLKF Suspend block flag (CICS API only). A Y indicates a suspend block has been allocated and its address is in SUSBLKP. If the API program supplies the suspend block (32 bytes, initialized with binary zeroes), the CICS SUSPENDs will be in effect over multiple translator calls. By having DataInterchange issue periodic SUSPENDs, AICA abends can be eliminated.

CLRERRS Clear the ERRCDDES array. A Y indicates the ERRCDDES array will be reset before processing. See the ERRCDDES field description located in this control block.

FARC The return code from the translator for the functional acknowledgment translation done during deenvelope. This return code can be found in *DataInterchange Messages and Codes*.

FAERC The extended return code from the translator for the functional acknowledgment translation done during deenvelope. This return code can be found in *DataInterchange Messages and Codes*.

SAPUPDT The SAP updates requested flag (for DataInterchange use only).

SAPTRX The SAP transaction indicator (for DataInterchange use only).

SAPCLIENT The SAP client code (for DataInterchange use only).

SAPDOCNUM The SAP document number (for DataInterchange use only).

SAPSEQ The SAP error record sequence number (for DataInterchange use only).

ROUTCODE A three-character generic routing code provided by the application and used by DataInterchange to select a generic send usage. A blank specifies a default generic send usage.

FAERREQ A flag indicating whether or not a functional acknowledgment file is required. A Y specifies that the file is required and an error will be produced if it is not available.

BOUNDARY The incremental translation flag. For more information, see "Outbound Incremental Translation" on page 3-44. During regular (or non-incremental translation), this flag should be blank.

SUSBLKP Suspend block pointer (CICS API only). This field works in tandem with SUSBLKF. If the API program supplies a suspend block (32 bytes, initialized with binary zeroes), this field should contain the address of the block and SUSBLKF should contain Y.

CUSERDATA User area provided by the user on send translation of C and D records. On receive translation, this value comes from the DataInterchange mapping variable DICUSERDATA. This field can be modified by user exits and is copied to the C record before the C record is output by DataInterchange. The default is to set the field to all blanks.

APPLTPID	Application trading partner ID. An application trading partner is considered an internal trading partner within your business organization. For example, an internal trading partner can be a division or department. The application trading partner may be used when no specific EDI trading partner is defined or in combination with the EDI trading partner. The interchange control numbers are generated using the application and EDI trading partner combination. This type of trading partner should be used with centralized EDI when the application trading partners do business with the same EDI trading partner.
EXTENDC	A Y means to use the extended C record format. Blank is the default.
VAXFLAG	An X enables Pageable Translation. The default is a blank, which means Pageable Translation is not to be used (see the PAGE utility keyword on page 1-99).
GDATE8	The group envelope in the 8-byte date format.
RECOVBAD	A Y specifies that the translator should try to recover from bad standard data. The translator will check for standard interchange headers when a segment terminator is not found on a particular standard segment. The translator will attempt to reset the delimiters and check the current segment/record for the segment terminator. Y is the default. An N indicates the translator should not try to recover.
RSRVD4	Reserved for DataInterchange.

Translator Unique Error Code Values

Table A-5 shows the error code values from the translator.

Table A-5 (Page 1 of 2). Unique Error Codes from Translator

Code (msg)	Code (msg)	Code (msg)	Code (msg)
Note: Warnings			
1 (TR0401)	2 (TR0841)	3 (TR0403)	4 (TR0404)
5 (TR0405)	6 (TR0406)	7 (TR0407)	8 (TR0408)
9 (TR0409)			
Note: Field Level Errors			
101 (TR0001)	102 (TR0002)	103 (TR0003)	104 (TR0004)
105 (TR0005)	106 (TR0006)	107 (TR0007)	108 (TR0008)
109 (TR0009)	110 (TR0014)	111 (TR0010)	112 (TR0011)
113 (TR0012)	114 (TR0013)	115 (TR0015)	116 (TR0016)
117 (TR0017)	118 (TR0018)	119 (TR0023)	120 (TR0024)
199 (TR0026)			
Note: Segment Level Errors			
201 (TR0050)	202 (TR0051)	203 (TR0052)	204 (TR0053)
205 (TR0054)	206 (TR0055)	207 (TR0019)	208 (TR0020)
209 (TR0021)	210 (TR0022)	211 (TR0056)	212 (TR0058)
213 (TR0059)	214 (TR0060)	215 (TR0057)	
Note: Transaction Level Errors (except for unique codes 301 and 303 which are Group level errors)			
301 (TR0100)	302 (TR0101)	303 (TR0102)	304 (TR0103)
305 (TR0104)	306 (TR0821)	307 (TR0818)	308 (TR0820)
309 (TR0822)	310 (TR0825)	311 (TR0826)	312 (TR0830)

Table A-5 (Page 2 of 2). Unique Error Codes from Translator

Code (msg)	Code (msg)	Code (msg)	Code (msg)
313 (TR0834)	314 (SA0042)	315 (TR0105)	316 (TR0106)
317 (TR0107)	318 (TR0108)	319 (TR0109)	320 (TR0110)
321 (TR0111)	322 (TR0112)	323 (TR0850)	324 (TR0113)
325 (TR0114)	326 (TR0025)	327 (TR0115)	328 (TR0116)
329 (TR0848)	330 (TR0117)	331 (TR0118)	332 (TR0119)
333 (TR0120)	334 (TR0121)	335 (TR1257)	336 (TR1258)
337 (TR1259)	338 (TR1260)	339 (TR1261)	340 (TR1262)
341 (TR0122)			
Note: Group Level Errors (except for unique codes 401 and 403 which are Interchange level errors)			
401 (TR0150)	402 (TR0151)	403 (TR0152)	404 (TR0153)
405 (TR0154)	406 (TR0155)	407 (TR0156)	408 (TR0157)
409 (TR0107)	410 (TR0108)	411 (TR0158)	412 (TR1257)
413 (TR1258)	414 (TR1259)	415 (TR1260)	416 (TR1261)
417 (TR1262)			
Note: Interchange Level Errors			
501 (TR0201)	502 (TR0203)	503 (TR0205)	504 (TR0206)
505 (TR0824)	901 (TR0810)	902 (TR0811)	903 (TR0812)
904 (TR0815)	905 (TR0816)	906 (TR0817)	907 (TR0843)
908 (TR0827)	909 (TR0828)	910 (TR0829)	911 (TR0832)
912 (TR0836)	913 (TR0838)	914 (TR0839)	915 (TR0840)
916 (TR1201)	917 (TR1202)	918 (TR1203)	919 (TR1205)
920 (TR1206)	921 (SA0042)	922 (TR0846)	923 (TR0847)
924 (TR0848)	925 (TR0849)	926 (TR0851)	927 (TR1207)
928 (TR1208)	929 (TR1209)	930 (TR1252)	931 (TR1253)
932 (TR1254)	933 (TR1255)	934 (TR1256)	935 (TR1257)
936 (TR1258)	937 (TR1259)	938 (TR1260)	939 (TR1261)
940 (TR1262)	941 (TR1263)		

Translator Input Data Block (TRIDB)

The translator input data block (TRIDB) is required on all requests for transaction services, enveloping services, and data extraction services. For more information on these services, see “Translation Services” on page 3-20, “Enveloping Services” on page 3-69, and “Data Extraction Services” on page 3-101.

Table A-6 on page A-30 shows how the block is used for all functions and what the initialization requirements are.

Note: When DataInterchange had a 32 K limit on transactions and application data, TRIDB used 2-byte length values. When the 32 K limit was removed, a new format for TRIDB using 4-byte length fields was created. Your application indicates the type of TRIDB that is being used in the **BLKTYPE** field of the TRCB.

Table A-6. TRIDB Functions and Initialization Requirements

Function	Reference	Initialization Requirement
1	"API - Retrieve Interchange Header" on page 3-99	TRIDB with a BLKLEN of 16 is sufficient.
2	"API - Retrieve Group Header" on page 3-100	TRIDB with a BLKLEN of 16 is sufficient.
3	"API - Retrieve Transaction Header" on page 3-100	TRIDB with a BLKLEN of 16 is sufficient.
131	"API - Translate to Standard" on page 3-27	TRIDB contains the application data within the DATA field. The DATALEN field contains the length of the data in the DATA field. The block has no minimum length and it can be as large as necessary to hold the application data.
111	"API - Translate to Standard" on page 3-27	TRIDB contains the application data within the DATA field. The DATALEN field contains the length of the data in the DATA field. The block has no minimum length and it can be as large as necessary to hold the application data.
211	"API - Translate File" on page 3-57	TRIDB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000 but should be large enough to hold the largest segment, excluding the binary segment.
212	"API - Translate File" on page 3-57	TRIDB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000 but should be large enough to hold the largest segment, excluding the binary segment.
213	"API - Translate Specific" on page 3-49	TRIDB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000 but should be large enough to hold the largest segment, excluding the binary segment.
214	"Deenvelope Function" on page 3-86	TRIDB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000 but should be large enough to hold the largest segment, excluding the binary segment.
215	"Envelope Function" on page 3-72	TRIDB with a BLKLEN of 16 is sufficient.
216	"API - Retrieve Detailed Data" on page 3-102	TRIDB with a BLKLEN of 16 is sufficient.
217	"API - Retrieve Transaction Image" on page 3-104	TRIDB with a BLKLEN of 16 is sufficient.
218	"API - Retrieve Transaction Acknowledgment Image" on page 3-105	TRIDB with a BLKLEN of 16 is sufficient.
219	"API - Retrieve FA Image" on page 3-105	TRIDB with a BLKLEN of 16 is sufficient.
990	"API - Close and Queue Interchange" on page 3-84	TRIDB with a BLKLEN of 16 is sufficient.
991	"API - Issue Commit" on page 3-98	TRIDB with a BLKLEN of 16 is sufficient.
1000	"API - End Translation/Enveloping" on page 3-85	TRIDB with a BLKLEN of 16 is sufficient.

Table A-7 on page A-31 and Table A-8 on page A-31 show the TRIDB definition with 2- and 4-byte lengths, respectively.

Table A-7. TRIDB Definition with 2-Byte Lengths

Name	Type	Offset	Length	Description
BLKLEN	Binary	0	2	Block length
RESERVED	Binary	2	2	Reserved
BLKNME	Character	4	8	Block name
DATALEN	Binary	12	2	Length of data in DATA field
DATA	Character	14	??	Application data for functions 111 and 131

Table A-8. TRIDB Definition with 4-Byte Lengths

Name	Type	Offset	Length	Description
BLKLEN	Binary	0	4	Block length
RESERVED	HEX	4	8	Reserved
DATALEN	Binary	12	4	Length of data in DATA field
DATA	Character	16	??	Application data for functions 111 and 131

TRIDB Field Descriptions

The following list describes the TRIDB fields.

- BLKLEN** The length of the TRIDB.
- RESERVED** Reserved. Initialize with binary zeros.
- BLKNME** The name of the block, which is EDITRIN.
- DATALEN** For functions 111 and 131, the amount of data being provided in the **DATA** field. For more information about providing partial data, see “Providing a Partial Structure” on page 3-41.
- DATA** For functions 111 and 131, the application data that should be translated. The format of the data must match the definition in the application data format.

Translator Output Data Block (TRODB)

The translator output data block (TRODB) is required on all requests for transaction services, enveloping services, and data extraction services. For more information on these services, see “Translation Services” on page 3-20, “Enveloping Services” on page 3-69, and “Data Extraction Services” on page 3-101.

Table A-9 on page A-32 shows how the block is used for all functions and what the initialization requirements are.

TRODB always has a minimum length of 32000. A value less than 32000 in the **BLKLEN** field results in a TR08291 error message, and the translator terminates. There is no maximum length for the TRODB.

Note: When DataInterchange had a 32 K limit on transactions and application data, TRODB used 2-byte length values. When the 32 K limit was removed, a new format for TRODB using 4-byte length fields was created. Your application indicates the type of TRODB that is being used in the **BLKTYPE** field of the TRCB.

Table A-9 (Page 1 of 2). TRODB Functions and Initialization Requirements

Function	Reference	Initialization Requirements
1	"API - Retrieve Interchange Header" on page 3-99	DATA field contains the interchange header image and DATALEN contains the number of bytes in the header segment.
2	"API - Retrieve Group Header" on page 3-100	DATA field contains the group header image and DATALEN contains the number of bytes in the header segment.
3	"API - Retrieve Transaction Header" on page 3-100	DATA field contains the transaction header image and DATALEN contains the number of bytes in the header segment.
131	"API - Translate to Standard" on page 3-27	TRODB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000 but should be large enough to hold the largest segment, excluding the binary segment.
111	"API - Translate to Standard" on page 3-27	TRODB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000 but should be large enough to hold the largest segment, excluding the binary segment.
211	"API - Translate File" on page 3-57	TRODB contains the application data within the DATA field. The DATALEN field contains the length of the data in the DATA field.
212	"API - Translate File" on page 3-57	TRODB contains the application data within the DATA field. The DATALEN field contains the length of the data in the DATA field.
213	"API - Translate Specific" on page 3-49	TRODB contains the application data within the DATA field. The DATALEN field contains the length of the data in the DATA field.
214	"Deenvelope Function" on page 3-86	TRODB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000 but should be large enough to hold the largest segment, excluding the binary segment.
215	"Envelope Function" on page 3-72	TRODB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000, but should be large enough to hold the largest segment, excluding the binary segment.
216	"API - Retrieve Detailed Data" on page 3-102	The DATA field contains the data extraction detail record and the DATALEN field contains the number of bytes in the record.
217	"API - Retrieve Transaction Image" on page 3-104	The DATA field contains the image record and the DATALEN field contains the number of bytes in the record.
218	"API - Retrieve Transaction Acknowledgment Image" on page 3-105	The DATA field contains the image record and the DATALEN field contains the number of bytes in the record.
219	"API - Retrieve FA Image" on page 3-105	The DATA field contains the image record and the DATALEN field contains the number of bytes in the record.

Table A-9 (Page 2 of 2). TRODB Functions and Initialization Requirements

Function	Reference	Initialization Requirements
990	"API - Close and Queue Interchange" on page 3-84	TRODB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000, but should be large enough to hold the largest segment, excluding the binary segment.
991	"API - Issue Commit" on page 3-98	TRODB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000, but should be large enough to hold the largest segment, excluding the binary segment.
1000	"API - End Translation/Enveloping" on page 3-85	TRODB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000, but should be large enough to hold the largest segment, excluding the binary segment.

Table A-10 and Table A-11 show the TRIDB definition with 2- and 4-byte lengths, respectively.

Table A-10. TRODB Definition with 2-Byte Lengths

Name	Type	Offset	Length	Description
BLKLEN	Binary	0	2	Block length
RESERVED	Binary	2	2	Reserved
BLKNME	Character	4	8	Block name
DATALEN	Binary	12	2	Length of data in the DATA field
DATA	Character	14	??	Application data for functions 211, 212, and 213

Table A-11. TRODB Definition with 4-Byte Lengths

Name	Type	Offset	Length	Description
BLKLEN	Binary	0	4	Block length
RESERVED	HEX	4	8	Reserved
DATALEN	Binary	12	4	Length of data in the DATA field
DATA	Character	16	??	Application data for functions 211, 212, and 213

TRODB Field Descriptions

The following list describes the TRODB fields.

- BLKLEN** Specifies the length of the TRODB. The minimum length is 32000.
- RESERVED** Reserved and should be initialized with binary zeros.
- BLKNME** Specifies the name of the block, which is EDITROUT.
- DATALEN** For functions 211, 212, 213, 216, 217, 218, and 219, the amount of data provided in the **DATA** field.
- DATA** For functions 211, 212, and 213, the application data that has been produced. The format of the data must match the definition in the application data format.
For functions 216, 217, 218, and 219, the detail extraction records.

Communication Interface Control Block (CMCB)

Table A-12 describes the fields in the communication interface control block (CMCB). The offset values in this table are relative to 0. If this control block is used in network operations (NETOP profile), 1 should be added to the offset value, because the offsets used for NETOPs are relative to 1.

Table A-12 (Page 1 of 2). Definition of Communication Interface Control Block

Field	Type	Offset	Length	Description
BLKLEN	Binary	0	2	Length of the CMCB
RESERV1	Binary	2	2	Reserved
BLKNME	Character	4	8	Identifies this block
TPNICKNM	Character	12	16	Trading partner nickname
NETID	Character	28	8	Network ID
NETOP	Character	36	8	Network operation
REQID	Character	44	16	Requestor ID
SEQNUM	Character	60	5	Msg/file/transaction sequence number
CONTRCV	Character	65	1	Continuous receive flag
FTYPE	Character	66	2	File type
FILERCV	Character	68	1	File received flag
RESERV2	Character	69	5	Reserved
CLRFILE	Character	74	1	Clear file indicator
DATAFMT	Character	75	1	Format of data being sent
ACCTYP	Character	76	1	Account type
DATATYP	Character	77	1	Data type
RECVTYP	Character	78	1	Type of receive
DCIND	Character	79	1	Delivery class
ACKIND	Character	80	1	Acknowledgment code
RESRECL	Character	81	1	Resolution of RECL
SCRIPT	Character	82	8	Script name
ENAME	Character	90	8	Envelope name
MSGNAME	Character	98	8	Message name
FILENAME	Character	106	56	File name
CANSD	Character	162	6	Cancel start date
CANST	Character	168	6	Cancel start time
CANED	Character	174	6	Cancel end date
CANET	Character	180	6	Cancel end time
TMZONE	Character	186	1	Time zone
MODEM	Character	187	1	Modem type
NPSSCDE	Character	188	5	Start session response code
NPESCDE	Character	193	5	End session response code

Table A-12 (Page 2 of 2). Definition of Communication Interface Control Block

Field	Type	Offset	Length	Description
NPERRCD	Character	198	5	Error code
NPSEVER	Character	203	2	Error severity
BLKTYPE	Character	205	1	Block type
FQUEUED	Character	206	1	Queued functions indicator
FMSGs	Character	207	1	Free-form messages indicator
FFILE	Character	208	1	Free-form files indicator
FEDIX	Character	209	1	EDI ISA/IEA file indicator
FEDIE	Character	210	1	EDI UNB/UNZ file indicator
FEDIU	Character	211	1	EDI BG/EG file indicator
FEDIG	Character	212	1	EDI GS/GE file indicator
FEDII	Character	213	1	EDI ICS/ICE file indicator
FEDIT	Character	214	1	EDI STX/END file indicator
FCANCEL	Character	215	1	CANCEL indicator
FCLASS	Character	216	1	Message class indicator
FAK	Character	217	1	Network acknowledgment indicator
FSYSMSG	Character	218	1	System messages indicator
FRCVBTP	Character	219	1	Receive by trading partner indicator
FRESTART	Character	220	1	Restart indicator
FNOUSERID	Character	221	1	Account number indicator
FACCTSEP	Character	222	1	Value to separate account and user ID
DDCOLON	Character	223	3	DD: string
RESERV3	Character	226	2	Reserved
ADMTYPE	Character	228	2	Admin response file type
UNIQID	Character	230	8	Unique ID returned by the network program
SAPUPDT	Character	238	1	SAP update flag
FSENTNET	Character	239	1	Skip send requested status
RESERV4	Character	240	14	Reserved for DataInterchange

CMCB Field Descriptions

The following descriptions cover only the fields that apply to the application program interface. Your definition, however, must include all fields in the block. The descriptions apply to all functions for which the block is used, unless exceptions are stated.

The following list describes the CMCB fields.

BLKLEN	Length of CMCB. A 2-byte binary field that contains the length of the CMCB control block. A CMCB is 254 bytes.
RESERV1	Reserved.

BLKNME	The name of this block, EDICMCB.								
TPNICKNM	The key for a member of the trading partner profile. For receiving, communications supplies the nickname for the sender of the first file received, if your application does not supply it.								
NETID	The key for a member of the network profile. Your application must supply the network ID, if it does not supply a requestor ID. If the application supplies a requestor ID, communications gets the network ID from the requestor profile and ignores this field. See REQID on page A-36 for related information.								
NETOP	Valid NETOP names are: SENDFILE Sends nonstandard file for function codes 0121 and 0221 SENDX12 Sends X12 standard transactions for function code 0211 SENDEDI Sends EDIFACT or UNTDI standard transactions for function code 0211 SENDUCS Sends UCS standard transactions for function code 0211 RCVFILE Receives nonstandard file for function codes 0132 and 0232 RCVX12 Receives X12 standard transactions for function codes 0132 and 0232 RCVEDI Receives EDIFACT or UNTDI standard transactions for function codes 0132 and 0232 RCVUCS Receives UCS standard transactions for function codes 0132 and 0232 CANCEL Cancels delivery of a file or message for function codes 0133 and 0233 NO-NETOP Used in function 252 to indicate that a file of responses should be processed but that no network program should be invoked. Used if you have network acknowledgment responses in a file that have not been processed.								
REQID	The key for a member of the requestor profile. Communication gets the network ID from the requestor profile member. Your application must supply the network ID if it does not supply the requestor ID. See NETID on page A-36 for related information.								
SEQNUM	The sequence number assigned to a transaction, file, or message. For sending or queuing to send, communications returns the sequence number. For canceling a file or message, your application supplies the sequence number of the file or message to be canceled.								
CONTRCV	In IINCICS environments, valid values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>(blank)</td><td>Single receive</td></tr> <tr> <td>C</td><td>Start continuous receive</td></tr> <tr> <td>E</td><td>End continuous receive</td></tr> </table>	Value	Description	(blank)	Single receive	C	Start continuous receive	E	End continuous receive
Value	Description								
(blank)	Single receive								
C	Start continuous receive								
E	End continuous receive								

FTYPE	<p>FTYPE indicates where received transactions are delivered. Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>MQ</td><td>MQSeries Queue</td></tr> <tr> <td>TM</td><td>Temporary storage queue (main)</td></tr> <tr> <td>TS</td><td>Temporary storage queue (auxiliary)</td></tr> <tr> <td>PG</td><td>Program</td></tr> </table>	Value	Description	MQ	MQSeries Queue	TM	Temporary storage queue (main)	TS	Temporary storage queue (auxiliary)	PG	Program
Value	Description										
MQ	MQSeries Queue										
TM	Temporary storage queue (main)										
TS	Temporary storage queue (auxiliary)										
PG	Program										
FILERCV	Y indicates that a file was received.										
RESERV2	Reserved.										
CLRFILE	<p>Indicates when communications is to clear the file you are sending. Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Clears the file after successfully sending the transaction data or nonstandard file</td></tr> <tr> <td>U</td><td>Clears the file unconditionally</td></tr> </table> <p>This request applies only to immediate sending of standard transactions and nonstandard files.</p>	Value	Description	Y	Clears the file after successfully sending the transaction data or nonstandard file	U	Clears the file unconditionally				
Value	Description										
Y	Clears the file after successfully sending the transaction data or nonstandard file										
U	Clears the file unconditionally										
DATAFMT	<p>DATAFMT applies only if your trading partner uses the IBM Global Network PC/IE product to receive the data. The first character of the envelope name field defines the receive class used by the PC/IE interface. If it is B, PC/IE assumes binary data and decodes it accordingly. If the first character is A or E, PC/IE does not decode the data.</p> <p>If the machine type field in the trading partner profile indicates that your trading partner is using a PC to receive the data, communications shifts the ENAME value right one position and sets the first character to B or A, depending on the value you supply in this field. See ENAME on page A-39 for related information. Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>B</td><td>Binary data</td></tr> <tr> <td>(blank)</td><td>Nonbinary data</td></tr> </table>	Value	Description	B	Binary data	(blank)	Nonbinary data				
Value	Description										
B	Binary data										
(blank)	Nonbinary data										
ACCTYP	<p>For sending or canceling data, your application must supply the D code, which indicates that the trading partner ID is an account number or user ID (IBM Global Network reference: DESTTYP). For receiving data, your application must supply one of the following values (IN reference: SRCTYP):</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>(blank)</td><td>Receives from any source</td></tr> <tr> <td>D</td><td>Receives from a trading partner identified by an account number or user ID</td></tr> </table>	Value	Description	(blank)	Receives from any source	D	Receives from a trading partner identified by an account number or user ID				
Value	Description										
(blank)	Receives from any source										
D	Receives from a trading partner identified by an account number or user ID										
DATATYP	<p>The type of file name supplied in the FILENAME field. Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>A</td><td>Data set name</td></tr> <tr> <td>D</td><td>ddname</td></tr> </table> <p>For sending, your application must supply the data type. You can use a ddname only for immediate sends.</p> <p>For receiving, communications supplies the data type if all the following are true:</p> <ul style="list-style-type: none"> • The application does not supply the data type. • NETOP is RECVX12, RECVEDI, or RECVUCS. • The function requested is receive immediate for function code 0232. 	Value	Description	A	Data set name	D	ddname				
Value	Description										
A	Data set name										
D	ddname										

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	DATATYP
IINB1	FILEID
IINB41	FILEID

RECVTYP The type of messages to receive. Valid values for receiving are:

Value	Description
-------	-------------

G	Receives only the first message or file that meets the receive criteria, which is the trading partner nickname, the envelope name (ENAME), or both. For example, receive only X12 (ENAME) from partner ABC.
---	---

(blank)	Receives all messages and files that meet the receive criteria.
---------	---

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	TPYRCV
IINB1	ALLFILES
IINB41	ALLFILES

DCIND The delivery class. Valid values for sending and canceling data are:

Value	Description
-------	-------------

(blank)	Normal priority store and forward
---------	-----------------------------------

P	High-priority store and forward
---	---------------------------------

I	Express delivery
---	------------------

For IBM networks, delivery class I requires the receiving partner to be signed on to the network when the data is sent.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	MSGDCLS
IINB1	PRIORITY
IINB41	PRIORITY

ACKIND The type of acknowledgment. The valid values for sending are:

Value	Description
-------	-------------

(blank)	Requests no acknowledgments
---------	-----------------------------

R	Requests acknowledgment for receipt
---	-------------------------------------

D	Requests acknowledgment for delivery
---	--------------------------------------

B	Requests acknowledgment for both receipt and delivery
---	---

A	Requests acknowledgment for purge
---	-----------------------------------

C	Requests acknowledgment for both receipt and purge
---	--

E	Requests acknowledgment for either receipt or purge
---	---

F	Requests acknowledgment for receipt and either delivery or purge
---	--

(binary zero)	
---------------	--

Uses the network acknowledgment code (**NETACK**) from the trading partner profile and returns it in this field

Valid values for canceling are:

Value	Description
(blank)	Requests no acknowledgments
H	Requests only header information in the acknowledgment
T	Requests both header and text information in the acknowledgment

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	MSGRCPT
IINB1	ACK
IINB41	ACK

RESRECL For IINR3, valid values for receiving are:

Value	Description
(blank)	Ends the job step with an A03 system abend code if the length of the output record is greater than the length of the data set record
S	Splits the output records to fit the length defined for the data set records

If the storage format (STGFRMT in the trading partner profile) is C or D, the problem does not occur. In these cases, the length of the output record is set to the length of the data set record.

For IINB1 and IINB41, valid values for receiving are:

Value	Description
E	Ends the session with an error if the length of the output record is greater than the length of the data set record
S	Splits the output records to fit the length defined for the data set records

If the storage format (STGFRMT in the trading partner profile) is C or D, the problem does not occur. In these cases, the length of the output record is set to the length of the data set record.

The Expedite reference is **RESRECL**.

SCRIPT Specifies the script name. This field can be used by communication software to specify a set of instructions to follow when processing requests for service. The set of instructions would be part of the communication software package and not part of DataInterchange.

ENAME The envelope name. For sending and canceling, if your application supplies a null value (binary zero), communications uses the message user class from the requestor profile and returns it here. If the requestor profile does not supply a value, a value of #IDI# is used by default.

For receiving, if your application supplies a null value, communications uses the message user class from the requestor profile and returns the message user class of the first file received.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	MSGUCLS
IINB1	ACK
IINB41	ACK

MSGNAME	<p>The message name. Any value you define that fits the field length, or blanks.</p> <p>For sending or canceling, your application supplies the message name. If it does not supply a value, a value of #IDI# is used by default.</p> <p>For receiving, communications returns the message name of the first file received.</p> <p>The Expedite reference is MSGNAME.</p>								
FILENAME	<p>The name of a file containing data to be sent or into which data is to be received from the network. The data type field (DATATYP) indicates whether the name is a data set name or a ddname.</p> <p>Your application must supply the file name for sending nonstandard files (function code 0121 and 0221). You can use a ddname only for immediate sends. For sending standard transactions, the file name is optional. FILENAME applies to function code 0211 for sending standard files. If the file name is blank, communications uses the ddname from the transaction data queue field in the network profile member. An E is appended to the ddname for send EDIFACT or UNTDI. A U is appended to the ddname for send UCS. X12 requests use the ddname as supplied. Communications also sets the data type (DATATYP) to D.</p> <p>For receiving nonstandard files, your application must supply the file name. You can use a ddname only for immediate receives. For receiving standard transactions, the file name is optional. If the file name is blank, communications uses the ddname from the receive file name field in the requestor profile member. DataInterchange also sets the data type to D.</p> <p>Expedite parameters for network profiles are:</p> <table data-bbox="354 1018 893 1165"> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>FILESPEC</td></tr> <tr> <td>IINB1</td><td>FILEID</td></tr> <tr> <td>IINB41</td><td>FILEID</td></tr> </table>	Network Profile ID	Expedite Parameter	IINR3	FILESPEC	IINB1	FILEID	IINB41	FILEID
Network Profile ID	Expedite Parameter								
IINR3	FILESPEC								
IINB1	FILEID								
IINB41	FILEID								
CANS	<p>The cancellation start date in <i>yyymmdd</i> format. CANS is optional and applies only to canceling files and messages.</p> <p>Expedite parameters for network profiles are:</p> <table data-bbox="354 1302 893 1438"> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>SUBDATE</td></tr> <tr> <td>IINB1</td><td>STARTDATE</td></tr> <tr> <td>IINB41</td><td>STARTDATE</td></tr> </table>	Network Profile ID	Expedite Parameter	IINR3	SUBDATE	IINB1	STARTDATE	IINB41	STARTDATE
Network Profile ID	Expedite Parameter								
IINR3	SUBDATE								
IINB1	STARTDATE								
IINB41	STARTDATE								
CANST	<p>The cancellation start time in <i>hhmmss</i> format. CANST is optional and applies only to canceling files and messages.</p> <p>Expedite parameters for network profiles are:</p> <table data-bbox="354 1579 893 1715"> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>SUBTIME</td></tr> <tr> <td>IINB1</td><td>STARTTIME</td></tr> <tr> <td>IINB41</td><td>STARTTIME</td></tr> </table>	Network Profile ID	Expedite Parameter	IINR3	SUBTIME	IINB1	STARTTIME	IINB41	STARTTIME
Network Profile ID	Expedite Parameter								
IINR3	SUBTIME								
IINB1	STARTTIME								
IINB41	STARTTIME								
CANED	<p>The cancellation end date in <i>yyymmdd</i> format. CANED is optional and applies only to canceling files and messages.</p> <p>The Expedite reference is ENDDATE.</p>								

CANET	The cancellation end time in <i>hhmmss</i> format. CANET is optional and applies only to canceling files and messages. The Expedite reference is ENDTIME .						
TMZONE	Indicates the time zone for cancellation requests. Valid values are: <table> <tr> <th>Code</th><th>Description</th></tr> <tr> <td>L</td><td>Local time</td></tr> <tr> <td>G</td><td>Greenwich Mean Time</td></tr> </table> The Expedite reference is TIMEZONE .	Code	Description	L	Local time	G	Greenwich Mean Time
Code	Description						
L	Local time						
G	Greenwich Mean Time						
MODEM	The type of modem used.						
NPSSCDE	The network program start-session response code.						
NPESCDE	The network program end-session response code.						
NPERRCD	The network program error code.						
NPSEVER	The network program error severity code.						
BLKTYPE	The type of data blocks passed to communications for sending messages. H indicates data blocks that are larger than 32 K bytes. Any other character indicates data blocks of 32 K bytes or less. BLKTYPE applies only to sending messages using function codes 0141 and 0241.						
FQUEUED	Y indicates that the network supports queued functions.						
FMSGGS	Y indicates that the network supports free-form messages.						
FFILE	Y indicates that the network supports nonstandard files.						
FEDIX	Y indicates that the network supports X12 ISA/IEA envelopes.						
FEDIE	Y indicates that the network supports EDIFACT UNB/UNZ envelopes.						
FEDIU	Y indicates that the network supports BG/EG envelopes.						
FEDIG	Y indicates that the network supports GS/GE envelopes.						
FEDII	Y indicates that the network supports ICS/ICE envelopes.						
FEDIT	Y indicates that the network supports STX/END envelopes.						
FCANCEL	Y indicates that the network supports the CANCEL function.						
FCLASS	Y indicates that the network supports user message classes.						
FAACK	Y indicates that the network supports network acknowledgments.						
FSYSMSG	Y indicates that the network supports system messages.						
FRCVBTP	Y indicates that the network supports receiving by trading partner.						
FRESTART	Y indicates that the network supports restart.						
FNOUSERID	Y indicates that the network supports account numbers only.						
FACCTSEP	The character used to separate the account number and user ID.						
DDCOLON	The text string DD:						
RESERV3	Reserved.						
ADMTYPE	Type of administrative response file for CICS ID.						
UNIQID	A unique ID returned by the network program.						
SAPUPDT	Y indicates that VANI is to update SAP (for DataInterchange use only).						

FSENTNET	Y indicates the Send requested status should be skipped and the status set directly to Sent to network. This is the case with network program EDIMQSR (the MQSeries network program). The default is N.
RESERV4	Reserved.

Trading Partner Profile Block (TPPDB)

The following descriptions include only the fields that apply to the application program interface. However, the block definition must include all fields in the block. All of the following fields are optional. If a value is present, it is used in place of the corresponding value in the trading partner profile. If you supply **TPNICKNM** in the CMCB and leave it blank in this block, communications returns a value for the fields that apply to the request.

Table A-13 shows the definition of the trading partner profile block. The offset values in this table are relative to 0. If this control block is used in network operations (NETOP profile), 1 should be added to the offset value because the offsets used for NETOPs are relative to 1.

Table A-13 (Page 1 of 3). Definition of the Trading Partner Profile Block

Field	Type	Offset	Length	Description
BLKLEN	Binary	0	2	Length of TPPDB
RESERV1	Binary	2	2	Reserved
BLKNME	Character	4	8	Identifies the TPPDB
TPNICKNM	Character	12	16	Trading partner nickname
NETID	Character	28	8	Network ID
SYSQUAL	Character	36	1	Intersystem address qualifier
SYSID	Character	37	8	Intersystem ID
ACCTNUM	Character	45	32	Requestor's network account number
USERID	Character	77	32	Requestor's network user ID
ENVLQUAL	Character	109	4	Interchange qualifier
ENVLID	Character	113	35	Interchange sender/receiver ID
CONAME	Character	148	40	Company name
ADDR1	Character	188	40	Company address line 1
ADDR2	Character	228	40	Company address line 2
PHONE	Character	268	25	Contact phone number
CONTACT	Character	293	30	Contact name
PASSWORD	Character	323	14	Interchange password for send
RCVPASS	Character	337	14	Interchange password for receive
SECUID	Character	351	8	Security profile member
NETCLS	Character	359	1	Network message class
NETCHG	Character	360	1	Network charges code
NETACK	Character	361	1	Network acknowledgment code
NETVCHK	Character	362	1	Destination verification code
NETRETN	Character	363	3	Mailbox retention period

Table A-13 (Page 2 of 3). Definition of the Trading Partner Profile Block

Field	Type	Offset	Length	Description
NETEDIO	Character	366	1	Option for storing received data
NETEDIP	Character	367	1	Special processing for received data
STGFRMTO	Character	368	1	Storage format override
MACHTYPE	Character	369	1	Machine type
STGFRMT	Character	370	1	Storage format
EOTID	Character	371	1	End of text/message delimiter
LOGENV	Character	372	1	Log envelope data
FNGRPENV	Character	373	1	Send functional group
SEDELIM	Character	374	1	Sub-element delimiter
DEDELIM	Character	375	1	Data Element delimiter
SGDELIM	Character	376	1	Segment delimiter
SGSEP	Character	377	1	Segment ID separator
DECNOT	Character	378	1	Decimal notation
RLSCHAR	Character	379	1	Release character
TPICTLNO	Character	380	9	Interchange mask
TPGCTLNO	Character	389	9	Group mask
TPTCTLNO	Character	398	9	Transaction mask
COMMENT1	Character	407	40	Comment line 1
COMMENT2	Character	447	40	Comment line 2
NETCMD5	Character	487	8	Net commands PDS member
TPDATA LINE	Character	495	32	Data line phone number
TIMEOUT	Character	527	4	Communications line timeout value
SEGMENTED	Character	531	1	Segmented output
SUFFIX	Character	532	2	File suffix
TPENV SUF	Character	534	2	Envelope prof member suffix
TPGENRCV	Character	536	1	Generic receive usages allowed
TPCMPRES	Character	537	1	Compress flag
TPRSRV1	Character	538	8	Reserved for DataInterchange
TPSUPAD3	Character	546	40	Company address line 3
TPSUPCTY	Character	586	30	City name
TPSUPST	Character	616	2	State code
TPSUPPST	Character	618	15	Postal code
TPSUPCON	Character	633	30	Country code
TPSUPFAX	Character	663	25	Fax number
TPSUPU3	Character	688	40	Comment line 3
TPSUPU4	Character	728	40	Comment line 4
TPSUPU5	Character	768	40	Comment line 5
TPSUPU6	Character	808	40	Comment line 6

Table A-13 (Page 3 of 3). Definition of the Trading Partner Profile Block

Field	Type	Offset	Length	Description
TPSUPU7	Character	848	40	Comment line 7
TPSUPU8	Character	888	40	Comment line 8
TPSUPU9	Character	928	40	Comment line 9
TPSUPU10	Character	968	40	Comment line 10
PRIORITY	Character	1008	1	Delivery priority
TPRSRV2	Character	1009	3	Reserved for DataInterchange
DESCRIPT	Character	1012	30	Profile member description
LOGLOCK	Character	1042	1	Logical lock flag
LASTUID	Character	1043	17	Last update user ID
LASTUDT	Binary	1060	4	Last update date/time
TPTYPE	Character	1064	1	Trading partner type
TPRSRV3	Character	1065	467	Reserved for DataInterchange

TPPDB Field Descriptions

The following list describes the TPPDB fields that apply to the application program interface.

BLKLEN Length of TPPDB. A 2-byte binary field that contains the length of the TPPDB data block. A TPPDB is 1532 bytes.

RESERV1 Reserved.

BLKNME The name of this block.

TPNICKNM The name you use to refer to the trading partner. **TPNICKNM** must identify a member of the trading partner profile.

NETID The network ID. **NETID** must match the key field of a member of the network profile. An example is IN.

SYSQUAL For IBM Global Network, the value I if intersystem addressing is required for this trading partner (IN reference: **DTBLTYP**). Enter the ID of the other system in the **SYSID** field.

SYSID For intersystem addressing, the ID of the system responsible for the receiver's account. The ID is limited to 3 characters.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	DTBLID
IINB1	SYSID
IINB41	SYSID

ACCTNUM For IBM networks, the account number that the network assigns to the trading partner. The entry must be left-justified. For sending and receiving standard transactions using ISA/IEA envelopes, the last position must be blank. The combination of this field and the **USERID** field must be unique. You must provide this field if you want to use network acknowledgments. **ACCTNUM** applies to sending or receiving nonstandard files and messages.

USERID	For IBM networks, the user ID that the network assigns your trading partner. The entry must be left-justified. The combination of this field and the account number must be unique. You must provide this field if you want to use network acknowledgments. Together, the account number and user ID make up the trading partner ID in the interchange envelope. UCS (BG/EG) envelopes are an exception. For UCS (BG/EG), the phone number is the trading partner ID in the envelope. USERID applies to sending or receiving nonstandard files and messages.														
ENVLQUAL	The type of interchange ID used in the ENVLID field. The EDI standard defines these codes. If this field or the interchange ID (ENVLID) is blank, the enveloper takes the qualifier from the envelope profile member.														
ENVLID	The ID used to fill in the interchange receiver (recipient) when you send to this partner, and to identify the interchange sender when you receive from this partner. If you leave this field blank, the enveloper uses the account number and user ID (or phone number for BG/EG interchanges).														
CONAME	The name of the trading partner's company. The company name may be used as envelope data by using the 'CO' envelope data type.														
ADDR1	Line 1 of the trading partner's address.														
ADDR2	Line 2 of the trading partner's address.														
PHONE	The trading partner's telephone number. For type U (BG/EG) enveloping, this phone number is used as the interchange receiver ID if the Interchange ID field is blank. When de-enveloping type U (BG/EG) interchanges, this phone number is used as the interchange sender ID if the Interchange ID field is blank.														
CONTACT	The name of the person you speak with when dealing with this trading partner.														
PASSWORD	The password that you and your trading partner agreed upon for sending to this trading partner. The PASSWORD value maps to the PW data type in the interchange envelope.														
RCVPASS	The password that you and your trading partner agreed upon for receiving from this trading partner. If this value matches the interchange password (indicated by the PW data type) that was received, translation occurs.														
SECUID	The name of the security profile member that specifies the encryption and authentication processes that apply to EDI data. For sending, the trading partner usage specifies the security profile member. If the send usage does not specify a member, the member specified here is used by default.														
NETCLS	Indicates any special status of the data being sent. Valid values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>blank</td><td>Normal status</td></tr> <tr> <td>T</td><td>Test status</td></tr> </table> <p>NETCLS applies to all send requests but does not apply to receive requests.</p> <p>Expedite parameters for network profiles are:</p> <table> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>MSGNCLS</td></tr> <tr> <td>IINB1</td><td>MODE</td></tr> <tr> <td>IINB41</td><td>MODE</td></tr> </table>	Value	Description	blank	Normal status	T	Test status	Network Profile ID	Expedite Parameter	IINR3	MSGNCLS	IINB1	MODE	IINB41	MODE
Value	Description														
blank	Normal status														
T	Test status														
Network Profile ID	Expedite Parameter														
IINR3	MSGNCLS														
IINB1	MODE														
IINB41	MODE														
NETCHG	Indicates how charges are shared between sender and receiver. Valid values are:														

Value	Description
1	Specifies the receiver pays all charges.
2	Specifies the receiver pays all charges if agreed to; or, charges are split between sender's and receiver's domains.
3	Specifies receiver pays all charges if agreed to; or, charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges. This is the default.
4	Specifies charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges.
5	Specifies charges are split between sender's and receiver's domains.
6	Specifies sender pays all charges.

NETCHG applies to all send requests but does not apply to receive requests.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	MSGCHRG
IINB1	CHARGE
IINB41	CHARGE

NETACK Indicates which network acknowledgments are requested. The network specifies the acceptable values. Valid values are:

Value	Description
blank	Specifies no acknowledgments
R	Specifies the receipt
D	Specifies the delivery
B	Specifies both receipt and delivery
A	Specifies the purge
C	Specifies both the receipt and purge
E	Specifies either the receipt or purge
F	Specifies the receipt and either delivery or purge

NETACK applies to all send requests, but only if the acknowledgment indicator in the CMCB contains nulls (binary zeros). See **ACKIND** on page A-38 for related information.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	MSGRCPT
IINB1	ACK
IINB41	ACK

NETVCHK Indicates whether the destination is verified before sending occurs. Valid values are:

Value	Description
N	No verification (default)
Y	Require verification
F	Request verification and sending even if the destination is not verified (useful for intersystem addressing)

If your request does not specify a trading partner, the requestor profile provides this information.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	MSGVCHK
IINB1	VERIFY
IINB41	VERIFY

NETRETN The number of days that data is to be kept in the network mailbox before it is purged, if it is not received. Enter blanks or zeroes to use the default number. For Expedite Base/MVS, the valid range is 001-180. For Expedite/CICS, the valid range is 001-099, left-justified. If your request does not specify a trading partner, the requestor profile provides this information.

NETEDIO Indicates whether you want EDI segments stored in the receiving file as separate records. You provide the file name in the requestor profile. Valid values are:

Value	Description
Y	End records at the segment delimiter (default)
N	Does not end records at the segment delimiter

If your request does not specify a trading partner, the requestor profile provides this information.

The expedite reference is **EDIOPT**.

NETEDIP Indicates whether EDI data you receive has special EDI processing (breaking records by the segment delimiter). Valid values are:

Value	Description
Y	Performs EDI processing if the common data header indicates that the data is in standard format (default)
N	Omits EDI processing, regardless of the common data header

If your request does not specify a trading partner, the requestor profile provides this information.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	EDIPROC
IINB1	AUTOEDI
IINB41	AUTOEDI

STGFRMTO Indicates whether you want to use the storage format defined in the common data header. Valid values are:

Value	Description
Y	Uses the storage format as defined in the common data header (default)
N	Ignores the storage format defined in the common data header

If there is no common data header, the format indicated in the storage format field is used. If your request does not specify a trading partner, the requestor profile provides this information.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	STGFORMO
IINB1	DLMOVERRIDE
IINB41	DLMOVERRIDE

MACHTYPE The trading partner's machine type. If your trading partner is using the Personal Computer/Information Exchange (PC/IE) product to receive your data, enter 1. If not, leave the field blank.

STGFRMT Indicates to the network how data is stored for free-form messages and files.
Consider the type of data you want to send and how the file is received when determining what codes to select. Contact a representative from each network you are using for all available codes.

For example, if you are using:

- The IBM Expedite/MVS Base Version 1.1 (IEBASE), valid values are:

Value	Description
C	Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Select this option to send files containing program source code that is defined with variable length records. Output records do not include the carriage return and line-feed characters.
L	Indicates that each record is preceded by a 2-byte hexadecimal record length. Select this option when sending a data set defined in a fixed format, or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.
N	Stores data as it is received. Output records are built based on the record length of the data set allocated to receive the data.

- IBM Expedite/CICS, valid values are:

Value	Description
A	Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Select this option to send files containing program source code that is defined with variable length records. Output records do not include the carriage return and line-feed characters.
L	Indicates that each record is preceded by a 2-byte hexadecimal record length. Select this option when sending a data set defined in a fixed format or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.
O	Other. Free format.

- The IBM Expedite/MVS Host Version 1.3 (IEBASE), valid values are:

Value	Description
A	Stores each record with a carriage return and line-feed character and uses the end of file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Program source code defined with variable length records is the type of file generally sent with this option. Output records do not include the carriage return and line-feed characters.
B	Indicates that each record is preceded by a 2-byte hexadecimal record length. Select this option when sending a data set defined in a fixed format, or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.
C	Stores data as it is received. Output records are built based on the record length of the data set allocated to receive the data.

STGFRMT applies only to sending or receiving nonstandard files.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	STGFORM
IINB1	DELIMITED
IINB41	DELIMITED

EOTID The character that signifies to the network the end-of-message text. **EOTID** applies only to sending or receiving free-form messages.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	EORCHAR
IINB1	ENDSTR
IINB41	ENDSTR

LOGENV A Y specifies that standard data will be logged. The setting of this field controls the logging of standard data only when the Log Standard Data field of the APPDEFS profile member is not a Y or an N.

FNGRPENV A Y specifies that functional groups will be created for transactions with type E (UNB/UNZ) envelopes. Functional groups are always created for type I (ICS/ICE), U (BG/EG), and X (ISA/IEA) envelopes, and they are never created for type T (STX/END) envelopes.

SEDELIM The character that separates sub-elements (component data elements) in a transaction set. A value here (other than low-value or space) overrides the character specified in the standard. This value is only used when interchanges are created (not received).

DEDELIM The character that separates the data elements in a transaction set. A value here (other than low-value or space) overrides the character specified in the standard. This value is only used when interchanges are created (not received).

SGDELIM The character that marks the end of each segment in a transaction set. A value here (other than low-value or space) overrides the character specified in the standard. This value is only used when interchanges are created (not received).

SGSEP The character that separates the segment ID and the first data element in a segment for type E (UNB/UNZ) envelopes only. A value here (other than low-value or space) overrides the character specified in the standard. This value is only used when interchanges are created (not received).

DECNOT	The character that represents decimal points in a transaction set. For type E (UNB/UNZ) envelopes, a value here (other than low-value or space) overrides the character specified in the standard. For all other types, a period represents the decimal point. This value is only used when interchanges are created (not received).						
RLSCHAR	For type E (UNB/UNZ) and T (STX/END) envelopes, this is the character that indicates when a delimiter is being used as part of the data. A value here (other than low-value or space) overrides the character specified in the standard. This value is only used when interchanges are created (not received).						
TPICTLNO	The initial reference number that the enveloper maps to the CN data type in the interchange header and trailer. This value will be used as the base value for each trading partner, receiver ID combination. This field does not represent the current control number for this trading partner.						
TPGCTLNO	The initial reference number or special codes that the enveloper maps to the CN data type in the functional group header and trailer. This value will be used as the base value for each trading partner, receiver ID combination. This value does not represent the current control number for this trading partner.						
TPTCTLNO	The initial reference number or special codes that the enveloper maps to the CN data type in the transaction set header and trailer. This value will be used as the base value for each trading partner, receiver ID combination. This value does not represent the current control number for this trading partner.						
COMMENT1	A 40-byte area for free-form notes about the trading partner.						
COMMENT2	A 40-byte area for free-form notes about the trading partner.						
NETCMDS	Enter the name of a member of a PDS that will be allocated to the <i>ddname</i> of EDINTCMD. This member will contain the commands that you want to pass to a network. DataInterchange will read the commands from the PDS member and write the commands to the <i>Network input file</i> specified in the network profile member after all substitutable variable tags have been resolved by DataInterchange.						
TPDATA LINE	Enter the phone number to connect to your trading partner directly. This is the number needed by your computer to talk directly to your trading partner's computer. See the <i>Contact phone</i> for the voice number for you (not your computer) to call a person at your trading partner's site.						
TIMEOUT	Enter a value that will be used as a maximum allowable time that the data line for communications can be idle without being dropped. If you specify a trading partner when requesting network activity (send or receive), the value for this field is taken from the TP profile. Otherwise, the value for this field is taken from the requestor profile.						
SEGMENTED	Enter a code that indicates whether or not you want EDI segments to be stored in the output file as separate records. The values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>To end records at the segment delimiter</td></tr> <tr> <td>N</td><td>To not end at the segment delimiter (default)</td></tr> </table>	Value	Description	Y	To end records at the segment delimiter	N	To not end at the segment delimiter (default)
Value	Description						
Y	To end records at the segment delimiter						
N	To not end at the segment delimiter (default)						
SUFFIX	A two-character suffix used as a suffix for the <i>ddname</i> used to store the results of a fixed-to-fixed translation. The basic part of the <i>ddname</i> is taken from the Application file name field of the target application data format.						
TPENV SUF	A two-character suffix used as a suffix for a generic standard envelope profile member name. The basic part of the name is taken from the send or receive usage override.						

TPGENRCV	A code to indicate whether generic receive usages are allowed for this trading partner. The values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Allows generic receive usages</td></tr> <tr> <td>Other</td><td>Does not allow generic receive usages</td></tr> </table>	Value	Description	Y	Allows generic receive usages	Other	Does not allow generic receive usages		
Value	Description								
Y	Allows generic receive usages								
Other	Does not allow generic receive usages								
TPCMPRES	The Expedite Base/MVS compression code. The values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Specifies that compression is wanted</td></tr> <tr> <td>N</td><td>Specifies that compression is not wanted</td></tr> <tr> <td>T</td><td>Specifies that Expedite Base/MVS is to use its own table to decide whether or not to compress.</td></tr> </table> <p>Note: This field is only valid during send.</p>	Value	Description	Y	Specifies that compression is wanted	N	Specifies that compression is not wanted	T	Specifies that Expedite Base/MVS is to use its own table to decide whether or not to compress.
Value	Description								
Y	Specifies that compression is wanted								
N	Specifies that compression is not wanted								
T	Specifies that Expedite Base/MVS is to use its own table to decide whether or not to compress.								
TPRSRV1	Reserved for DataInterchange.								
TPSUPAD3	Line 3 of the trading partner's address.								
TPSUPCTY	The trading partner's city.								
TPSUPST	The trading partner's state code.								
TPSUPPST	The trading partner's postal code.								
TPSUPCON	The trading partner's country.								
TPSUPFAX	The trading partner's fax number.								
TPSUPU3	A 40-byte area for free-form notes about the trading partner.								
TPSUPU4	A 40-byte area for free-form notes about the trading partner.								
TPSUPU5	A 40-byte area for free-form notes about the trading partner.								
TPSUPU6	A 40-byte area for free-form notes about the trading partner.								
TPSUPU7	A 40-byte area for free-form notes about the trading partner.								
TPSUPU8	A 40-byte area for free-form notes about the trading partner.								
TPSUPU9	A 40-byte area for free-form notes about the trading partner.								
TPSUPU10	A 40-byte area for free-form notes about the trading partner.								
PRIORITY	Blank (for normal priority) or P (for high priority). These codes are used by Expedite Base/MVS and Expedite/CICS to prioritize delivery of messages. High priority messages will be delivered to your trading partners before normal priority messages.								
TPRSRV2	Reserved for DataInterchange.								
DESCRIPT	The description of this profile member.								
LOGLOCK	The profile member logical lock flag (for DataInterchange use only).								
LASTUID	The user ID of the last person to update this profile member.								
LASTUDT	The date and time that this profile member was last updated.								
TPTYPE	Trading partner type. An E specifies an EDI trading partner, which is a trading partner considered to be external to your business organization. An A specifies an application trading partner, which is a trading partner considered to be internal within your business								

| organization. A B specifies both an external (EDI) trading partner and an external (EDI) trading partner and an internal (application) trading partner. E is the default.

| **TPRSRV3** Reserved for DataInterchange.

Communication Data Block (DATBLK)

Use Table A-14 and Table A-15 when defining the DATBLK. There are two definitions for this block:

- Data blocks that are 32 K bytes or less
- Data blocks that are over 32 K bytes. Use the **BLKTYPE** field in the CMCB to indicate which definition you are using.

Table A-14. DATBLK Up to 32 K

Field	Type	Offset	Length	Description
BLKLEN	Binary	0	2	Length of data block
RESERV1	Binary	1	2	Reserved
BLKNME	Character	2	8	Identifies this block
DATALEN	Binary	10	2	Length of the following message
DATA	Character	12	Variable	Message text

Table A-15. DATBLK Over 32 K

Field	Type	Offset	Length	Description
BLKLEN	Binary	0	4	Length of data block
RESERV1	Character	2	8	Reserved
DATALEN	Binary	10	4	Length of following message
DATA	Character	12	Variable	Message text

DATBLK Field Descriptions

Your application assigns values to this block only when sending a message for function codes 0141 and 0241.

The following list describes the DATBLK fields.

BLKLEN Specifies the DATBLK length, including this field. The DATBLK length is either 14 or 16 bytes, plus the length of the message.

BLKNAME Specifies a name your application gives to the data block.

DATALEN Specifies the length of the message in the **DATA** field. The message must not exceed 32 K bytes unless you have indicated in the **BLKTYPE** field of the CMCB that you are sending data blocks that are larger than 32 K bytes.

DATA Specifies the message text. For IBM networks, your application must arrange the message text in 80-byte segments that begin with the letter T.

Network Profile Block (NPDB)

The offset values in Table A-16 are relative to 0. If this control block is used in network operations (NETOP profile), 1 should be added to the offset value because the offsets used for NETOPs are relative to 1.

Table A-16. Definition of the Network Profile Block (NPDB)

Field	Type	Offset	Length	Description
BLKLEN	Binary	0	2	Length of NPDB
RESERV1	Binary	2	2	Reserved
BLKNME	Character	4	8	Identifies the NPDB
NETID	Character	12	8	Network ID
NETNME	Character	20	30	Network name
COMROT	Character	50	8	Communication routine name
NETPGM	Character	58	8	Network program name
PGMPARM	Character	66	57	Network program parameters
CMDIN	Character	123	8	Network command input file
CMDLRECL	Character	131	4	Network command record length
QDATA	Character	135	8	Transaction data queue file
DATLRECL	Character	143	4	Transaction data record length
TMZONE	Character	147	5	Time zone
SYSTYP	Character	152	8	System type
SYSLVL	Character	160	4	System level
TXTHDR	Character	164	1	Message text header character
CMDOUT	Character	165	8	Network command output file
MSGROUT	Character	173	8	Program to process messages
SEQNUM	Character	181	5	Sequence number for network
NETACKFILE	Character	186	8	File where network acks are written
NETPHONE	Character	194	32	Dial connection phone number
SCRIPT	Character	226	8	Script name
FILLER	Character	234	14	Reserved for DataInterchange
DESCRIPT	Character	248	30	Member description
LOGLOCK	Character	278	1	Logical lock flag
LASTUID	Character	279	17	Last update user ID
LASTUDT	Binary	296	4	Last update date/time

NPDB Field Descriptions

The following list describes the NPDB fields.

BLKLEN	Length of NPDB. A 2-byte binary field that contains the length of the NPDB data block. An NPDB is 300 bytes.								
RESERV1	Reserved.								
BLKNME	The name of the network profile block, EDINPDB.								
NETID	The name (key) for referring to this network. Examples are MYNET and IBM Global Network.								
NETNME	The network, such as, IBM Global Network or My EDI Network.								
COMROT	The communication routine or the name of the main program that builds network commands and calls the network program to process the commands. For IBM networks, the program supplied by DataInterchange is VANIINB1. For point-to-point communication, DataInterchange provides the program named PTTOPT. Any programs you write must be defined in a member of the ADAMCTL profile.								
NETPGM	The name of the network program that sends and receives the transactions, messages, and files. For IBM networks, the program is IEBASE.								
PGMPARM	The network program parameters. The parameters that are passed to the network program. For IBM networks, the parameters are: NOSPIE,NOSTAE/,IBMØDIMR,,,,,,Y								
CMDIN	The network command input file. The file that contains the commands the network program processes (IN reference: INMSG).								
CMDLRECL	The network command record length or the length of records in the network command input file. For IBM networks, the length is 80.								
QDATA	<p>The file that contains standard transactions that are waiting to be sent to trading partners. If you leave the field blank, the file has one of the following names by default:</p> <ul style="list-style-type: none">• QDATA, for transactions enclosed in ISA/IEA interchange envelopes. The transaction data is in X12 syntax, which includes ocean, rail, TDCC, UCS, and WINS.• QDATAE, for transactions enclosed in UNB/UNZ or STX/END interchange envelopes. The transaction data is in EDIFACT syntax, which includes ODETTE, or in UNTDI syntax.• QDATAU, for transactions enclosed in BG/EG interchange envelopes. The transaction data is in X12 syntax, which includes ocean, rail, TDCC, UCS, and WINS. <p>If you enter a name, an E is appended to the <i>ddname</i> for sending EDIFACT or UNTDI requests, or a U is appended to the <i>ddname</i> for sending UCS requests. X12 requests use the <i>ddname</i> as supplied.</p> <p>The type of send command (SENDX12, for example) determines which file is used. For example, if you enter the name SENDPO and use the file to send EDIFACT transactions, DataInterchange expects to find an allocation for the <i>ddname</i> SENDPOE.</p>								
DATLRECL	<p>The length of records in the transaction data queue. For MVS, the logical record length that you allocate for the file.</p> <p>Expedite parameters for network profiles are:</p> <table><thead><tr><th>Network Profile ID</th><th>Expedite Parameter</th></tr></thead><tbody><tr><td>IINR3</td><td>INMSGHL</td></tr><tr><td>IINB1</td><td>VANIINB1</td></tr><tr><td>IINB41</td><td>VANIINB1</td></tr></tbody></table>	Network Profile ID	Expedite Parameter	IINR3	INMSGHL	IINB1	VANIINB1	IINB41	VANIINB1
Network Profile ID	Expedite Parameter								
IINR3	INMSGHL								
IINB1	VANIINB1								
IINB41	VANIINB1								

TMZONE	The time zone for your location. The network specifies the allowable codes. The Expedite reference is TIMEZONE .								
SYSTYP	The type of system you have, such as 4381. Expedite parameters for network profiles are: <table> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>SYSTYPE</td></tr> <tr> <td>IINB1</td><td>None</td></tr> <tr> <td>IINB41</td><td>None</td></tr> </table>	Network Profile ID	Expedite Parameter	IINR3	SYSTYPE	IINB1	None	IINB41	None
Network Profile ID	Expedite Parameter								
IINR3	SYSTYPE								
IINB1	None								
IINB41	None								
SYSLVL	The level of your system, such as R14. Expedite parameters for network profiles are: <table> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>SYSLEVEL</td></tr> <tr> <td>IINB1</td><td>None</td></tr> <tr> <td>IINB41</td><td>None</td></tr> </table>	Network Profile ID	Expedite Parameter	IINR3	SYSLEVEL	IINB1	None	IINB41	None
Network Profile ID	Expedite Parameter								
IINR3	SYSLEVEL								
IINB1	None								
IINB41	None								
TXTHDR	Message text header. The character that indicates the start of text for a message.								
CMDOUT	Network command output file. The name of the file containing the network's responses to the command input file. Expedite parameters for network profiles are: <table> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>OUTMSG</td></tr> <tr> <td>IINB1</td><td>OUTFILE</td></tr> <tr> <td>IINB41</td><td>OUTFILE</td></tr> </table>	Network Profile ID	Expedite Parameter	IINR3	OUTMSG	IINB1	OUTFILE	IINB41	OUTFILE
Network Profile ID	Expedite Parameter								
IINR3	OUTMSG								
IINB1	OUTFILE								
IINB41	OUTFILE								
MSGROUT	The name of the program that processes messages from the network. Expedite parameters for network profiles are: <table> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>INMOGHL</td></tr> <tr> <td>IINB1</td><td>INBIMSG</td></tr> <tr> <td>IINB41</td><td>INBIMSG</td></tr> </table>	Network Profile ID	Expedite Parameter	IINR3	INMOGHL	IINB1	INBIMSG	IINB41	INBIMSG
Network Profile ID	Expedite Parameter								
IINR3	INMOGHL								
IINB1	INBIMSG								
IINB41	INBIMSG								
SEQNUM	A sequential number assigned to all outbound documents.								
NETACKFILE	The name of a file (MVS <i>ddname</i>) where you would like the network to write network acknowledgments when you request a status update. The network acknowledgments are read and evaluated by the message handler program.								
NETPHONE	The phone number to dial to connect to your network.								
SCRIPT	The script name. This field can be used by communication software to identify a set of instructions to follow when processing requests for service. The set of instructions would be part of the communication software package and not part of DataInterchange.								
FILLER	Reserved.								
DESCRIPT	The description of this profile member.								
LOGLOCK	The profile member logical lock flag (for DataInterchange use only).								
LASTUID	The user ID of the last person to update this profile member.								
LASTUDT	The date and time that this profile member was last updated.								

Requestor Profile Block (REQDB)

Table A-17 describes the requestor profile block (REQDB). The offset values in this table are relative to 0. If this control block is used in network operations (NETOP profile), 1 should be added to the offset value because the offsets used for NETOPs are relative to 1.

Table A-17. Definition of the Requestor Profile Block (REQDB)

Field	Type	Offset	Length	Description
BLKLEN	Binary	0	2	Length of REQDB
RESERV1	Binary	2	2	Reserved
BLKNME	Character	4	8	Identifies the REQDB
REQID	Character	12	16	Requestor ID
NETID	Character	28	8	Network ID
ACCTNO	Character	36	32	Network account number
USERID	Character	68	32	Network user ID
PASSWD	Character	100	16	Network password
MSGUCL	Character	116	8	Network message user class
INDDNAME	Character	124	8	Receive file name
NETCLS	Character	132	1	Network message class
NETCHG	Character	133	1	Network charge code
NETACK	Character	134	1	Network acknowledgment
NETVCHK	Character	135	1	Destination verification
NETRETN	Character	136	3	Mailbox retention period
NETEDIO	Character	139	1	EDI receive option
NETEDIP	Character	140	1	EDI processing override
STGFRMTO	Character	141	1	Storage format override
STGFRMT	Character	142	1	Storage format
NETCMDMBR	Character	143	8	Net commands PDS member
TIMEOUT	Character	151	4	Communication line timeout value
NTACKPGM	Character	155	8	Remote Network Acknowledgments processing program
ALTNETPHONE	Character	163	32	Alternate data line phone
COMPRESS	Character	195	1	Compression (N,Y, or T)
PRIORITY	Character	196	1	Delivery priority
FILLER	Character	197	15	Reserved for DataInterchange
DESCRIPT	Character	213	30	Member description
LOGLOCK	Character	243	1	Logical lock flag
LASTUID	Character	244	17	Last update user ID
LASTUDT	Binary	261	4	Last update date/time

REQDB Field Descriptions

The following list describes the REQDB fields.

BLKLEN	Length of REQDB. A 2-byte binary field that contains the length of the REQDB data block. A REQDB is 264 bytes.														
RESERV1	Reserved.														
BLKNME	The name of the requestor profile block, EDIREQDB.														
REQID	The name (key) used to refer to this requestor.														
NETID	The name (key) that identifies the network. NETID must match the name of a member in the network profile.														
ACCTNO	For IBM networks, the account number that the network assigns to the requestor. The entry must be left-justified. For sending and receiving standard transactions using ISA/IEA enveloping, the last position must be blank.														
USERID	For IBM networks, the user ID that the network assigns to the requestor. The entry must be left-justified.														
PASSWD	The requestor's password for using the network. For IBM networks, the first 8 bytes are the current password, and the second 8 bytes are the new password (used for changing the password).														
MSGUCL	The message user class. A user-defined code that trading partners agree to use for identifying classes of information to be sent or received. Examples of classes are DEPT01, X12, MSG, FILE, EDI, and UCS. MSGUCL can be omitted to indicate that all information for the mailbox is to be sent or received.														
INDDNAME	The name of the file into which information is received from the network. The translator processes standard transactions from this file.														
NETCLS	Indicates any special status of the data being sent. NETCLS does not apply to receiving. Valid values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>blank</td><td>Normal status</td></tr> <tr> <td>T</td><td>Test status</td></tr> </table> <p>If your request specifies a trading partner, the code is taken from the trading partner profile and this field is not used.</p> <p>Expedite parameters for network profiles are:</p> <table> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>MSGNCLS</td></tr> <tr> <td>IINB1</td><td>MODE</td></tr> <tr> <td>IINB41</td><td>MODE</td></tr> </table>	Value	Description	blank	Normal status	T	Test status	Network Profile ID	Expedite Parameter	IINR3	MSGNCLS	IINB1	MODE	IINB41	MODE
Value	Description														
blank	Normal status														
T	Test status														
Network Profile ID	Expedite Parameter														
IINR3	MSGNCLS														
IINB1	MODE														
IINB41	MODE														
NETCHG	Indicates how charges are shared between sender and receiver. Valid values are: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>1</td><td>Specifies receiver pays all charges.</td></tr> <tr> <td>2</td><td>Specifies receiver pays all charges if agreed to; or, charges are split between sender's and receiver's domains.</td></tr> <tr> <td>3</td><td>Specifies receiver pays all charges if agreed to; or, charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges. This is the default.</td></tr> </table>	Value	Description	1	Specifies receiver pays all charges.	2	Specifies receiver pays all charges if agreed to; or, charges are split between sender's and receiver's domains.	3	Specifies receiver pays all charges if agreed to; or, charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges. This is the default.						
Value	Description														
1	Specifies receiver pays all charges.														
2	Specifies receiver pays all charges if agreed to; or, charges are split between sender's and receiver's domains.														
3	Specifies receiver pays all charges if agreed to; or, charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges. This is the default.														

- 4 Specifies charges are split between sender's and receiver's domains if agreed to; otherwise, the sender pays all charges.
- 5 Specifies charges are split between sender's and receiver's domains.
- 6 Sender pays all charges.

If your request specifies a trading partner, the value is taken from the trading partner profile and this field is not used.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	MSGCHRG
IINB1	CHARGE
IINB41	CHARGE

NETACK Network acknowledgment. Indicates which network acknowledgments (receipt, delivery, purge) you want to receive when sending to this trading partner. Valid values are:

Value	Description
(blank)	Requests no acknowledgments
R	Requests receipt acknowledgments only
D	Requests delivery acknowledgments only
B	Requests both receipt and delivery acknowledgments
A	Requests purge acknowledgments only
C	Requests both receipt and purge acknowledgments
E	Requests either purge or delivery acknowledgments
F	Requests receipt acknowledgments and either delivery or purge acknowledgments

If your request specifies a trading partner, the value is taken from the trading partner profile and this field is not used.

Expedite parameters for network profiles are:

Network Profile ID	Expedite Parameter
IINR3	MSGRCPT
IINB1	ACK
IINB41	ACK

NETVCHK Indicates whether the destination is verified before sending occurs. Valid values are:

Value	Description
N	Requests no verification (default)
Y	Requires verification
F	Requests verification and sending even if the destination is not verified (useful for intersystem addressing)

If your request specifies a trading partner, the value is taken from the trading partner profile and this field is not used.

Expedite parameters for network profiles are:

	<table> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>MSGVCHK</td></tr> <tr> <td>IINB1</td><td>VERIFY</td></tr> <tr> <td>IINB41</td><td>VERIFY</td></tr> </table>	Network Profile ID	Expedite Parameter	IINR3	MSGVCHK	IINB1	VERIFY	IINB41	VERIFY						
Network Profile ID	Expedite Parameter														
IINR3	MSGVCHK														
IINB1	VERIFY														
IINB41	VERIFY														
NETRETN	The number of days that data is to be kept in the network mailbox before it is purged, if it is not received. Enter blanks or zeroes to use the default number. For Expedite Base/MVS, the valid range is 001-180. For Expedite/CICS, the valid range is 01-99, left-justified. If your request specifies a trading partner, the value is taken from the trading partner profile, and this field is not used.														
NETEDIO	<p>A value that indicates whether you want to store EDI segments in the receiving file as separate records. Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Ends records at the segment delimiter (this is the default)</td></tr> <tr> <td>N</td><td>Does not end records at the segment delimiter</td></tr> </table> <p>If your request specifies a trading partner, the value is taken from the trading partner profile and this field is not used.</p> <p>The Expedite reference is EDIOPT.</p>	Value	Description	Y	Ends records at the segment delimiter (this is the default)	N	Does not end records at the segment delimiter								
Value	Description														
Y	Ends records at the segment delimiter (this is the default)														
N	Does not end records at the segment delimiter														
NETEDIP	<p>Indicates whether EDI data you receive has special EDI processing (breaking records by the segment delimiter). Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Performs EDI processing if the common data header indicates that the data is in standard format. This is the default.</td></tr> <tr> <td>N</td><td>Omits EDI processing, regardless of the common data header.</td></tr> </table> <p>If your request specifies a trading partner, the value is taken from the trading partner profile and this field is not used.</p> <p>Expedite parameters for network profiles are:</p> <table> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>EDIPROC</td></tr> <tr> <td>IINB1</td><td>AUTOEDI</td></tr> <tr> <td>IINB41</td><td>AUTOEDI</td></tr> </table>	Value	Description	Y	Performs EDI processing if the common data header indicates that the data is in standard format. This is the default.	N	Omits EDI processing, regardless of the common data header.	Network Profile ID	Expedite Parameter	IINR3	EDIPROC	IINB1	AUTOEDI	IINB41	AUTOEDI
Value	Description														
Y	Performs EDI processing if the common data header indicates that the data is in standard format. This is the default.														
N	Omits EDI processing, regardless of the common data header.														
Network Profile ID	Expedite Parameter														
IINR3	EDIPROC														
IINB1	AUTOEDI														
IINB41	AUTOEDI														
STGFRMTO	<p>Indicates whether you want to use the storage format defined in the common data header. Valid values are:</p> <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>Y</td><td>Uses the storage format as defined in the common data header (this is the default)</td></tr> <tr> <td>N</td><td>Ignores the storage format defined in the common data header</td></tr> </table> <p>If there is no common data header, the format indicated in the storage format field is used.</p> <p>If your request specifies a trading partner, the value is taken from the trading partner profile and this field is not used.</p> <p>Expedite parameters for network profiles are:</p> <table> <tr> <th>Network Profile ID</th><th>Expedite Parameter</th></tr> <tr> <td>IINR3</td><td>STGFORMO</td></tr> </table>	Value	Description	Y	Uses the storage format as defined in the common data header (this is the default)	N	Ignores the storage format defined in the common data header	Network Profile ID	Expedite Parameter	IINR3	STGFORMO				
Value	Description														
Y	Uses the storage format as defined in the common data header (this is the default)														
N	Ignores the storage format defined in the common data header														
Network Profile ID	Expedite Parameter														
IINR3	STGFORMO														

IINB1	DLMOVERRIDE
IINB41	DLMOVERRIDE

STGFRMT

Indicates to the network how data is stored for free-format messages and files.

Consider the type of data you want to send and how the file is received when determining what values to select. Contact a representative from each network you are using for all available values.

For example, if you are using:

- The IBM Expedite/MVS Base Version 1.1 (IEBASE), valid values are:

Value	Description
-------	-------------

C	Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Program source code defined with variable length records is the type of file generally sent with this option. Output records do not include the carriage return and line-feed characters.
---	---

L	Indicates that each record is preceded by a 2-byte hexadecimal record length. Select this option when sending a data set defined in a fixed format, or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.
---	--

N	Stores data as it is received. Output records are built based on the record length of the data set allocated to receive the data.
---	---

- IBM Expedite/CICS, valid values are:

Value	Description
-------	-------------

A	Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Program source code defined with variable length records is the type of file generally sent with this option. Output records do not include the carriage return and line-feed characters.
---	---

L	Indicates that each record is preceded by a 2-byte hexadecimal record length. Select this option when sending a data set defined in a fixed format, or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.
---	--

O	Other. Free format.
---	---------------------

- The IBM Expedite/MVS Host Version 1.3 (IEBASE), valid values are:

Value	Description
-------	-------------

A	Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Program source code defined with variable length records is the type of file generally sent with this option. Output records do not include the carriage return and line-feed characters.
---	---

B	Specifies each record is preceded by a 2-byte hexadecimal record length. Select this option when sending a data set defined in a fixed format, or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.
---	---

	C	Stores data as it is received. Output records are built based on the record length of the data set allocated to receive the data.
	Expedite parameters for network profiles are:	
	Network Profile ID	Expedite Parameter
	IINR3	STGFORM
	IINB1	DELIMITED
	IINB41	DELIMITED
NETCMDMBR	The name of a member of a PDS that will be allocated to the ddname of EDINTCMD. This member will contain the commands that you want to pass to a network. DataInterchange will read the commands from the PDS member and write the commands to the <i>Network input file</i> specified in the network profile member after all substitutable variable tags have been resolved by DataInterchange	
TIMEOUT	A value which will be used as a maximum allowable time that the data line for communications can be idle without being dropped. If you specify a trading partner when requesting network activity (send or receive), the value for this field is taken from the TP profile. Otherwise, the value for this field is taken from the requestor profile.	
NTACKPGM	The name of a program (chosen from the list below) that will be used to process network acknowledgments from a secondary network.	
	Note: When a gateway is used to connect to another VAN (such as IBM Global Network), the gateway is referred to as the primary network because it is the network with which DataInterchange interfaces. The other VAN, such as IBM Global Network, is referred to as the secondary or remote network because DataInterchange goes through the gateway network to reach the other network. When the gateway is used to connect directly to a trading partner's site or when the gateway is used as the only network, there is no secondary network.	
	This program will be used only if you are using a gateway to connect to a secondary network (assuming we call the gateway your primary network) and have requested and received network acknowledgments (into <i>Net acks file</i>) from the secondary network.	
	INB1MSG -- IN	
ALTNETPHONE	The alternate phone number to dial to connect to your network.	
COMPRESS	The Expedite Base/MVS compression code. A Y specifies that compression is wanted, an N specifies that compression is not wanted, a T specifies that Expedite Base/MVS is to use its own table to decide whether or not to compress. This field is only valid during send.	
PRIORITY	Blank (for normal priority) or P (for high priority). These codes are used by Expedite Base/MVS and Expedite/CICS to prioritize delivery of messages. High priority messages will be delivered to your trading partners before normal priority messages.	
FILLER	Reserved for DataInterchange.	
DESCRIPT	The description of this profile member.	
LOGLOCK	The profile member logical lock flag (for DataInterchange use only).	
LASTUID	The user ID of the last person to update this profile member.	
LASTUDT	The date and time that this profile member was last updated.	

Appendix B. DataInterchange Utility Condition Codes and API Return Codes

The DataInterchange Utility references the Utility Control Information Block (FFUS). See “Format of DataInterchange Utility Control Information” on page 5-18 for more information. As such, Utility return codes are placed within this block. In MVS, knowledge of this block is not necessary, except to know that the condition code returned from the Utility equates to the CCBERC field in the FFUS block. In CICS, the codes returned from the Utility are returned in the CCBRC and CCBERC fields in the FFUS block. Please note that CCBRC and CCBERC refer to fields within the FFUS block and should not be confused with the ZCCBRC and ZCCBERC fields in the Common Control Block (CCB). CCBRC is the Utility severity code and, therefore, is also referred to as UTILSEV. CCBERC is the Utility condition code and, therefore, is also referred to as UTILCCODE. It is the names UTILSEV and UTILCCODE that appear in the tables that follow.

This Appendix contains the condition codes, return codes, and extended return codes that DataInterchange can pass to an application. The codes are arranged in the following groups:

DataInterchange Utility severity codes CCBRC (or UTILSEV) and DataInterchange Utility condition codes CCBERC (or UTILCCODE).

- General functions of the DataInterchange Utility
- Translation
- Communications
- Combination Commands

Application Programming Interface return codes.

- Initialization return codes
- Translation return codes
- Communications return codes
- Ending DataInterchange return codes
- Negative return codes

For more information and recommended actions, see *DataInterchange Messages and Codes*.

Return Codes

The tables in this chapter refer to the DataInterchange Utility Severity (UTILSEV) and Condition (UTILCCODE) codes. For MVS batch users, the UTILCCODE code is returned as the Job Step Condition Code. For CICS users, these codes are returned in the Utility Control Block. See “Format of DataInterchange Utility Control Information” on page 5-18. UTILSEV refers to the Utility Control Block field, CCBRC. And UTILCCODE refers to the Utility Control Block field, CCBERC. The Utility Control Block fields CCBRC and CCBERC should not be confused with the Common Control Block (CCB) fields ZCCBRC and ZCCBERC.

DataInterchange Utility Condition Codes

When the DataInterchange Utility completes its processing, a job step condition code is returned in MVS. Possible values are provided in the tables below under column "Condition code - UTILCCODE." For CICS users, this value is set in the CCBERC field in the Utility Control Information Block along with an associated severity code, CCBRC. For more information about the format of DataInterchange Utility control information, see "Format of DataInterchange Utility Control Information" on page 5-18. The DataInterchange Utility returns the most serious return code encountered during a job step. In MVS, your JCL can include logic that tests the return code and, if appropriate, ends the job. If a nonzero job step condition code is returned, examine the audit trail report (ddname PRTFILE) for details.

In those cases where you want to ignore certain condition codes, the keywords IFCC and SETCC can be used on any PERFORM statement to override up to 10 different possible condition codes. For more on overriding the condition codes, see "General DataInterchange Utility Condition Codes."

Note: The IBM Global Network recommends caution should be exercised to prevent overriding a meaningful error and, thus, causing unpredictable results.

General DataInterchange Utility Condition Codes

Table B-1 describes condition codes for general DataInterchange Utility functions — noncommunications and nontranslation.

Table B-1 (Page 1 of 3). DataInterchange Utility Condition Codes for Noncommunication and Nontranslation Functions

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Description
0000	0000	N/A	Processing completed successfully for all PERFORM commands.
0008	0003	N/A	See "Translation Condition Codes" on page B-4. Or this error may indicate that records were written to the exception file. (See CCEXCEPTION value 'Y' message ID FF0194.)
0008	0016	N/A	Edit services could not successfully initialize.
0008	0020	FF0486	Command only supported in CICS.
0008	0032	N/A	Cannot open the print file (ddname PRTFILE in MVS).
0008	0036	FF0550	Report continuous receive status error.
0008	0048	Multiple	An error occurred while parsing the command language input. (Msgs: TS0100 - TS0230.)
0008	0064	FF0410	A command file name was not passed into the DataInterchange Utility (CICS only).
0008	0080	FF0411	Cannot open the command file (ddname SYSIN in MVS).
0008	0088	Multiple	An error has occurred attempting to invoke a response application. (Msgs: FF0477 - FF0478.)
0008	0096	FF0200	The DataInterchange Utility attempted to obtain storage, but could not.
0008	0112	FF0412	An error occurred while attempting to read from the command file.
0008	0120	N/A	Service director could not successfully initialize.
0008	0128	FF0413	Too many commands were contained in the command file.
0008	0144	FF0414	An error occurred while attempting to close the command file.

Table B-1 (Page 2 of 3). DataInterchange Utility Condition Codes for Noncommunication and Nontranslation Functions

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Description
0008	0160	FF0404	An error occurred while attempting to free a storage location.
0008	0176	Multiple	An error occurred in the Transaction Store while selecting transactions. (Check for Repository Msgs.)
0008	0184	FF0415	There are no transactions in the Transaction Store to process.
0008	0192	FF0416	A selection criteria was specified, but no transactions in the store meet the criteria.
0008	0208	FF0417	A query file name was not passed into the DataInterchange Utility (CICS only).
0008	0224	FF0418	Cannot open the query file (ddname EDIQUERY in MVS).
0008	0240	FF0419	An error occurred while attempting to write to the query file.
0008	0256	FF0420	An error occurred while attempting to close the query file.
0008	0272	Multiple	An error occurred during the processing of a PURGE command. (Msgs: FF0422 - FF0425.)
0008	0288	Multiple	An error occurred during the processing of an UNPURGE command. (Msgs: FF0426 - FF0429.)
0008	0300	Multiple	An error occurred in Expedite/CICS during start and stop of continuous receive. (Msgs: CR0040 and VM1017 or VN1019.)
0008	0303	Multiple	An error occurred while attempting to retrieve a continuous receive profile during a start or stop of continuous receive. (Msgs: CR0010 and PS0301.)
0008	0304	Multiple	An error occurred during the processing of a HOLD command. (Msgs: FF0430 - FF0433.)
0008	0320	Multiple	An error occurred during the processing of a RELEASE command. (Msgs: FF0434 - FF0437.)
0008	0336	FF0510	An error occurred during the processing of a PRINT command.
0008	0464	Multiple	An error occurred during the processing of an ENVELOPE command. (Msgs: FF0438 - FF0443 - FF0571.)
0008	0480	Multiple	An error occurred during the processing of a REENVELOPE command. (Msgs: FF0444 - FF0449 - FF0571.)
0008	0496	FF0421	A request to send or receive data was issued, but a requestor ID was not supplied.
0008	0512	FF0453	An error occurred while attempting to retrieve a requestor ID.
0008	0544	FF0450	A request to send or receive data was issued, but a requestor ID was not supplied that matches a network for which data was enveloped.
0008	0560	Multiple	An error occurred while attempting to retrieve a message from the message file. (Msgs: MS0010 - MS0020.)
0008	0576	FF0401	An error occurred while attempting to write to the print file.
0008	0592	FF0451	A command requiring a selection criteria was issued, but no selection criteria were specified.
0008	0608	FF0452	A requestor ID was specified, but it could not be found.

Table B-1 (Page 3 of 3). DataInterchange Utility Condition Codes for Noncommunication and Nontranslation Functions

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Description
0008	0624	Multiple	An error occurred during the processing of a REMOVE TRANSACTIONS command. (Msgs: FF0454 - FF0462.)
0008	0632	FF0573	Remove log error.
0008	0634	FF0575	Load log error.
0008	0636	FF0577	Unload log error.
0008	0640	FF0300	Cannot open the tracking file (ddname FFSTRAK in MVS).
0008	0656	FF0302	An error occurred while attempting to write to the tracking file.
0008	0672	FF0300	Cannot open the exception file (ddname FFSEXCP in MVS).
0008	0688	FF0302	An error occurred while attempting to write to the exception file.
0008	0704	Multiple	An error occurred during the processing of an EXPORT command. (Msgs: EI0002 - EI0063.)
0008	0720	Multiple	An error occurred during the processing of an IMPORT command. (Msgs: EI0002 - EI0063.)
0008	0736	FF0467	The command file does not contain any commands to process.
0008	0752	Multiple	An error occurred during the processing of a CLOSE MAILBOX command. (Check for communication errors.)
0008	0768	FF0469	Elapsed time exceeded for REMOVE TRANSACTIONS.
0008	0784	FF0471	No records match selection criteria for DATA EXTRACT command.
0008	0800	FF0472	Error during MAPPING MIGRATION command.
0008	0816	FF0421	Mandatory REQID keyword not provided.
0008	0832	FF0474	Error reading NETPROF profile member.
0008	0848	FF0475	Error encountered by message handler program.
0008	0864	FF0525	Error during UPDATE STATUS command.
0008	0880	Multiple	Error during management reporting command. (Msgs: FF0527, FF0529, FF0531, or FF0534.)
0008	0896	FF0546	The delete profile request failed.
0008	0903	FF0194	Records written to the exception file. (See CCEXCEPTION value 'X'.)
0008	0912	FF0480	No message handler specified.
0008	0928	FF0481	No network output file specified.
0008	0976	FF0542	Reconstruct command failed.
0008	0992	FF0548	The query profile request failed.

Translation Condition Codes

This section categorizes all translation errors, provides outbound and inbound translation considerations, and tabulates all translator errors with their corresponding DataInterchange Utility condition codes. The applicable DataInterchange Utility commands are:

Outbound Translation:

- PERFORM TRANSLATE TO STANDARD
- PERFORM TRANSLATE AND ENVELOPE
- PERFORM TRANSLATE AND SEND

Inbound Translation:

- PERFORM DEENVELOPE
- PERFORM TRANSLATE TO APPLICATION
- PERFORM RETRANSLATE TO APPLICATION
- PERFORM DEENVELOPE AND TRANSLATE
- PERFORM RECEIVE AND TRANSLATE

Translation errors are categorized by level. The highest error encountered is reflected in the DataInterchange Utility Condition Code.

Code	Description
0	Normal or warning conditions
1	Data element errors
2	Segment errors
3	Transaction errors
4	Group envelope errors
5	Interchange envelope errors
6	Invalid data errors
> 120	Environmental and Program errors

The translator continues processing in all cases except severe errors (Environmental and Program errors). For example, if the translator encounters an interchange level error, code of 5, it continues by processing the next interchange. Similarly, if an "unacceptable" translation error occurs, the translator skips the current transaction and goes on to the next transaction. The "acceptable error level" is defined in the trading partner transaction usage. You indicate a level of 0, 1, or 2 as being acceptable. When a program error occurs, the translator aborts immediately. The DataInterchange Utility records the errors in the audit trail file (print file) and event log.

Note: The DataInterchange Utility does not return the information records you requested for translation errors with a level of 3 or greater.

Outbound Translation Considerations

Errors that exceed the acceptable error level you chose when defining the transaction stop the translation of the current transaction. A transaction that is not successfully translated is written to the exception file. If the DataInterchange Utility encounters one of the errors described in this section and it is unacceptable, it writes the entire transaction to the exception file. Also, if C and D records are being used, the return codes in the control record are updated accordingly. Data records in the exception file are copies of your input records.

Notes:

1. Depending on the error, the exception file may not contain all the untranslated records. You may have to use your original application files to correct and reprocess the untranslated transactions. The audit trail report contains information that can help you recover the untranslated transactions. The event log is also an important source of diagnostic information.
2. Initialization errors, such as failure to open a file, result in SYSPRINT error messages.

Additionally, the DataInterchange Utility can generate errors prior to translator invocation. Here are the condition codes which may be caused by problems with an application file (utility keyword APPFILE):

Table B-2. DataInterchange Utility Condition Codes from Invalid Application Input

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Description
0008	0006	FF0131	Input APPFILE is empty
0008	0009	FF0152	A transaction in APPFILE does not contain a valid record code
0008	0011	FF0151	A transaction in APPFILE does not contain a C record
0008	0017	FF0150	A transaction in APPFILE does not contain a D record

Inbound Translation Considerations

Errors that exceed the acceptable error level you chose when defining the transaction stop the translation of the current transaction. A transaction that is not successfully translated is NOT written to the exception file. If the DataInterchange Utility encounters one of the errors described in this section and it is unacceptable, it writes the transaction to the Transaction Store, but produces no application output data.

Notes:

1. Application output will only be written to the exception file in the event that the application file(s) cannot be open.
2. You can use the RETRANSLATE TO APPLICATION function to reprocess the untranslated transactions. The audit trail report contains information that can help you recover the untranslated transactions. The event log is also an important source of diagnostic information.

Translation Condition Code Tables

The next nine tables list all possible translator errors. Each table represents a category as follows:

Error	Example
Warning conditions	Structure not used, end of file
Data element errors	Data element too short or too long
Segment errors	Mandatory segment missing, unrecognized segment
Transaction syntax errors	Control number mismatch, segment count incorrect
Transaction environmental errors	Mapping not found
Group errors	Control number mismatch, password invalid
Interchange errors	Control number mismatch, trading partner not known
Invalid data errors	Not an interchange
Environmental or program errors	Database error, insufficient virtual storage
<p>Note: Each translation error level can have many causes. Each cause has a unique <i>Msg ID</i> and a <i>Unique code</i> assigned. For more information regarding a specific error message, see <i>DataInterchange Messages and Codes</i>. For more information regarding how <i>Unique code</i> can be used, see Table A-5 on page A-28.</p>	

Normal or Warning Condition Codes

Table B-3 on page B-7 shows warnings or normal conditions that do not set a condition code.

Table B-3. DataInterchange Utility Condition Codes for Translation with Normal or Warning Conditions

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0000	0000	None	0	Processing of requested function completed normally.
0000	0000	TR0841	2	Structure not defined in the application data format.
0000	0000	TR0403	3	Structure not used in the mapping.
0000	0000	TR0404	4	Structure defined as part of parent (not passed separately).
0000	0000	TR0405	5	Raw data structure could not be identified.
0000	0000	TR0406	6	Raw data structure received but structure defined to start the translation not yet received.
0000	0000	TR0407	7	Mismatching loop ID values in LS and LE segments.
0000	0000	TR0408	8	LS segment does not have a corresponding LE segment.
0000	0000	TR0409	9	No mapping provided for hierarchical code and parent hierarchical code combination.
0000	0000	TR0824	505	Transmission of envelope to communications failed.
0000	0000	TR0401	1	End of file, no more envelopes in the envelope queue.

Data Element Condition Codes

Table B-4 shows the errors that occur at the data element level. The translation might still be acceptable based on the acceptable error level established in the send or receive usage record.

Table B-4 (Page 1 of 2). DataInterchange Utility Condition Codes for Data Element Translation Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0008	0001	None	0	The highest severity of error detected during translation of this transaction is at the data element level.
0008	0001	TR0001	101	A mapped mandatory data element is blank for a segment containing data in other data elements.
0008	0001	TR0002	102	Data element is too long.
0008	0001	TR0003	103	Data element is too short.
0008	0001	TR0004	104	Code in ID type field not found in validation table.
0008	0001	TR0005	105	Code in ID type field not found in translation table.
0008	0001	TR0006	106	User exit for data element failed. Exit routine returned an error.
0008	0001	TR0007	107	Invalid date format. Unable to reformat date according to customized date edit number.
0008	0001	TR0008	108	Data element conversion failed. Format of data in input buffer conflicts with mapping.
0008	0001	TR0009	109	Standard length exceeds application length.

Table B-4 (Page 2 of 2). DataInterchange Utility Condition Codes for Data Element Translation Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0008	0001	TR0010	111	Paired conditionality in segment not satisfied.
0008	0001	TR0011	112	Required conditionality in segment not satisfied.
0008	0001	TR0012	113	Mutually exclusive conditionality in segment not satisfied.
0008	0001	TR0013	114	Conditional conditionality in segment not satisfied.
0008	0001	TR0014	110	Conditional-paired conditionality in segment not satisfied.
0008	0001	TR0015	115	Mandatory composite field missing.
0008	0001	TR0016	116	Data element validation failed.
0008	0001	TR0017	117	Attempt to increment accumulator will exceed maximum size.
0008	0001	TR0018	118	Attempt to add a value to an accumulator will exceed maximum size.
0008	0001	TR0023	119	Data in application data format has been overlaid.
0008	0001	TR0024	120	Field data that was not mapped has been received.

Segment Condition Codes

Table B-5 shows the errors that occur on the segment level. The translation might still be acceptable based on the acceptable error level established in the send or receive usage record.

Table B-5 (Page 1 of 2). DataInterchange Utility Condition Codes for Segment Translation Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0008	0002	None	0	The highest severity of error detected during translation of this transaction is at the segment level.
0008	0002	TR0019	207	Error getting storage while processing binary segment.
0008	0002	TR0020	208	Error opening a file while processing binary segment.
0008	0002	TR0021	209	Error reading a file while processing binary segment.
0008	0002	TR0022	210	Error writing a file while processing binary segment.
0008	0002	TR0050	201	A segment contains more elements than the standard allows.
0008	0002	TR0051	202	Unrecognized segment ID. The segment is not defined for the transaction.
0008	0002	TR0052	203	A mandatory, mapped segment for this trading partner is missing.
0008	0002	TR0053	204	Occurrences of a repeating segment in the input data exceed the maximum use count for the structure in the data format.
0008	0002	TR0054	205	Loop repetition count exceeded. Loop repeats more times than definition specifies.
0008	0002	TR0055	206	Segment repetition count exceeded. Segment repeats more times than definition specifies.

Table B-5 (Page 2 of 2). DataInterchange Utility Condition Codes for Segment Translation Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0008	0002	TR0056	211	Unexpected segment data received.
0008	0002	TR0057	215	Application data received out of sequence.
0008	0002	TR0058	212	Creation of standard loop has been aborted.
0008	0002	TR0059	213	Creation of a repeating segment has been aborted.
0008	0002	TR0060	214	Creation of a segment has been aborted.

Transaction Condition Codes

This section describes the transaction errors. Table B-6 shows errors that indicate a serious problem with the syntax of a transaction. Table B-21 on page B-22 shows the environment-related errors. Such errors include a situation in which DataInterchange is unable to locate a map or control string, which is necessary to process the transaction.

Table B-6. DataInterchange Utility Condition Codes for Transaction Translation - EDI Syntax Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0008	0003	None	0	The highest severity of error detected during translation of this transaction is at the transaction set level.
0008	0003	TR0025	326	Duplicate transaction within group or interchange.
0008	0003	TR0101	302	Transaction set control numbers do not match in header and trailer.
0008	0003	TR0103	304	Transaction set trailer contains invalid segment count.
0008	0003	TR0105	315	Security profile member could not be found for encrypted transaction.
0008	0003	TR0106	316	Authentication failed.
0008	0003	TR0107	317	S2S segment without S2E. Incomplete security segments for received transaction.
0008	0003	TR0108	318	S2E segment without S2S. Incomplete security segments for received transaction.
0008	0003	TR0207	506	Envelope is not defined correctly. Envelope definition damaged.
0008	0003	TR0208	507	Envelope is not defined correctly. Envelope definition damaged.
0008	0003	TR1257	335	Mandatory composite missing in service segment.
0008	0003	TR1258	336	Service segment data element too long.
0008	0003	TR1259	337	Service segment data element value not defined in validation table.
0008	0003	TR1260	338	Service segment data element value not consistent with data type.
0008	0003	TR1261	339	Mandatory data element missing in service segment.
0008	0003	TR1262	340	Service segment data element too small.

Transaction Environmental Errors

Table B-7 (Page 1 of 2). DataInterchange Utility Condition Codes for Transaction Translation - Environmental Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0008	0003	TR0104	305	Receive trading partner transaction not found.
0008	0003	TR0109	319	Attempt to receive translate (TRANSLATE TO APPLICATION) a transaction that was previously send translated (TRANSLATE TO STANDARD).
0008	0003	TR0110	320	Invalid transaction handle.
0008	0003	TR0111	321	Attempt to envelope a received transaction.
0008	0003	TR0112	322	Attempt to translate a transaction that does not have a group or interchange segment associated with it.
0008	0003	TR0113	324	Attempt to envelope a transaction from a bundle without enveloping the controlling transaction.
0008	0003	TR0114	325	Attempt to envelope a transaction that was not translated successfully.
0008	0003	TR0115	327	Transaction control number assigned by application not valid.
0008	0003	TR0116	328	Transaction control number assigned by application is a duplicate.
0008	0003	TR0117	330	Call to transaction services for details failed.
0008	0003	TR0118	331	Error attempting to get image from Transaction Store.
0008	0003	TR0119	332	Request to get an image that does not exist.
0008	0003	TR0120	333	Request to get an image but the base has not been established.
0008	0003	TR0121	334	Attempt to envelope a transaction with an incorrect status.
0008	0003	TR0155	406	Authentication routine required but not provided.
0008	0003	TR0156	407	Encryption routine required but not provided.
0008	0003	TR0157	408	Filtering routine required but not provided.
0008	0003	TR0818	307	The translator could not write to the event log. The log might be full.
0008	0003	TR0820	308	Send trading partner transaction not for this trading partner, application, and direction.
0008	0003	TR0821	306	Application data format not found.
0008	0003	TR0822	309	Control string not found for this transaction.
0008	0003	TR0825	310	Trading partner profile member not found.
0008	0003	TR0826	311	Sender ID in envelope profile member is blank.
0008	0003	TR0830	312	Input data block contains no data. Application data length is 0.
0008	0003	TR0834	313	Overran output buffer. User's output buffer is too small for the data.
0008	0003	TR0848	329	Failed to load a user exit routine.

Table B-7 (Page 2 of 2). DataInterchange Utility Condition Codes for Transaction Translation - Environmental Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0008	0003	TR0850	323	Standard delimiters are not unique. Update delimiters in envelope standard or trading partner profile.
0008	0003	SA0042	314	Access denied to function within resource.

Group Condition Codes

Table B-8 shows the errors that indicate a serious problem with an entire functional group within an interchange. None of the transactions within the group are processed.

Table B-8. DataInterchange Utility Condition Codes for Group Translation Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0008	0004	None	0	An error is detected in the functional group header or trailer. No transactions in the functional group are translated. Processing continues.
0008	0004	TR0100	301	Transaction set header is missing or invalid.
0008	0004	TR0102	303	Transaction set trailer is missing or invalid.
0008	0004	TR0107	409	S1S segment without S1E.
0008	0004	TR0108	410	S1E segment without S1S.
0008	0004	TR0151	402	Functional group control numbers do not match in header and trailer.
0008	0004	TR0153	404	Functional group trailer contains invalid transaction count.
0008	0004	TR0154	405	Authentication failed for group.
0008	0004	TR0155	406	Authentication routine required but not provided.
0008	0004	TR0156	407	Encryption routine required but not provided.
0008	0004	TR0157	408	Filtering routine required but not provided.
0008	0004	TR0158	411	There is a duplicate group within the interchange.
0008	0004	TR0207	506	Envelope is not defined correctly. Envelope definition damaged.
0008	0004	TR0208	507	Envelope is not defined correctly. Envelope definition damaged.
0008	0004	TR1257	412	Mandatory composite missing in service segment.
0008	0004	TR1258	413	Service segment data element too long.
0008	0004	TR1259	414	Service segment data element value not defined in validation table.
0008	0004	TR1260	415	Service segment data element value not consistent with data type.
0008	0004	TR1261	416	Mandatory data element missing in service segment.
0008	0004	TR1262	417	Service segment data element too small.

Interchange Condition Codes

Table B-9 on page B-12 shows the errors indicating the translator was able to isolate an interchange to process, but there is a serious problem with that interchange and none of the transactions are processed.

Table B-9. DataInterchange Utility Condition Codes for Interchange Translation Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0008	0005	None	0	An error is detected in the interchange header or trailer. No transactions in the interchange are translated. Processing continues.
0008	0005	TR0150	401	Functional group header is missing or invalid.
0008	0005	TR0152	403	Functional group trailer is missing or invalid.
0008	0005	TR0201	501	Interchange header contains invalid sender ID. Cannot locate trading partner profile member.
0008	0005	TR0203	502	Interchange control numbers do not match in header and trailer.
0008	0005	TR0205	503	Interchange trailer contains invalid functional group count.
0008	0005	TR0206	504	Password does not match password in trading partner profile.
0008	0005	TR0207	506	Interchange is not defined correctly or the interchange is badly damaged and cannot be parsed.
0008	0005	TR0208	507	Received interchange segment contains invalid data.
0008	0005	TR0209	508	Expected delimiter not found. Encrypted data not immediately followed by segment delimiter.
0008	0005	TR0210	509	Envelope definition not found for received interchange.
0008	0005	TR0211	510	A duplicate interchange was detected and is being skipped.
0008	0005	TR1257	511	Mandatory composite missing in service segment.
0008	0005	TR1258	512	Service segment data element too long.
0008	0005	TR1259	513	Service segment data element value not defined in validation table.
0008	0005	TR1260	514	Service segment data element value not consistent with data type.
0008	0005	TR1261	515	Mandatory data element missing in service segment.
0008	0005	TR1262	516	Service segment data element too small.

Invalid Data Condition Codes

Table B-10 on page B-13 shows the errors that indicate that the translator cannot identify the data it is reading from the file as standard data. The translator expects the ISA, UNB, SCH, ICS, BG, or GS segment to signal the start of standard data. Anything before, between, or after any of these valid interchanges is flagged as an error.

Table B-10. DataInterchange Utility Condition Codes Set by Invalid Data in the Input File

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0008	0006	None	0	Invalid data found in the input file. Data does not match the format required by the standard.
0008	0006	TR0200	601	Interchange envelope header is missing or invalid.
0008	0006	TR0204	602	Interchange envelope trailer is missing or invalid.
None	????	TR0841	???	MUW application structure ID is invalid.
0008	0006	TR0842	603	No standard data found in input file.
0008	0006	TR0202	604	Interchange header found while looking for a trailer.

Environmental and Program Error Condition Codes

Table B-11 shows the errors considered so serious that the translator logs the appropriate message for the error and ends immediately.

Table B-11 (Page 1 of 2). DataInterchange Utility Condition Codes for Environmental and Program Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0012	0121	TR1201	916	A program error occurred during anchor processing.
0012	0122	TR1202	917	Unable to free main storage.
0012	0123	TR1203	918	Unable to read repository.
0012	0125	TR1205	919	QSAM failed to read file.
0012	0126	TR1206	920	QSAM failed to close file.
0012	0127	TR1207	927	DataInterchange ended because of severe error generating functional acknowledgments.
0012	0128	TR1208	928	DataInterchange ended because of severe error updating the management reporting databases.
0012	0129	TR1209	929	Error reading envelope control string
0012	0130	TR0810	901	Profile read for update failed.
0012	0131	TR0811	902	Profile write failed.
0012	0132	TR0812	903	Parameter count is invalid.
0012	0135	TR0815	904	A request to get main storage failed.
0012	0136	TR0816	905	Invalid function code.
0012	0137	TR0817	906	The transaction processor did not end because it is not active. This error also occurs if input records are out of sequence, such as a C record followed by a blank record.
0012	0147	TR0827	908	You cannot perform this function now. Function code not valid at this time.
0012	0148	TR0828	909	Input data block size invalid. Must be at least 32 K.
0012	0149	TR0829	910	Output data block size invalid. Must be at least 32 K.
0012	0152	TR0832	911	Transaction Store call failed.
0012	0156	TR0836	912	Repository read failed.

Table B-11 (Page 2 of 2). DataInterchange Utility Condition Codes for Environmental and Program Errors

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Unique Code	Description
0012	0158	TR0838	913	QSAM failed to open file.
0012	0159	TR0839	914	Requestor profile not found.
0012	0160	TR0840	915	Standards profile member not found.
0012	0163	TR0843	907	Receive file name in requestor profile is blank.
0012	0164	SA0042	921	Access denied to function within resource.
0012	0166	TR0846	922	Raw data control string not found.
0012	0167	TR0847	923	Failed to find security profile member.
0012	0168	TR0848	924	Failed to load a user exit routine.
0012	0169	TR0849	925	Error returned by user exit routine.
0012	0171	TR0851	926	An application data format ID is required when raw data is specified.
0012	0172	TR1252	930	Error reading target ADF control string.
0012	0173	TR1253	931	No beginning/ending structure.
0012	0174	TR1254	932	No internal trading partner ID value.
0012	0175	TR1255	933	User exit (IUSEREXIT) not defined.
0012	0176	TR1256	934	IUSEREXIT returned an error.
0012	0177	TR1257	935	Mandatory composite missing in service segment.
0012	0178	TR1258	936	Service segment data element too long.
0012	0179	TR1259	937	Service segment data element value not defined in validation table.
0012	0180	TR1260	938	Service segment data element value not consistent with data type.
0012	0181	TR1261	939	Mandatory data element missing in service segment.
0012	0182	TR1262	940	Service segment data element too small.
0012	0183	TR1263	941	Database error obtaining lock.

Communications Condition Codes

DataInterchange Utility condition codes are specialized for communications functions. The condition code is not common between MVS and CICS. There is a different set of codes, depending on the environment. These codes apply to the following commands:

- PERFORM SEND
- PERFORM RECEIVE
- Combination commands including SEND or RECEIVE

Table B-12 on page B-15 describes DataInterchange Utility job step condition codes for sending and receiving in **MVS**.

Table B-12. DataInterchange Utility Condition Codes for Communications Services in MVS

Job Step Condition Code	Msg ID	Description
0000	Multiple	Successful transmission. (Msgs: FF0020 or FF0030)
0001	VN1015	Successful transmission, but an invalid parameter was passed to the network program. Examine the network command output file for warning conditions. This applies to MVS only. Note: Msg ID, VN1015, is also used under more serious circumstances (see DataInterchange Utility Condition code = 0015 below). A condition code of 0001 reflects a warning - all valid data was processed successfully. A code of 0015 indicates that one or more envelopes were not processed — processing ended prematurely.
0002	VN1040	No data received on a receive request. (Msgs: FF0142 and VN1040.)
0002	FF0140	QSAM error prevented completion of request to clear the file (send requests).
0003	VN1041	Not able to process the network command output file.
0003	FF0111	Network profile not found.
0004	Multiple	Error in profile services. (Msgs: VN1004 - VN1005 and PS0010 - PS0310.)
0005	FF0140	Network profile does not provide name of send/receive program.
0005	CM0005	Cannot pass control to VANIINB1.
0006	SA0042	No authority to perform network functions.
0007	CM0006	Network ID is invalid for the MVS environment.
0015	VN1015	Invalid parameter passed to network program. One or more envelopes were not sent/received. Processing ended prematurely. Examine the network command output file for errors.
nnnn	Multiple	Other severity 0008 condition codes are possible, but unlikely. The code will reflect the last three characters of the communications extended return code. <i>Example:</i> Communications returns 8 and 1012 (for msg ID VN1012); the DataInterchange Utility returns 0012. See Table B-27 on page B-28.
nnnn	Multiple	When the communications return code is 0012, all condition codes are 120 plus the last three characters of the communications extended return code. <i>Example:</i> Communications returns 12 and 1016 (for msg ID VN1016); the DataInterchange Utility returns 0136. See Table B-28 on page B-28.

Table B-13 describes DataInterchange Utility condition codes for sending and receiving in **CICS**.

Table B-13 (Page 1 of 2). DataInterchange Utility Condition Codes for Communications Services in CICS

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Description
0000	0000	Multiple	Successful transmission. (Msgs: FF0020 or FF0030)
0004	0002	VN1040	No data received on a receive request. (Msgs: FF0142 and VN1040.)
0008	0003	FF0111	Network profile not found.
0008	0004	Multiple	Error in profile services. (Msgs: VN1004 - VN1005 and PS0010 - PS0310.)
0008	0007	CM0006	Network ID is invalid for the CICS environment.

Table B-13 (Page 2 of 2). DataInterchange Utility Condition Codes for Communications Services in CICS

Severity UTILSEV	Condition Code UTILCCODE	Msg ID	Description
0008	0017	VN1017	Error occurred during the execution of an Expedite/CICS command. Examine the print file or the event log for the Expedite/CICS error.
0008	0019	VN1019	A timeout error occurred while DataInterchange was waiting for Expedite/CICS to complete a continuous receive termination request.
0008	0020	VN1020	A timeout error occurred while DataInterchange was waiting for Expedite/CICS to complete a request to receive network acknowledgments.
0008	0021	VN1021	A timeout error occurred while DataInterchange was waiting for Expedite/CICS to complete a single receive request.
0008	nnnn	Multiple	Other severity 0008 condition codes are possible, but unlikely. The code will reflect the last three characters of the communications extended return code. <i>Example:</i> Communications returns 8 and 1012 (for msg ID VN1012); the DataInterchange Utility returns 0012. See Table B-27 on page B-28.
0012	nnnn	Multiple	When the communications return code is 0012, all condition codes are 120 plus the last three characters of the communications extended return code. <i>Example:</i> Communications returns 12 and 1016 (for msg ID VN1016); the DataInterchange Utility returns 0136. See Table B-28 on page B-28.
0008	0300	Multiple	Error occurred while attempting to retrieve a continuous receive profile during a start or stop of continuous receive. (Msgs: CR0010 and PS0301.) Examine the event log for PS0301 and take the action indicated.
0008	0303	Multiple	Error occurred in Expedite/CICS during start and stop of continuous receive. (Msgs: CR0040 and VM1017 or VN1019).

Combination Command Condition Codes

When there is a problem processing a combination command, DataInterchange supplies one condition code for your application. This condition code might not specifically identify the problem area. If the condition code is insufficient for problem identification, restart the DataInterchange Utility using separate commands. This section describes the error conditions that can occur during combination command processing.

Translate and Send Errors

When you issue a PERFORM TRANSLATE AND SEND command to DataInterchange, several error conditions can occur. If a severe error occurs during translation, DataInterchange ignores the send request, and the utility condition code reflects the translator-extended return code. If an error that is not severe occurs during translation, DataInterchange attempts to complete the send process. If the send is successful, the utility condition code reflects the translator-extended return code. If an error occurred during the send, the utility condition code is a communications condition code, regardless of translation errors.

Envelope and Send Errors

When you issue a `PERFORM ENVELOPE AND SEND` or `PERFORM REENVELOPE AND SEND` command, errors can occur during the enveloping process. When an enveloping error occurs, the utility condition code returned is 464 (ENVELOPE) or 480 (REENVELOPE), and the send is not done. If enveloping or reenveloping is successful, the send is issued. The utility condition code is set to a communications condition code if an error occurs during the send.

Combination Receive Errors

When you issue a `PERFORM RECEIVE AND DEENVELOPE` or `PERFORM RECEIVE AND TRANSLATE` command, several error conditions can occur. If an error occurs during a receive, a communications condition code returns. If a receive is successful, deenveloping (and translation) is done. If an error occurs during this process, the corresponding translator extended return code is given as the utility condition code. When multiple `WHERE` clauses are specified, giving multiple requestor IDs, the first error (except for an empty mailbox) terminates the command, and this error is given as the utility condition code.

Application Programming Interface Return Codes

The following sections describe the application programming interface return codes.

Initialization Return Codes

The results of the initialization request are posted in the return code and extended return code fields of the common control block. Table B-14 describes initialization return codes.

Table B-14. Initialization Return Codes

Return code	Ext code	Description
00	00	Initialization was successful.
04	04	No service table defined (failure to locate FXXZIN load module).
04	08	Insufficient virtual storage to load programs and tables.
04	12	Failure initializing the EDIT service. This may be caused by an incorrect ZCCBLPID value (one that cannot be matched with a LANGPROF member). In VSAM installations, this error may occur because of the inability to access PROFDEF, PROFDAT, EDITPXRF, TABLDEF, or TABLDAT VASM files. In DB2 installations, this error may occur because of the inability to access the profile or translation/validation tables (in many cases, this could be a DB2 timestamp error, meaning that the timestamp in load module EDIPSMD is different from its corresponding DBRM timestamp). If a DB2 timestamp error is suspected (SQL code - 818), the DB2 plan must be rebound. Besides EDIPSMD, there are two other DataInterchange DB2 load modules: EDIRPML and (for CICS users) EDICRIN that could generate DB2 timestamp problems if out of sync.
04	16	A DataInterchange session for the same user is already active.
04	1024	Access to the system is denied.

Negative Return Codes

On any request for a DataInterchange service, you can receive a negative return code in the ZCCBRC field. A negative return code indicates that the service requested was never invoked, and that an error occurred that prevented the service from being called. Table B-15 lists the negative return codes and describes the errors they represent.

Table B-15. Negative Return Codes and Messages

Return code	Ext code	Description
-1		The CCB is not valid for one of the following reasons: <ul style="list-style-type: none">• The CCB is not initialized with the environmental initialization function request.• The CCB has been destroyed.• The program is not using the same CCB used on the initialization request.
-2		The requested service is not known to DataInterchange. The name in the ZSNBNAME field of the SNB does not identify a logical service. Logical service names must be padded with blanks.
-3		The requested service is known, but does not exist in any library, and therefore was not loaded.
-4		Inability to establish environment needed by service because of insufficient virtual storage.
-5	04	The parameter count from the SNB is less than 2. Any call to the service director must pass the SNB and CCB as parameters.
-5	08	The language the service is written in is not supported.
-8		An abend has been detected by DataInterchange for CICS. The extended return code is the EBCDIC representation of the CICS abend code.

DataInterchange Termination Return Codes

Table B-16 describes errors that can occur when you terminate the DataInterchange environment.

Table B-16. Termination Return Codes

Return code	Description
0	Termination was successful.
>0	Termination failed, try again.
<0	Not a valid common control block address or an incorrect name in the ZSNBNAME field.

Translation Return Codes

Many translator error conditions can be detected and reported. The following lists the categories of errors, with specific examples:

Error	Example
Warning conditions	Structure not used, end of file
Data element errors	Data element too short or too long

Segment errors	Mandatory segment missing, unrecognized segment
Transaction errors	Control number mismatch
Group errors	Control number mismatch, password invalid
Interchange errors	Control number mismatch, trading partner not known
Invalid data errors	Not an interchange
Environmental or program errors	File read error, insufficient virtual storage

More than one error can occur during translation of a transaction. The extended return code indicates the level at which an error occurred, such as the data element or segment level. The most serious errors are returned in the ZCCBRC and ZCCBERC fields.

A particular translation error (return code and extended return code combination) can have many causes. Each cause has a unique message ID and a unique error code. DataInterchange logs each message when it detects an error.

API programs can use the ERRCD field of the translator control block (TRCB) to retrieve unique codes. See “Translator Control Block (TRCB)” on page A-6 for more detailed information on these errors.

When translation errors occur during a receive function, the functional acknowledgment to the sender, if requested, indicates the error code identified by the message. Refer to your standards documentation for codes returned in functional acknowledgments. Use the TRANSACTION STORE FACILITY option on the Administrator's Menu to obtain information about functional acknowledgments. For more information on the Transaction Store Facility, see the *DataInterchange Administrator's Guide*.

Normal or Warning Conditions

Table B-17 shows the return codes DataInterchange uses when it detects no errors.

Table B-17 (Page 1 of 2). Translation with Normal or Warning Return Codes

Return Code	Ext Code	Msg ID	Unique Code	Description
00	00	None	0	Processing of requested function completed normally.
00	00	TR0841	2	Structure not defined in the application data format.
00	00	TR0403	3	Structure not used in the mapping.
00	00	TR0404	4	Structure defined as part of parent (not passed separately).
00	00	TR0405	5	Raw data structure could not be identified.
00	00	TR0406	6	Raw data structure received but structure defined to start the translation not yet received.
00	00	TR0407	7	Mismatching loop ID values in LS and LE segments.
00	00	TR0408	8	LS segment does not have a corresponding LE segment.
00	00	TR0409	9	No mapping provided for hierarchical code and parent hierarchical code combination.
00	00	TR0824	505	Transmission of envelope to communications failed.
04	01	TR0401	1	End of file, no more envelopes in network queue to receive.

Data Element Errors

Table B-18 on page B-20 shows the errors that occur at the data element level. The translation might still be acceptable based on the acceptable error level established in the send or receive usage record.

Table B-18 (Page 1 of 2). Data Element Level Translation Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
08	01	None	0	The highest severity of error detected during translation of this transaction is at the data element level.
08	01	TR0001	101	A mapped mandatory data element is blank for a segment containing data in other data elements.
08	01	TR0002	102	Data element is too long.
08	01	TR0003	103	Data element is too short.
08	01	TR0004	104	Code in ID type field not found in validation table.
08	01	TR0005	105	Code in ID type field not found in translation table.
08	01	TR0006	106	User exit for data element failed. Exit routine returned an error.
08	01	TR0007	107	Invalid date format. Unable to reformat date according to customized date edit number.
08	01	TR0008	108	Data element conversion failed. Format of data in input buffer conflicts with mapping.
08	01	TR0009	109	Standard length exceeds application length.
08	01	TR0010	111	Paired conditionality in segment not satisfied.
08	01	TR0011	112	Required conditionality in segment not satisfied.
08	01	TR0012	113	Mutually exclusive conditionality in segment not satisfied.
08	01	TR0013	114	Conditional conditionality in segment not satisfied.
08	01	TR0014	110	Conditional-paired conditionality in segment not satisfied.
08	01	TR0015	115	Mandatory composite field missing.
08	01	TR0016	116	Data element validation failed.
08	01	TR0017	117	Attempt to increment accumulator will exceed maximum size.
08	01	TR0018	118	Attempt to add a value to an accumulator will exceed maximum size.
08	01	TR0023	119	Data in application data format has been overlaid.
08	01	TR0024	120	Field data that was not mapped has been received.

Segment Errors

Table B-19 on page B-21 shows the errors that occur on the segment level. The translation might still be acceptable based on the acceptable error level established in the send or receive usage record.

Table B-19. Segment Level Translation Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
08	02	None	0	The highest severity of error detected during translation of this transaction is at the segment level.
08	02	TR0019	207	Error getting storage while processing binary segment.
08	02	TR0020	208	Error opening a file while processing binary segment.
08	02	TR0021	209	Error reading a file while processing binary segment.
08	02	TR0022	210	Error writing a file while processing binary segment.
08	02	TR0050	201	A segment contains more elements than the standard allows.
08	02	TR0051	202	Unrecognized segment ID. The segment is not defined for the transaction.
08	02	TR0052	203	A mandatory, mapped segment for this trading partner is missing.
08	02	TR0053	204	Occurrences of a repeating segment in the input data exceed the maximum use count for the structure in the data format.
08	02	TR0054	205	Loop repetition count exceeded. Loop repeats more times than definition specifies.
08	02	TR0055	206	Segment repetition count exceeded. Segment repeats more times than definition specifies.
08	02	TR0056	211	Unexpected segment data received.
08	02	TR0057	215	Application data received out of sequence.
08	02	TR0058	212	Creation of standard loop has been aborted.
08	02	TR0059	213	Creation of a repeating segment has been aborted.
08	02	TR0060	214	Creation of a segment has been aborted.

Transaction Errors

This section describes the transaction errors. Table B-20 on page B-22 shows errors that indicate a serious problem with the syntax of a transaction. Table B-21 on page B-22 shows the environment-related errors. Such errors include a situation in which DataInterchange is unable to locate a map or control string, which is necessary to process the transaction. An API program receives a single 8,3 return code, indicating that a transaction is being skipped.

Table B-20. Transaction Level Syntax Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
08	03	None	0	The highest severity of error detected during translation of this transaction is at the transaction set level.
08	03	TR0025	326	Duplicate transaction within group or interchange.
08	03	TR0101	302	Transaction set control numbers do not match in header and trailer.
08	03	TR0103	304	Transaction set trailer contains invalid segment count.

Table B-20. Transaction Level Syntax Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
08	03	TR0105	315	Security profile member could not be found for encrypted transaction.
08	03	TR0106	316	Authentication produced a MAC value that does not match the received value.
08	03	TR0107	317	S2S segment without S2E. Incomplete security segments for received transaction.
08	03	TR0108	318	S2E segment without S2S. Incomplete security segments for received transaction.
08	03	TR0207	506	Envelope is not defined correctly. Envelope definition damaged.
08	03	TR0208	507	Envelope is not defined correctly. Envelope definition damaged.
08	03	TR1257	335	Mandatory composite missing in service segment.
08	03	TR1258	336	Service segment data element too long.
08	03	TR1259	337	Service segment data element value not defined in validation table.
08	03	TR1260	338	Service segment data element value not consistent with data type.
08	03	TR1261	339	Mandatory data element missing in service segment.
08	03	TR1262	340	Service segment data element too small.

Table B-21 (Page 1 of 2). Transaction Level Environmental Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
08	03	TR0104	305	Receive trading partner transaction not found.
08	03	TR0109	319	Attempt to receive translate (TRANSLATE TO APPLICATION) a transaction that was previously send translated (TRANSLATE TO STANDARD).
08	03	TR0110	320	Invalid transaction handle.
08	03	TR0111	321	Attempt to envelope a received transaction.
08	03	TR0112	322	Attempt to translate a transaction that does not have a group or interchange segment associated with it.
08	03	TR0113	324	Attempt to envelope a transaction from a bundle without enveloping the controlling transaction.
08	03	TR0114	325	Attempt to envelope a transaction that was not translated successfully.
08	03	TR0115	327	Transaction control number assigned by application not valid.
08	03	TR0116	328	Transaction control number assigned by application is a duplicate.
08	03	TR0117	330	Call to transaction services for details failed.
08	03	TR0118	331	Error attempting to get image from Transaction Store.

Table B-21 (Page 2 of 2). Transaction Level Environmental Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
08	03	TR0119	332	Request to get an image that does not exist.
08	03	TR0120	333	Request to get an image but the base has not been established.
08	03	TR0121	334	Attempt to envelope a transaction with an incorrect status.
08	03	TR0155	406	Authentication routine required but not provided.
08	03	TR0156	407	Encryption routine required but not provided.
08	03	TR0157	408	Filtering routine required but not provided.
12	16	TR0816	???	Invalid function code. Function code is not: 1, 2, 3, 131, 111, 212, 211, 213, 214, 215 or 1000.
08	17	TR0817	???	Transaction processor did not terminate because it was not active.
08	03	TR0818	307	The translator could not write to the event log. The log might be full.
08	03	TR0820	308	Send trading partner transaction not for this trading partner, application, and direction.
08	03	TR0821	306	Application data format not found.
08	03	TR0822	309	Control string not found for this transaction.
08	03	TR0825	310	Trading partner profile member not found.
08	03	TR0826	311	Sender ID in envelope profile member is blank.
08	03	TR0830	312	Input data block contains no data. Application data length is 0.
08	03	TR0834	313	Overran output buffer. User's output buffer is too small for the data.
08	03	TR0848	329	Failed to load a user exit routine.
08	03	TR0850	323	Standard delimiters are not unique. Update delimiters in envelope standard or trading partner profile.
08	03	SA0042	314	Access denied to function within resource.

Group Errors

Table B-22 on page B-24 shows the errors that indicate a serious problem with an entire functional group within an interchange. None of the transactions within the group are processed. An API program receives a single 8,4 return code, indicating that a group is being skipped.

Table B-22. Group Level Translation Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
08	04	None	0	An error is detected in the functional group header or trailer. No transactions in the functional group are translated. Processing continues.
08	04	TR0100	301	Transaction set header is missing or invalid.
08	04	TR0102	303	Transaction set trailer is missing or invalid.

Table B-22. Group Level Translation Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
08	04	TR0107	409	S1S segment without S1E.
08	04	TR0108	410	S1E segment without S1S.
08	04	TR0151	402	Functional group control numbers do not match in header and trailer.
08	04	TR0153	404	Functional group trailer contains invalid transaction count.
08	04	TR0154	405	Authentication failed for group.
08	04	TR0155	406	Authentication routine required but not provided.
08	04	TR0156	407	Encryption routine required but not provided.
08	04	TR0157	408	Filtering routine required but not provided.
08	04	TR0158	411	There is a duplicate group within the interchange.
08	04	TR0207	506	Envelope is not defined correctly. Envelope definition damaged.
08	04	TR0208	507	Envelope is not defined correctly. Envelope definition damaged.
08	04	TR1257	412	Mandatory composite missing in service segment.
08	04	TR1258	413	Service segment data element too long.
08	04	TR1259	414	Service segment data element value not defined in validation table.
08	04	TR1260	415	Service segment data element value not consistent with data type.
08	04	TR1261	416	Mandatory data element missing in service segment.
08	04	TR1262	417	Service segment data element too small.

Interchange Errors

Table B-23 shows the errors indicating the translator was able to isolate an interchange to process, but there is a serious problem with that interchange, and none of the transactions are processed. An API program receives a single 8,5 return code, indicating that an interchange is being skipped.

Table B-23 (Page 1 of 2). Interchange Level Translation Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
08	05	None	0	An error is detected in the interchange header or trailer. No transactions in the interchange are translated. Processing continues.
08	05	TR0150	401	Functional group header is missing or invalid.
08	05	TR0152	403	Functional group trailer is missing or invalid.
08	05	TR0201	501	Interchange header contains invalid sender ID. Cannot locate trading partner profile member.
08	05	TR0203	502	Interchange control numbers do not match in header and trailer.

Table B-23 (Page 2 of 2). Interchange Level Translation Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
08	05	TR0205	503	Interchange trailer contains invalid functional group count.
08	05	TR0206	504	Password does not match password in trading partner profile.
08	05	TR0207	506	Interchange is not defined correctly or the interchange is badly damaged and cannot be parsed.
08	05	TR0208	507	Received interchange segment contains invalid data.
08	05	TR0209	508	Expected delimiter not found. Encrypted data not immediately followed by segment delimiter.
08	05	TR0210	509	Envelope definition not found for received interchange.
08	05	TR0211	510	A duplicate interchange was detected and is being skipped.
08	05	TR1257	511	Mandatory composite missing in service segment.
08	05	TR1258	512	Service segment data element too long.
08	05	TR1259	513	Service segment data element value not defined in validation table.
08	05	TR1260	514	Service segment data element value not consistent with data type.
08	05	TR1261	515	Mandatory data element missing in service segment.
08	05	TR1262	516	Service segment data element too small.

Invalid Data Errors

Table B-24 shows the errors that indicate that the translator cannot identify the data it is reading from the file as standard data. The translator expects the ISA, UNB, SCH, ICS, BG, or GS segment to signal the start of standard data. Anything before, between, or after any of these valid interchanges is flagged as an error.

Table B-24. Invalid Data in the Input File

Return Code	Ext Code	Msg ID	Unique Code	Description
08	06	None	0	Invalid data found in the input file. Data does not match the format required by the standard.
08	06	TR0200	601	Interchange envelope header is missing or invalid.
08	06	TR0204	602	Interchange envelope trailer is missing or invalid.
08	06	TR0842	603	No standard data found in input file.
08	06	TR0202	604	Interchange header found while looking for a trailer.

Environmental and Program Errors

Table B-25 on page B-26 shows the errors considered so serious that the translator logs the appropriate message for the error and ends (TRABORT flag in TRCB is set to Y).

Table B-25 (Page 1 of 2). Environmental and Program Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
12	10	TR0810	901	Profile read for update failed.
12	11	TR0811	902	Profile write failed.
12	12	TR0812	903	Parameter count is invalid.
12	15	TR0815	904	A request to get main storage failed.
12	16	TR0816	905	Invalid function code.
12	17	TR0817	906	The transaction processor did not end because it is not active. This error also occurs if input records are out of sequence, such as a C record followed by a blank record.
12	27	TR0827	908	You cannot perform this function now. Function code not valid at this time.
12	28	TR0828	909	Input data block size invalid. Must be at least 32K.
12	29	TR0829	910	Output data block size invalid. Must be at least 32K.
12	32	TR0832	911	Transaction Store call failed.
12	36	TR0836	912	Repository read failed.
12	38	TR0838	913	QSAM failed to open file.
12	39	TR0839	914	Requestor profile not found.
12	40	TR0840	915	Standards profile member not found.
12	43	TR0843	907	Receive file name in requestor profile is blank.
12	44	SA0042	921	Access denied to function within resource.
12	46	TR0846	922	Raw data control string not found.
12	47	TR0847	923	Failed to find security profile member.
12	48	TR0848	924	Failed to load a user exit routine.
12	49	TR0849	925	Error returned by user exit routine.
12	51	TR0851	926	An application data format ID is required when raw data is specified.
12	01	TR1201	916	A program error occurred during anchor processing.
12	02	TR1202	917	Unable to free main storage.
12	03	TR1203	918	Unable to read repository.
12	05	TR1205	919	QSAM failed to read file.
12	06	TR1206	920	QSAM failed to close file.
12	07	TR1207	927	DataInterchange ended because of severe error generating functional acknowledgments.
12	08	TR1208	928	DataInterchange ended because of severe error updating the management reporting databases.
12	09	TR1209	929	Error reading envelope control string.
12	52	TR1252	930	Error reading target ADF control string.
12	53	TR1253	931	No beginning/ending structure.
12	54	TR1254	932	No internal trading partner ID value.
12	55	TR1255	933	User exit (IUSEREXIT) not defined.

Table B-25 (Page 2 of 2). Environmental and Program Errors

Return Code	Ext Code	Msg ID	Unique Code	Description
12	56	TR1256	934	IUSEREXIT returned an error.
12	57	TR1257	935	Mandatory composite missing in service segment.
12	58	TR1258	936	Service segment data element too long.
12	59	TR1259	937	Service segment data element value not defined in validation table.
12	60	TR1260	938	Service segment data element value not consistent with data type.
12	61	TR1261	939	Mandatory data element missing in service segment.
12	62	TR1262	940	Service segment data element too small.
12	63	TR1263	941	Database error obtaining lock.

Communication Return Codes

This section describes the codes that can be returned by the communication routine. Program your application to inspect the return code after each call and take appropriate action. When the communications routine receives an error from the network program, it logs an error (message ID VN1015) and returns with extended return code 15, and a return code of either 4, 8, or 12 based on the severity of the error.

It is the responsibility of the network message handler to process the network output file and set the **NPSEVER** and **NPERRCD** fields of the CMCB to indicate the severity and error code for any errors contained in the network output file. For more information, see "Message Handler" on page 6-13. Communications will then log the VN1015 message with a symptom string that includes:

```
NETWORK PROGRAM_RC=xx NETWORK PROGRAM_ERC=yyyy
```

Where:

xx is the **NPSEVER** value

yyyy

is the **NPERRCD** value

Communication return codes are divided into four categories. Each type of return code contains several extended return codes that provide detailed information about the error.

Return Code Explanation

0	Successful completion of the communication request.
4	A warning message. DataInterchange does not end a send or receive operation for this return code.
8	An error has occurred and communication processing might discontinue.
12	A severe error has occurred and communication processing has ended.

The following tables describe the meanings of the extension codes for each return code.

Table B-26. Communications—Warnings

Return Code	Ext Code	Description
04	0000	Default values used for trading partner profile.
04	0001	Network profile does not provide name of send/receive program.
04	0002	QSAM error prevented completion of request to clear the file (applies to sending only).
04	1015	Invalid parameter passed to network program. Check network profile, trading partner profile, or communication interface control block.
04	1040	No data received on a receive request.
04	1041	Not able to process the network command output file.

Table B-27. Communications—Nonsevere Errors

Return Code	Ext Code	Description
08	0001	No profile data passed.
08	0002	Profile ID not found.
08	0003	Profile key not found.
08	1001	Invalid function code passed.
08	1002	Trading partner ID not found in communication interface control block.
08	1003	Profile ID not found.
08	1004	Profile key not found.
08	1007	Open file error.
08	1012	Mismatched network operation and network ID.
08	1013	Invalid account type passed.
08	1014	Invalid data type passed.
08	1015	Invalid parameter passed to network program.
08	1024	A restart was requested but the network is not in a restart situation — request was terminated.
08	1025	A restart is required for the network — request was terminated.
08	1026	A restart was requested but restart is not supported — request was terminated.

Table B-28 (Page 1 of 2). Communications—Severe Errors

Return Code	Ext Code	Description
12	0004	Profile services error.
12	0005	Failed to start communication routine.
12	1005	Profile services error.
12	1006	Get storage failed.
12	1008	Write file error.
12	1009	Close file error.
12	1010	Free storage failed.

Table B-28 (Page 2 of 2). Communications—Severe Errors

Return Code	Ext Code	Description
12	1011	Network program not found.
12	1015	Invalid parameter passed to the network program.
12	1016	Network program execution failed.
12	1018	An error occurred while attempting a restart.

Status Update Return Codes

The results of a status update request are posted in the return code and extended return code fields of the common control block. Table B-29 describes the status update return codes.

Table B-29. Status Update Return Codes

Return code	Ext code	Description
0	0	Status update was successful.
8		Status update failed.
8	210	No interchange was found.
8	211	Internal program error
8	212	Error attempting to update the database.

Initialize SYNC

The results of the initialization request are posted in the return code and extended return code fields of the common control block. Table B-30 describes the initialize SYNC return codes.

Table B-30. Initialize SYNC Return Codes

Return code	Ext code	Description
0	0	Initialization was successful.
12	12	Initialization failed due to insufficient virtual storage.

COMMIT Work

The results of the COMMIT work request are posted in the return code and extended return code fields of the common control block. Table B-31 on page B-30 describes the COMMIT work return codes.

Table B-31. COMMIT Work Return Codes

Return code	Ext code	Description
0	0	COMMIT work successful.

Table B-31. COMMIT Work Return Codes

Return code	Ext code	Description
4	1	COMMIT ignored. The request to COMMIT was ignored for one of the following reasons: <ul style="list-style-type: none">• An interval of -1 was established on the SYNCPOINT initialization function.• The interval has not been reached yet.
12	N	COMMIT failed. The SQLCODE from DB2 indicating why the COMMIT was not honored.

ROLLBACK Work

The results of the ROLLBACK work request are posted in the return code and extended return code fields of the common control block. Table B-32 describes the ROLLBACK work return codes.

Table B-32. ROLLBACK Work Return Codes

Return code	Ext code	Description
0	0	ROLLBACK work successful.
12	N	ROLLBACK work failed. The SQLCODE from DB2 indicating the reason for ROLLBACK failure.

Appendix C. Copy Books and Include Files

This appendix provides a hardcopy listing of some of the control blocks that may be used by programs using the application programming interface (API). All of the control blocks are provided in softcopy format in the following distribution load libraries.

Library Name	Description
EDI.V3R1M0.SEDIASM1	ASSEMBLER and COPY samples
EDI.V3R1M0.SEDICBL1	COBOL and CBLCPY samples
EDI.V3R1M0.SEDICCC1	C and H samples
EDI.V3R1M0.SEDIPLI1	PL/I and INCLUDE samples

The following table provides the names and descriptions of members that are provided in each of the libraries listed previously. Use these members as a starting point for copying books that are tailored for use by your installation.

Table C-1 (Page 1 of 2). Library Members

EDICCB	Common Control Block (CCB)
EDICMCB	Communications Control Block (CMCB)
EDIDBLK	Translator Data Block (TRIDB, TRODB, DATABLK)
EDIDEA	Data Extract Application Record
EDIDEE	Data Extract Interchange Record
EDIDEG	Data Extract Group Record
EDIDENTA	Network Activity Record
EDIDER	Data Extract Image Record
EDIDET	Data Extract Transaction Record
EDIDETPA	Transaction Activity Record
EDIDETPC	Trading Partner Capability Record
EDIDETPI	Trading Partner Information Record
EDIFCB	Function Control Block (FCB)
EDIFFC	DataInterchange Utility C Record
EDIFFD	DataInterchange Utility D Record
EDIFFDU	DataInterchange Utility Control Information Block
EDIFFE	DataInterchange Utility E Record
EDIFFG	DataInterchange Utility G Record
EDIFFI	DataInterchange Utility I Record
EDIFFQ	DataInterchange Utility Q Record
EDIFFT	DataInterchange Utility T Record
EDIFFZ	DataInterchange Utility Z Record
EDIHBLK	Translator Huge Block (TRIDB, TRODB)
EDINPDB	Network Profile Data Block (NPDB)
EDIRQDB	Requestor Profile Data Block (REQDB)

Table C-1 (Page 2 of 2). Library Members

EDISNB	Service Name Block (SNB)
EDISPDB	Security Profile Data Block (SPDB)
EDISUDB	Status Update Data Block
EDISUK0	Status Update 210 Key Block
EDISUK1	Status Update 211 Key Block
EDISUK2	Status Update 212 Key Block
EDISUK3	Status Update 213 Key Block
EDISUK4	Status Update 214 Key Block
EDISUK5	Status Update 215 Key Block
EDITPDB	Trading Partner Profile Data Block (TPPDB)
EDITRCB	Translator Control Block (TRCB)
EDIVNMH	VANICICS commarea to message handler
EDIVNNP	VANICICS commarea to network program

In addition, the EDI.V1R4M0.SEDICBL1 library contains the following example programs.

Table C-2. Library Members

EDISAMR1	Uses a PERFORM ENVELOPE DATA EXTRACT to create a report based on Application Control Values.
EDISAMS1	Uses a PERFORM ENVELOPE DATA EXTRACT to create a report based on Application Control Values.
EDISAMT1	Uses a PERFORM TRADING PARTNER CAPABILITY DATA EXTRACT to provide a report on the transaction capability of trading partners.

Copy Books for COBOL

COBOL CCB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Common Control Block (CCB)                                                *
*                                                                                   *
* Length: 608 bytes                                                                *
*                                                                                   *
*****
01 CCB-DATA.
   03 ZCCBLL          PIC S9(4) COMP-4.
   03 ZCCBID          PIC S9(4) COMP-4.
   03 ZCCBEYE         PIC X(8).
   03 ZCCBRC          PIC S9(9) COMP-4.
   03 ZCCBERC         PIC S9(9) COMP-4.
   03 ZCCBSID         PIC X(8).
   03 ZCCBUID         PIC X(8).
   03 ZCCBAID         PIC X(8).

```


	03 ZCCBCID	PIC X(8).
	03 ZCCBXFID	PIC S9(4) COMP-4.
	03 ZCCBLPID	PIC X(6).
	03 ZCCBCPID	PIC S9(9) COMP-4.
	03 ZCCBRSV	PIC S9(9) COMP-4.
	03 ZCCBCCXP	PIC S9(9) COMP-4.
	03 ZCCBCABP	PIC S9(9) COMP-4.
	03 ZCCBDBID	PIC X(4).
	03 ZCCBDBPL	PIC X(8).
	03 ZCCBDBUI	PIC X(8).
	03 ZCCBDBPW	PIC X(18).
	03 ZCCBDSV1	PIC X(26).
	03 ZCCBRSV2	PIC S9(9) COMP-4 OCCURS 117 TIMES.

COBOL SNB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Service Name Block (SNB)                                                 *
*                                                                                   *
* Length: 32 bytes                                                                *
*                                                                                   *
*****
01 SNB-DATA.
   03 ZSNBLL          PIC S9(4) COMP-4.
   03 ZSNBID          PIC S9(4) COMP-4.
   03 ZSNBEYE        PIC X(8).
   03 ZSNBNAME        PIC X(8).
   03 ZSNBNDX        PIC S9(9) COMP-4.
   03 ZSNBPC          PIC S9(4) COMP-4.
   03 ZSNBFLG0        PIC X(1).
   03 ZSNBFLG1        PIC X(1).
   03 ZSNBFANC        PIC S9(9) COMP-4.

```

COBOL FCB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Function Code Block (FCB)                                                 *
*                                                                                   *
* Length: 4 bytes                                                                *
*                                                                                   *
*****
01 FCB-DATA.
   03 ZFCBLL          PIC S9(4) COMP-4.
   03 ZFCBFUNC        PIC S9(4) COMP-4.

```

COBOL CMCB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Communications Control Block (CMCB)                                       *
*                                                                                   *
* Length: 254 bytes                                                                *
*                                                                                   *
*****
01 CMCB-DATA.
    03 CMCB-BLKLEN          PIC S9(4) COMP-4.
    03 CMCB-RESERV1         PIC S9(4) COMP-4.
    03 CMCB-BLKNAME         PIC X(8).
    03 CMCB-TPNICKNM        PIC X(16).
    03 CMCB-NETID           PIC X(8).
    03 CMCB-NETOP           PIC X(8).
    03 CMCB-REQID           PIC X(16).
    03 CMCB-SEQNUM          PIC X(5).
    03 CMCB-CONTRCV         PIC X(1).
    03 CMCB-FTYPE           PIC X(2).
    03 CMCB-FILERCV         PIC X(1).
    03 CMCB-RESERV2         PIC X(5).
    03 CMCB-CLRFILE         PIC X(1).
    03 CMCB-DATAFMT         PIC X(1).
    03 CMCB-ACCTYP          PIC X(1).
    03 CMCB-DATATYP         PIC X(1).
    03 CMCB-RECVTYP         PIC X(1).
    03 CMCB-DCIND           PIC X(1).
    03 CMCB-ACKIND          PIC X(1).
    03 CMCB-RESRECL         PIC X(1).
    03 CMCB-SCRIPT          PIC X(8).
    03 CMCB-ENAME           PIC X(8).
    03 CMCB-MSGNAME         PIC X(8).
    03 CMCB-FILENAME        PIC X(56).
    03 CMCB-CANS            PIC X(6).
    03 CMCB-CANST           PIC X(6).
    03 CMCB-CANED           PIC X(6).
    03 CMCB-CANET           PIC X(6).
    03 CMCB-TMZONE          PIC X(1).
    03 CMCB-MODEM           PIC X(1).
    03 CMCB-NPSSCDE         PIC X(5).
    03 CMCB-NPESCDE         PIC X(5).
    03 CMCB-NPERRCD         PIC X(5).
    03 CMCB-NPSEVER         PIC X(2).
    03 CMCB-BLKTYPE         PIC X(1).
    03 CMCB-FQUEUED         PIC X(1).
    03 CMCB-FMSG           PIC X(1).
    03 CMCB-FFILE           PIC X(1).
    03 CMCB-FEDIX           PIC X(1).
    03 CMCB-FEDIE           PIC X(1).
    03 CMCB-FEDIU           PIC X(1).
    03 CMCB-FEDIG           PIC X(1).
    03 CMCB-FEDII           PIC X(1).
    03 CMCB-FEDIT          PIC X(1).
    03 CMCB-FCANCEL         PIC X(1).

```

	03 CMCB-FCLASS	PIC X(1).
	03 CMCB-FACK	PIC X(1).
	03 CMCB-FSYSMSG	PIC X(1).
	03 CMCB-FRCVBTP	PIC X(1).
	03 CMCB-FRESTART	PIC X(1).
	03 CMCB-FNOUSERID	PIC X(1).
	03 CMCB-FACCTSEP	PIC X(1).
	03 CMCB-DDCOLON	PIC X(3).
	03 CMCB-RESERV3	PIC X(2).
	03 CMCB-ADMTYPE	PIC X(2).
	03 CMCB-UNIQID	PIC X(8).
	03 CMCB-SAPUPDT	PIC X(1).
	03 CMCB-FSENTNET	PIC X(1).
	03 CMCB-RESERV4	PIC X(14).

COBOL TRCB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Translator Control Block (TRCB)                                           *
*                                                                                   *
* Length: 1536 bytes                                                             *
*                                                                                   *
*****
01 TRCB-DATA.
    03 TRCB-BLKLEN          PIC S9(4) COMP-4.
    03 TRCB-RSRVD1         PIC S9(4) COMP-4.
    03 TRCB-BLKNME         PIC X(8).
    03 TRCB-REQID          PIC X(16).
    03 TRCB-APPFILE        PIC X(8).
    03 TRCB-ATFID          PIC X(16).
    03 TRCB-ATSID          PIC X(16).
    03 TRCB-EJECT          PIC X(1).
    03 TRCB-INTPID         PIC X(35).
    03 TRCB-APPCTLNUM      PIC X(32).
    03 TRCB-TRNID          PIC X(16).
    03 TRCB-TEST           PIC X(1).
    03 TRCB-IHCTL          PIC X(9).
    03 TRCB-GHCTL          PIC X(9).
    03 TRCB-THCTL          PIC X(9).
    03 TRCB-ENVTYPE        PIC X(1).
    03 TRCB-BLKTYPE        PIC X(1).
    03 TRCB-DUPTRAN        PIC X(1).
    03 TRCB-ITPBREAK       PIC X(1).
    03 TRCB-REQSIZE        PIC S9(9) COMP-4.
    03 TRCB-XPANDED        PIC X(1).
    03 TRCB-NEWENV         PIC X(1).
    03 TRCB-NEWGRP         PIC X(1).
    03 TRCB-NEWTRN         PIC X(1).
    03 TRCB-QSIZE          PIC S9(9) COMP-4.
    03 TRCB-ESIZE          PIC S9(9) COMP-4.
    03 TRCB-GRPNUM         PIC S9(9) COMP-4.
    03 TRCB-TRNNUM         PIC S9(9) COMP-4.
    03 TRCB-SEGNUM         PIC S9(9) COMP-4.

```

	03 TRCB-TRNGRP	PIC S9(9) COMP-4.
	03 TRCB-SEGTRN	PIC S9(9) COMP-4.
	03 TRCB-ERRNUM	PIC S9(9) COMP-4.
	03 TRCB-QBT	PIC X(8).
	03 TRCB-IHXCTL	PIC X(14).
	03 TRCB-ISYNTAXID	PIC X(4).
	03 TRCB-ISYNTAXVER	PIC X(1).
	03 TRCB-ISIDQUAL	PIC X(4).
	03 TRCB-ISID	PIC X(35).
	03 TRCB-ISENDNAME	PIC X(14).
	03 TRCB-IREVROUT	PIC X(14).
	03 TRCB-IRIDQUAL	PIC X(4).
	03 TRCB-IRID	PIC X(35).
	03 TRCB-IRECVNAME	PIC X(14).
	03 TRCB-IROUTEADDR	PIC X(14).
	03 TRCB-IDATE	PIC X(6).
	03 TRCB-ITIME	PIC X(6).
	03 TRCB-IVERREL	PIC X(5).
	03 TRCB-IGT	PIC X(6).
	03 TRCB-ITT	PIC X(6).
	03 TRCB-IST	PIC X(10).
	03 TRCB-IBT	PIC X(8).
	03 TRCB-ISPW	PIC X(14).
	03 TRCB-IAPREF	PIC X(14).
	03 TRCB-ISTDID	PIC X(4).
	03 TRCB-RSRVD2	PIC X(1).
	03 TRCB-IPRIOR	PIC X(1).
	03 TRCB-ICOMMAGREE	PIC X(35).
	03 TRCB-GHXCTL	PIC X(14).
	03 TRCB-GFGID	PIC X(6).
	03 TRCB-GSIDQUAL	PIC X(4).
	03 TRCB-GSID	PIC X(35).
	03 TRCB-GRIDQUAL	PIC X(4).
	03 TRCB-GRID	PIC X(35).
	03 TRCB-GDATE	PIC X(6).
	03 TRCB-GTIME	PIC X(6).
	03 TRCB-GVER	PIC X(12).
	03 TRCB-GREL	PIC X(12).
	03 TRCB-GTT	PIC X(6).
	03 TRCB-GAPW	PIC X(14).
	03 TRCB-GRESPAGENCY	PIC X(2).
	03 TRCB-RSRVD3	PIC X(12).
	03 TRCB-THXCTL	PIC X(14).
	03 TRCB-TTC	PIC X(6).
	03 TRCB-TVER	PIC X(6).
	03 TRCB-TREL	PIC X(6).
	03 TRCB-TST	PIC X(10).
	03 TRCB-LASTINENV	PIC X(1).
	03 TRCB-XACFIELD	PIC X(35).
	03 TRCB-FABUILT	PIC X(1).
	03 TRCB-QGTNUM	PIC S9(9) COMP-4.
	03 TRCB-QTTNUM	PIC S9(9) COMP-4.
	03 TRCB-QSTNUM	PIC S9(9) COMP-4.
	03 TRCB-QGT	PIC X(6).
	03 TRCB-QTT	PIC X(6).
	03 TRCB-QST	PIC X(10).
	03 TRCB-FASPM	PIC X(8).

	03 TRCB-ENVCHK	PIC X(1).
	03 TRCB-TRNSTAT	PIC X(1).
	03 TRCB-APTYPE	PIC X(2).
	03 TRCB-TSKEY	PIC X(10).
	03 TRCB-TSKEYU	PIC X(20).
	03 TRCB-MAPKEY	PIC X(16).
	03 TRCB-BATCHID	PIC X(8).
	03 TRCB-ENVLDATE	PIC X(8).
	03 TRCB-TRXLIFE	PIC S9(4) COMP-4.
	03 TRCB-IMGLIFE	PIC S9(4) COMP-4.
	03 TRCB-HOLDFLAG	PIC X(1).
	03 TRCB-BNDLFLAG	PIC X(1).
	03 TRCB-RAWDATA	PIC X(1).
	03 TRCB-ENVLDELAY	PIC X(1).
	03 TRCB-TRXACCEPT	PIC X(1).
	03 TRCB-TRABORT	PIC X(1).
	03 TRCB-FILEID	PIC X(8).
	03 TRCB-DSNAME	PIC X(56).
	03 TRCB-QNETID	PIC X(8).
	03 TRCB-QPTTOPT	PIC X(1).
	03 TRCB-QSRPGM	PIC X(1).
	03 TRCB-QDDNAME	PIC X(8).
	03 TRCB-FUNACKFLE	PIC X(8).
	03 TRCB-QTPNICK	PIC X(16).
	03 TRCB-QRC	PIC S9(4) COMP-4.
	03 TRCB-QERC	PIC S9(4) COMP-4.
	03 TRCB-ERRCDES	PIC S9(4) COMP-4 OCCURS 10 TIMES.
	03 TRCB-INMEMTRANS	PIC S9(4) COMP-4.
	03 TRCB-NOCOMMIT	PIC X(1).
	03 TRCB-SCOPE	PIC X(1).
	03 TRCB-TPNICK	PIC X(16).
	03 TRCB-CONCATENATE	PIC X(1).
	03 TRCB-ASSERTLVL	PIC X(1).
	03 TRCB-RAWDATAOUT	PIC X(1).
	03 TRCB-FIXEDTRX	PIC X(1).
	03 TRCB-MRREQID	PIC X(16).
	03 TRCB-ERRFILTER	PIC X(80).
	03 TRCB-FFILEID	PIC X(8).
	03 TRCB-VARTRACE	PIC X(1).
	03 TRCB-EXPTRACE	PIC X(1).
	03 TRCB-SSEGVAL	PIC X(1).
	03 TRCB-TRXFACODE	PIC X(1).
	03 TRCB-IUSEREXIT	PIC X(8).
	03 TRCB-IUSERAREA	PIC S9(9) COMP-4.
	03 TRCB-IUSERACCESS	PIC X(1).
	03 TRCB-IUSERTYPE	PIC X(2).
	03 TRCB-MAPCHAIN	PIC X(1).
	03 TRCB-FORCETEST	PIC X(1).
	03 TRCB-ENVPRBRK	PIC X(1).
	03 TRCB-SUSBLKF	PIC X(1).
	03 TRCB-CLRERRS	PIC X(1).
	03 TRCB-FARC	PIC S9(9) COMP-4.
	03 TRCB-FAERC	PIC S9(9) COMP-4.
	03 TRCB-SAPUPDT	PIC X(1).
	03 TRCB-SAPTRX	PIC X(1).
	03 TRCB-SAPCLIENT	PIC X(3).
	03 TRCB-SAPDOCNUM	PIC X(16).

	03 TRCB-SAPSEQ	PIC X(6).
	03 TRCB-ROUTCODE	PIC X(3).
	03 TRCB-FAEREQ	PIC X(1).
	03 TRCB-BOUNDARY	PIC X(1).
	03 TRCB-SUSBLKP	PIC S9(9) COMP-4.
	03 TRCB-CUSERDATA	PIC X(256).
	03 TRCB-APPLTPID	PIC X(16).
	03 TRCB-EXTENDC	PIC X(1).
	03 TRCB-VAXFLAG	PIC X(1).
	03 TRCB-GDATES	PIC X(8).
	03 TRCB-RECOVBAD	PIC X(1).
	03 TRCB-RSRVD4	PIC X(61).

COBOL NPDB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Network Profile Data Block (NPDB)                                         *
*                                                                                   *
* Length: 300 bytes                                                               *
*                                                                                   *
*****
01 CMNP-DATA.
    03 CMNP-BLKLEN          PIC S9(4) COMP-4.
    03 CMNP-RESERV1         PIC S9(4) COMP-4.
    03 CMNP-BLKNME          PIC X(8).
    03 CMNP-NETID           PIC X(8).
    03 CMNP-NETNME          PIC X(30).
    03 CMNP-COMROT          PIC X(8).
    03 CMNP-NETPGM          PIC X(8).
    03 CMNP-PGMPARM         PIC X(57).
    03 CMNP-CMDIN           PIC X(8).
    03 CMNP-CMDLRECL        PIC X(4).
    03 CMNP-QDATA           PIC X(8).
    03 CMNP-DATLRECL        PIC X(4).
    03 CMNP-TMZONE          PIC X(5).
    03 CMNP-SYSTYP          PIC X(8).
    03 CMNP-SYSLVL          PIC X(4).
    03 CMNP-TXTHDR          PIC X(1).
    03 CMNP-CMDOUT          PIC X(8).
    03 CMNP-MSGROUT         PIC X(8).
    03 CMNP-SEQNUM          PIC X(5).
    03 CMNP-NETACKFILE      PIC X(8).
    03 CMNP-NETPHONE        PIC X(32).
    03 CMNP-SCRIPT          PIC X(8).
    03 CMNP-FILLER          PIC X(14).
    03 CMNP-DESCRIPT        PIC X(30).
    03 CMNP-LOGLOCK         PIC X(1).
    03 CMNP-LASTUID         PIC X(17).
    03 CMNP-LASTUDT         PIC S9(9) COMP-4.

```

COBOL TPPDB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Trading Partner Profile Data Block (TPPDB)                               *
*                                                                                   *
* Length: 1532 bytes                                                             *
*                                                                                   *
*****
01 CMTP-DATA.
    03 CMTP-BLKLEN          PIC S9(4) COMP-4.
    03 CMTP-RESERV1         PIC S9(4) COMP-4.
    03 CMTP-BLKNAME        PIC X(8).
    03 CMTP-TPNICKNM       PIC X(16).
    03 CMTP-NETID          PIC X(8).
    03 CMTP-SYSQUAL        PIC X(1).
    03 CMTP-SYSID          PIC X(8).
    03 CMTP-ACCTNUM        PIC X(32).
    03 CMTP-USERID         PIC X(32).
    03 CMTP-ENVLQUAL       PIC X(4).
    03 CMTP-ENVLID         PIC X(35).
    03 CMTP-CONAME         PIC X(40).
    03 CMTP-ADDR1          PIC X(40).
    03 CMTP-ADDR2          PIC X(40).
    03 CMTP-PHONE          PIC X(25).
    03 CMTP-CONTACT        PIC X(30).
    03 CMTP-PASSWORD       PIC X(14).
    03 CMTP-RCVPASS        PIC X(14).
    03 CMTP-SECUID         PIC X(8).
    03 CMTP-NETCLS         PIC X(1).
    03 CMTP-NETCHG        PIC X(1).
    03 CMTP-NETACK         PIC X(1).
    03 CMTP-NETVCHK        PIC X(1).
    03 CMTP-NETRETN        PIC X(2).
    03 CMTP-NETEDIO        PIC X(1).
    03 CMTP-NETEDIP        PIC X(1).
    03 CMTP-STGFRMTO        PIC X(1).
    03 CMTP-MACHTYPE       PIC X(1).
    03 CMTP-STGFRMT        PIC X(1).
    03 CMTP-EOTID          PIC X(1).
    03 CMTP-LOGENV         PIC X(1).
    03 CMTP-FNGRPENV       PIC X(1).
    03 CMTP-SEDELIM        PIC X(1).
    03 CMTP-DEDELIM        PIC X(1).
    03 CMTP-SGDELIM        PIC X(1).
    03 CMTP-SGSEP          PIC X(1).
    03 CMTP-DECNOT         PIC X(1).
    03 CMTP-RLSCHAR        PIC X(1).
    03 CMTP-TPICTLNO       PIC X(9).
    03 CMTP-TPGCTLNO       PIC X(9).
    03 CMTP-TPTCTLNO       PIC X(9).
    03 CMTP-COMMENT1       PIC X(40).
    03 CMTP-COMMENT2       PIC X(40).
    03 CMTP-NETCMDS        PIC X(8).
    03 CMTP-TPDATA LINE    PIC X(32).

```

	03	CMTP-TIMEOUT	PIC X(4).
	03	CMTP-SEGMENTED	PIC X(1).
	03	CMTP-SUFFIX	PIC X(2).
	03	CMTP-TPENVUSUF	PIC X(2).
	03	CMTP-TPGENRCV	PIC X(1).
	03	CMTP-TPCMPRES	PIC X(1).
	03	CMTP-TPRSRV1	PIC X(8).
	03	CMTP-TPSUPAD3	PIC X(40).
	03	CMTP-TPSUPCTY	PIC X(30).
	03	CMTP-TPSUPST	PIC X(2).
	03	CMTP-TPSUPPST	PIC X(15).
	03	CMTP-TPSUPCON	PIC X(30).
	03	CMTP-TPSUPFAX	PIC X(25).
	03	CMTP-TPSUPU3	PIC X(40).
	03	CMTP-TPSUPU4	PIC X(40).
	03	CMTP-TPSUPU5	PIC X(40).
	03	CMTP-TPSUPU6	PIC X(40).
	03	CMTP-TPSUPU7	PIC X(40).
	03	CMTP-TPSUPU8	PIC X(40).
	03	CMTP-TPSUPU9	PIC X(40).
	03	CMTP-TPSUPU10	PIC X(40).
	03	CMTP-PRIORITY	PIC X(1).
	03	CMTP-TPRSRV2	PIC X(3).
	03	CMTP-DESCRIPT	PIC X(30).
	03	CMTP-LOGLOCK	PIC X(1).
	03	CMTP-LASTUID	PIC X(17).
	03	CMTP-LASTUDT	PIC S9(9) COMP-4.
	03	CMTP-TPTYPE	PIC X(1).
	03	CMTP-TPRSRV3	PIC X(467).

COBOL REQDB

```

*****
*                               DataInterchange/MVS                               *
*                               *                                                 *
* Release Level: Version 3, Release 1, Modification Level 0                     *
*                               *                                                 *
* Name: Requestor Profile Data Block (REQDB)                                    *
*                               *                                                 *
* Length: 264 bytes                                                         *
*                               *                                                 *
*****
01 CMRQ-DATA.
    03 CMRQ-BLKLEN                PIC S9(4) COMP-4.
    03 CMRQ-RESERV1              PIC S9(4) COMP-4.
    03 CMRQ-BLKNME               PIC X(8).
    03 CMRQ-REQID                PIC X(16).
    03 CMRQ-NETID                PIC X(8).
    03 CMRQ-ACCTNO               PIC X(32).
    03 CMRQ-USERID               PIC X(32).
    03 CMRQ-PASSWD               PIC X(16).
    03 CMRQ-MSGUCL               PIC X(8).
    03 CMRQ-INDDNAME             PIC X(8).
    03 CMRQ-NETCLS               PIC X(1).
    03 CMRQ-NETCHG               PIC X(1).
    03 CMRQ-NETACK               PIC X(1).
    03 CMRQ-NETVCHK              PIC X(1).

```



```

|      03 CMRQ-NETRETN          PIC X(3).
|      03 CMRQ-NETEDIO         PIC X(1).
|      03 CMRQ-NETEDIP         PIC X(1).
|      03 CMRQ-STGFRMTO        PIC X(1).
|      03 CMRQ-STGFRMT         PIC X(1).
|      03 CMRQ-NETCMDMBR       PIC X(8).
|      03 CMRQ-TIMEOUT         PIC X(4).
|      03 CMRQ-NTACKPGM        PIC X(8).
|      03 CMRQ-ALTNETPHONE     PIC X(32).
|      03 CMRQ-COMPRESS        PIC X(1).
|      03 CMRQ-PRIORITY        PIC X(1).
|      03 CMRQ-FILLER          PIC X(15).
|      03 CMRQ-DESCRIPT        PIC X(30).
|      03 CMRQ-LOGLOCK         PIC X(1).
|      03 CMRQ-LASTUID         PIC X(17).
|      03 CMRQ-LASTUDT         PIC S9(9) COMP-4.

```

COBOL DATBLK

```

| *****
| *                               DataInterchange/MVS                               *
| *                                                                                   *
| * Release Level: Version 3, Release 1, Modification Level 0                       *
| *                                                                                   *
| * Name: Translator Data Block (DATBLK)                                           *
| *                                                                                   *
| * Length: Variable                                                                *
| *                                                                                   *
| *****
| 01 DATA-BLK.
|      03 DB-BLKLEN            PIC S9(4) COMP-4.
|      03 DB-RESERV1          PIC S9(4) COMP-4.
|      03 DB-BLKNAME          PIC X(8).
|      03 DB-DATLEN           PIC S9(4) COMP-4.
|      03 DB-DATA             PIC X(1).

```

COBOL HUGEBLK

```

| *****
| *                               DataInterchange/MVS                               *
| *                                                                                   *
| * Release Level: Version 3, Release 1, Modification Level 0                       *
| *                                                                                   *
| * Name: Translator Huge Block (HUGEBLK)                                         *
| *                                                                                   *
| * Length: Variable (greater than 32 K)                                          *
| *                                                                                   *
| *****
| 01 HUGE-BLK.
|      03 HB-BLKLEN            PIC S9(9) COMP-4.
|      03 HB-NEXTBUF           PIC S9(9) COMP-4.
|      03 HB-PREVBUFF          PIC S9(9) COMP-4.
|      03 HB-DATLEN           PIC S9(9) COMP-4.
|      03 HB-DATA             PIC X(1).

```

COBOL SPDB

```
*****
*                               DataInterchange/MVS                               *
*                               *                                                 *
* Release Level: Version 3, Release 1, Modification Level 0                      *
*                               *                                                 *
* Name: Security Profile Data Block (SPDB)                                     *
*                               *                                                 *
* Length: 156 bytes                                                         *
*                               *                                                 *
*****
01 CMSP-DATA.
    03 CMSP-BLKLEN                PIC S9(4) COMP-4.
    03 CMSP-RESERV1              PIC S9(4) COMP-4.
    03 CMSP-BLKNAME              PIC X(8).
    03 CMSP-SECID                PIC X(8).
    03 CMSP-ORIGNAME             PIC X(16).
    03 CMSP-RECPNAME             PIC X(16).
    03 CMSP-AUTHTYPE             PIC X(1).
    03 CMSP-AUTHCODE             PIC X(1).
    03 CMSP-ENCRTYPE             PIC X(1).
    03 CMSP-FLTRTYPE            PIC X(1).
    03 CMSP-ENCRPROG             PIC X(8).
    03 CMSP-AUTHPROG             PIC X(8).
    03 CMSP-COMPPROG             PIC X(8).
    03 CMSP-FLTRPROG             PIC X(8).
    03 CMSP-BUFSIZE              PIC X(5).
    03 CMSP-FILLER               PIC X(11).
    03 CMSP-DESCRIPT             PIC X(30).
    03 CMSP-LOGLOCK              PIC X(1).
    03 CMSP-LASTUID             PIC X(17).
    03 CMSP-LASTUDT             PIC S9(9) COMP-4.
```

COBOL Utility Control Information Block

```
*****
*                               DataInterchange/MVS                               *
*                               *                                                 *
* Release Level: Version 3, Release 1, Modification Level 0                      *
*                               *                                                 *
* Name: Utility Control Block (FFUS)                                           *
*                               *                                                 *
* Length: 248 bytes                                                         *
*                               *                                                 *
*****
01 FFUS-DATA.
    03 FFUS-SYNCVAL              PIC S9(9) COMP-4.
    03 FFUS-CMDP                 PIC S9(9) COMP-4.
    03 FFUS-CMDLEN              PIC S9(9) COMP-4.
    03 FFUS-CMDNAME             PIC X(8).
    03 FFUS-CMDTYPE             PIC X(2).
    03 FFUS-DELIMITER           PIC X(1).
    03 FFUS-PRTNAME             PIC X(8).
    03 FFUS-PRTTYPE             PIC X(2).
    03 FFUS-RPTNAME             PIC X(8).
    03 FFUS-RPTTYPE             PIC X(2).
    03 FFUS-EXCPNAME            PIC X(8).
```

	03 FFUS-EXCPTYPE	PIC X(2).
	03 FFUS-TRAKNAME	PIC X(8).
	03 FFUS-TRAKTYPE	PIC X(2).
	03 FFUS-QRYNAME	PIC X(8).
	03 FFUS-QRYTYPE	PIC X(2).
	03 FFUS-APPLID	PIC X(8).
	03 FFUS-LANGID	PIC X(6).
	03 FFUS-RESPID	PIC X(8).
	03 FFUS-RESPTYP	PIC X(2).
	03 FFUS-RTERMID	PIC X(4).
	03 FFUS-USRFLD	PIC X(16).
	03 FFUS-SYSID	PIC X(8).
	03 FFUS-RESPFLAG	PIC X(1).
	03 FFUS-FILETYP	PIC X(2).
	03 FFUS-ECBP	PIC S9(9) COMP-4.
	03 FFUS-CCBRC	PIC S9(9) COMP-4.
	03 FFUS-CCBERC	PIC S9(9) COMP-4.
	03 FFUS-FILEID	PIC X(8).
	03 FFUS-APPFILE	PIC X(8).
	03 FFUS-ABNDCODE	PIC X(4).
	03 FFUS-THANDLE	PIC X(20).
	03 FFUS-FFIEHDR	PIC S9(9) COMP-4.
	03 FFUS-FARC	PIC S9(9) COMP-4.
	03 FFUS-FAERC	PIC S9(9) COMP-4.
	03 FFUS-FABUILT	PIC X(1).
	03 FFUS-FFMTFLG	PIC X(1).
	03 FFUS-USERSYNC	PIC X(1).
	03 FFUS-RES1	PIC X(1).
	03 FFUS-USERCOND	PIC S9(9) COMP-4.
	03 FFUS-RES2	PIC X(23).
	03 FFUS-RESPACTV	PIC X(1).
	03 FFUS-WORKNAME	PIC X(8).
	03 FFUS-BATFLG	PIC X(1).
	03 FFUS-NOEXCP	PIC X(1).
	03 FFUS-INVPARM	PIC X(1).
	03 FFUS-LOGACTV	PIC X(1).
	03 FFUS-FFUSADDR	PIC S9(9) COMP-4.
	03 FFUS-CCBP	PIC S9(9) COMP-4.
	03 FFUS-FFMTCBP	PIC S9(9) COMP-4.

Include Files for PL/I

PL/I CCB

```

| /*****
| /*                               DataInterchange/MVS                               */
| /*                               */
| /* Release Level: Version 3, Release 1, Modification Level 0                      */
| /*                               */
| /* Name: Common Control Block (CCB)                                             */
| /*                               */
| /* Length: 608 bytes                                                            */
| /*                               */
| *****/
| DCL 1 CCB,
|       3 ZCCBLL          FIXED BIN(15), /* CCB block length          */

```

```

|      3 ZCCBID      FIXED BIN(15), /* Reserved for future use */
|      3 ZCCBEYE     CHAR(8),      /* CCB block name */
|      3 ZCCBRC      FIXED BIN(31), /* Return code */
|      3 ZCCBERC     FIXED BIN(31), /* Extended return code */
|      3 ZCCBSID     CHAR(8),      /* System ID */
|      3 ZCCBUID     CHAR(8),      /* User ID */
|      3 ZCCBAID     CHAR(8),      /* Application ID */
|      3 ZCCBCID     CHAR(8),      /* Error module ID */
|      3 ZCCBXFID    FIXED BIN(15), /* Component function ID */
|      3 ZCCBLPID    CHAR(6),      /* Language profile ID */
|      3 ZCCBCPID    FIXED BIN(31), /* Code page ID */
|      3 ZCCBRSV     FIXED BIN(31), /* Reserved for future use */
|      3 ZCCBCCXP    POINTER,      /* CCB extension pointer */
|      3 ZCCBCABP    POINTER,      /* Common area block pointer */
|      3 ZCCBDBID    CHAR(4),      /* MVS DB2 subsystem ID */
|      3 ZCCBDBPL    CHAR(8),      /* MVS DB2 plan / AIX alias */
|      3 ZCCBDBUI    CHAR(8),      /* AIX DB2 user ID */
|      3 ZCCBDBPW    CHAR(18),     /* AIX DB2 password */
|      3 ZCCBRSV1    CHAR(26),     /* DI internal use only */
|      3 ZCCBRSV2(117) FIXED BIN(31); /* DI internal use only */

```

PL/I SNB

```

| /*****
| /*                      DataInterchange/MVS                      */
| /*                      */
| /* Release Level: Version 3, Release 1, Modification Level 0 */
| /*                      */
| /* Name: Service Name Block (SNB) */
| /*                      */
| /* Length: 32 bytes */
| /*                      */
| *****/
| DCL 1 SNB,
|      3 ZSNBLL      FIXED BIN(15), /* SNB block length */
|      3 ZSNBID      FIXED BIN(15), /* Reserved for future use */
|      3 ZSNBEYE     CHAR(8),      /* SNB block name */
|      3 ZSNBNAME     CHAR(8),      /* Service name */
|      3 ZSNBNDX     FIXED BIN(31), /* Service index */
|      3 ZSNBPC      FIXED BIN(15), /* Service parameter count */
|      3 ZSNBFLG0    CHAR(1),      /* First flag byte */
|      3 ZSNBFLG1    CHAR(1),      /* Second flag byte */
|      3 ZSNBFANC    POINTER;      /* First anchor pointer */

```

PL/I FCB

```

| /*****
| /*                      DataInterchange/MVS                      */
| /*                      */
| /* Release Level: Version 3, Release 1, Modification Level 0 */
| /*                      */
| /* Name: Function Code Block (FCB) */
| /*                      */
| /* Length: 4 bytes */
| /*                      */
| *****/

```

```

DCL 1 FCB,
      3 ZFCBLL      FIXED BIN(15), /* FCB block length      */
      3 ZFCBFUNC    FIXED BIN(15); /* Service function code */

```

PL/I CMCB

```

/*****
/*          DataInterchange/MVS          */
/*          */
/* Release Level: Version 3, Release 1, Modification Level 0 */
/*          */
/* Name: Communications Control Block (CMCB) */
/*          */
/* Length: 254 bytes */
/*          */
*****/
DCL 1 CMCB,
      3 BLKLEN      FIXED BIN(15), /* CMCB block length      */
      3 RESERV1     FIXED BIN(15), /* Reserved for future use */
      3 BLKNME      CHAR(8),       /* CMCB block name        */
      3 TPNICKNM     CHAR(16),      /* Trading partner nickname */
      3 NETID        CHAR(8),       /* Network ID              */
      3 NETOP        CHAR(8),       /* Network operation        */
      3 REQID        CHAR(16),      /* Requestor ID            */
      3 SEQNUM       CHAR(5),       /* Network sequence number */
      3 CONTRCV      CHAR(1),       /* Continuous receive flag */
      3 FTYPE        CHAR(2),       /* Send/receive file type  */
      3 FILERCVD     CHAR(1),       /* File received indicator */
      3 RESERV2      CHAR(5),       /* Reserved for future use */
      3 CLRFILE      CHAR(1),       /* Clear file after processing */
      3 DATAFMT     CHAR(1),       /* Data format             */
      3 ACCTYP       CHAR(1),       /* Account type            */
      3 DATATYP      CHAR(1),       /* Data type ddname/dataset */
      3 RECVTYP      CHAR(1),       /* Receive first or all files */
      3 DCIND        CHAR(1),       /* Delivery priority       */
      3 ACKIND       CHAR(1),       /* Acknowledgment request code */
      3 RESRECL      CHAR(1),       /* Resolve record length   */
      3 SCRIPT       CHAR(8),       /* Script name             */
      3 ENAME        CHAR(8),       /* Message user class      */
      3 MSGNAME      CHAR(8),       /* Message name            */
      3 FILENAME     CHAR(56),      /* Ddname or dataset name  */
      3 CANSD        CHAR(6),       /* Cancellation start date */
      3 CANST        CHAR(6),       /* Cancellation start time */
      3 CANED        CHAR(6),       /* Cancellation end date   */
      3 CANET        CHAR(6),       /* Cancellation end time   */
      3 TMZONE       CHAR(1),       /* Time zone for time interval */
      3 MODEM        CHAR(1),       /* Modem type              */
      3 NPSSCODE     CHAR(5),       /* Start session response code */
      3 NPESCODE     CHAR(5),       /* End session response code */
      3 NPERRCD      CHAR(5),       /* Network error code      */
      3 NPSEVER      CHAR(2),       /* Network error code severity */
      3 BLKTYPE      CHAR(1),       /* Data block type huge or VA */
      3 FQUEUED      CHAR(1),       /* Queue functions supported */
      3 FMSGGS       CHAR(1),       /* Free-form msgs supported */
      3 FFILE        CHAR(1),       /* Nonstandard files supported */
      3 FEDIX        CHAR(1),       /* EDI ISA/IEA envs supported */
      3 FEDIE        CHAR(1),       /* EDI UNB/UNZ envs supported */

```

```

|      3 FEDIU      CHAR(1),      /* EDI BG/EG envs supported */
|      3 FEDIG      CHAR(1),      /* EDI GS/GE envs supported */
|      3 FEDII      CHAR(1),      /* EDI ICS/ICE envs supported */
|      3 FEDIT      CHAR(1),      /* EDI STX/END envs supported */
|      3 FCANCEL    CHAR(1),      /* CANCEL function supported */
|      3 FCLASS     CHAR(1),      /* User msg classes supported */
|      3 FACK       CHAR(1),      /* Network acks supported */
|      3 FSYMSG     CHAR(1),      /* System messages supported */
|      3 FRCVBTP    CHAR(1),      /* Receive by trading partner */
|      3 FRESTART   CHAR(1),      /* Restart supported */
|      3 FNOUSERID  CHAR(1),      /* Acct number only supported */
|      3 FACCTSEP   CHAR(1),      /* Account/user ID separator */
|      3 DDCOLON    CHAR(3),      /* DD: string for IEBASE */
|      3 RESERV3    CHAR(2),      /* Reserved for future use */
|      3 ADMTYPE    CHAR(2),      /* Admin response file type */
|      3 UNIQID     CHAR(8),      /* Unique ID returned by Exp */
|      3 SAPUPDT    CHAR(1),      /* VANI SAP status tracking */
|      3 FSENTNET   CHAR(1),      /* Skip send-requested status */
|      3 RESERV4    CHAR(14);     /* Reserved for future use */

```

PL/I TRCB

```

| /*****
| /*                               */
| /*                               */
| /* Release Level: Version 3, Release 1, Modification Level 0 */
| /*                               */
| /* Name: Translator Control Block (TRCB) */
| /*                               */
| /* Length: 1536 bytes */
| /*                               */
| *****/
| DCL 1 TRCB,
|      3 BLKLEN     FIXED BIN(15), /* TRCB block length */
|      3 RSRVD1     FIXED BIN(15), /* Reserved for future use */
|      3 BLKNME     CHAR(8),       /* TRCB block name */
|      3 REQID      CHAR(16),      /* Requestor ID */
|      3 APPFILE    CHAR(8),       /* Application file written to */
|      3 ATFID      CHAR(16),      /* Application data format ID */
|      3 ATSID      CHAR(16),      /* Structure name returned */
|      3 EJECT      CHAR(1),       /* End of current transaction */
|      3 INTPID     CHAR(35),      /* Internal trading partner ID */
|      3 APPCTLNUM  CHAR(32),      /* Application control number */
|      3 TRNID      CHAR(16),      /* Standard transaction ID */
|      3 TEST       CHAR(1),       /* Usage indicator (I,P,T,U) */
|      3 IHCTL      CHAR(9),       /* Interchange control number */
|      3 GHCTL      CHAR(9),       /* Group control number */
|      3 THCTL      CHAR(9),       /* Transaction control number */
|      3 ENVTYP     CHAR(1),       /* Envelope type (E,I,T,U,X) */
|      3 BLKTYPE    CHAR(1),       /* TRIDB and TRODB format */
|      3 DUPTRAN    CHAR(1),       /* Duplicate transaction env */
|      3 ITPBREAK   CHAR(1),       /* ITP change causes new env */
|      3 REQSIZE    FIXED BIN(31), /* Output buffer required size */
|      3 XPANDED    CHAR(1),       /* Expanded TRCB block flag */
|      3 NEWENV     CHAR(1),       /* Transaction caused new env */
|      3 NEWGRP     CHAR(1),       /* Transaction caused new grp */
|      3 NEWTRN     CHAR(1),       /* Transaction caused new tran */

```

3	QSIZE	FIXED BIN(31),	/* Total queued envelope size */
3	ESIZE	FIXED BIN(31),	/* Current envelope size */
3	GRPNUM	FIXED BIN(31),	/* Number of groups in env */
3	TRNNUM	FIXED BIN(31),	/* Number of trans in env */
3	SEGNUM	FIXED BIN(31),	/* Number of segments in env */
3	TRNGRP	FIXED BIN(31),	/* Number of trans in group */
3	SEGTRN	FIXED BIN(31),	/* Number of segments in tran */
3	ERRNUM	FIXED BIN(31),	/* Number of translate errors */
3	QBT	CHAR(8),	/* QSIZE character equivalent */
3	IHXCTL	CHAR(14),	/* Interchange hdr control num */
3	ISYNTAXID	CHAR(4),	/* Syntax ID for E,T envelopes */
3	ISYNTAXVER	CHAR(1),	/* Syntax verification for E,T */
3	ISIDQUAL	CHAR(4),	/* Sender qualifier for E,I,X */
3	ISID	CHAR(35),	/* Interchange sender ID */
3	ISENDNAME	CHAR(14),	/* Sender name for U,T envs */
3	IREVROUT	CHAR(14),	/* Reverse routing addr E envs */
3	IRIDQUAL	CHAR(4),	/* Receiver qual for E,I,X */
3	IRID	CHAR(35),	/* Interchange receiver ID */
3	IRECVNAME	CHAR(14),	/* Receiver name for U,T envs */
3	IROUTEADDR	CHAR(14),	/* Routing address for E envs */
3	IDATE	CHAR(6),	/* Interchange date */
3	ITIME	CHAR(6),	/* Interchange time */
3	IVERREL	CHAR(5),	/* Interchange version/release */
3	IGT	CHAR(6),	/* GRPNUM character equivalent */
3	ITT	CHAR(6),	/* TRNNUM character equivalent */
3	IST	CHAR(10),	/* SEGNUM character equivalent */
3	IBT	CHAR(8),	/* ESIZE character equivalent */
3	ISPW	CHAR(14),	/* Interchange password */
3	IAPREF	CHAR(14),	/* Application reference */
3	ISTDID	CHAR(4),	/* Interchange standard ID */
3	RSRVD2	CHAR(1),	/* Reserved for future use */
3	IPRIOR	CHAR(1),	/* Interchange priority code */
3	ICOMMAGREE	CHAR(35),	/* Communication agreement */
3	GHXCTL	CHAR(14),	/* Group hdr control number */
3	GFGID	CHAR(6),	/* Group functional group ID */
3	GSIDQUAL	CHAR(4),	/* Group sender ID qualifier */
3	GSID	CHAR(35),	/* Group sender ID */
3	GRIDQUAL	CHAR(4),	/* Group receiver ID qualifier */
3	GRID	CHAR(35),	/* Group receiver ID */
3	GDATE	CHAR(6),	/* Group date */
3	GTIME	CHAR(6),	/* Group time */
3	GVER	CHAR(12),	/* Group version */
3	GREL	CHAR(12),	/* Group release */
3	GTT	CHAR(6),	/* TRNGRP character equivalent */
3	GAPW	CHAR(14),	/* Group password */
3	GRESAGENCY	CHAR(2),	/* Group responsible agency */
3	RSRVD3	CHAR(12),	/* Reserved for future use */
3	THXCTL	CHAR(14),	/* Transaction control number */
3	TTC	CHAR(6),	/* Current transaction ID */
3	TVER	CHAR(6),	/* Transaction version */
3	TREL	CHAR(6),	/* Transaction release */
3	TST	CHAR(10),	/* SEGTRN character equivalent */
3	LASTINENV	CHAR(1),	/* Last transaction in env */
3	XACFIELD	CHAR(35),	/* Application control field */
3	FABUILT	CHAR(1),	/* Functional ack built flag */
3	QGTNUM	FIXED BIN(31),	/* Queued group total */
3	QTTNUM	FIXED BIN(31),	/* Queued transaction total */

3	QSTNUM	FIXED BIN(31),	/* Queued segment total */
3	QGT	CHAR(6),	/* QGTNUM character equivalent */
3	QTT	CHAR(6),	/* QTTNUM character equivalent */
3	QST	CHAR(10),	/* QSTNUM character equivalent */
3	FASPM	CHAR(8),	/* FA standard profile member */
3	ENVCHK	CHAR(1),	/* Verify previous enveloping */
3	TRNSTAT	CHAR(1),	/* Transaction status */
3	APTYPE	CHAR(2),	/* Application file type */
3	TSKEY	CHAR(10),	/* Packed transaction handle */
3	TSKEYU	CHAR(20),	/* Unpacked transaction handle */
3	MAPKEY	CHAR(16),	/* Map used for translation */
3	BATCHID	CHAR(8),	/* Batch ID */
3	ENVLDATE	CHAR(8),	/* Earliest envelope date */
3	TRXLIFE	FIXED BIN(15),	/* Trans store tran life */
3	IMGLIFE	FIXED BIN(15),	/* Trans store image life */
3	HOLDFLAG	CHAR(1),	/* Trans store hold status */
3	BNDLFLAG	CHAR(1),	/* Trx starts a new bundle */
3	RAWDATA	CHAR(1),	/* Raw data indicator */
3	ENVLDELAY	CHAR(1),	/* Delay enveloping flag */
3	TRXACCEPT	CHAR(1),	/* TRX acceptable flag */
3	TRABORT	CHAR(1),	/* Translator terminated flag */
3	FILEID	CHAR(8),	/* Standard data file name */
3	DSNAME	CHAR(56),	/* Physical dataset name */
3	QNETID	CHAR(8),	/* Trading partner's NETID */
3	QPTTOPT	CHAR(1),	/* Point-to-point QNETID flag */
3	QSRPGM	CHAR(1),	/* Send/recv prog QNETID flag */
3	QDDNAME	CHAR(8),	/* Standard data file name */
3	FUNACKFLE	CHAR(8),	/* Functional ack file name */
3	QTPNICK	CHAR(16),	/* Trading partner nickname */
3	QRC	FIXED BIN(15),	/* Queueing return code */
3	QERC	FIXED BIN(15),	/* Queueing ext return code */
3	ERRCDES(10)	FIXED BIN(15),	/* First 10 errors encountered */
3	INMEMTRANS	FIXED BIN(15),	/* Number of trxs in storage */
3	NOCOMMIT	CHAR(1),	/* Database commits allowed */
3	SCOPE	CHAR(1),	/* Env vs trx level recovery */
3	TPNICK	CHAR(16),	/* Trading partner nickname */
3	CONCATENATE	CHAR(1),	/* Concatenate records flag */
3	ASSERTLVL	CHAR(1),	/* Assertion level */
3	RAWDATAOUT	CHAR(1),	/* Raw data output */
3	FIXEDTRX	CHAR(1),	/* Fixed-to-fixed transaction */
3	MRREQID	CHAR(16),	/* Management reporting REQID */
3	ERRFILTER	CHAR(80),	/* Error filter values */
3	FFILEID	CHAR(8),	/* Fixed-to-fixed file name */
3	VARTRACE	CHAR(1),	/* Variable trace wanted */
3	EXPTRACE	CHAR(1),	/* Expression trace wanted */
3	SSEGVAL	CHAR(1),	/* Service segment validation */
3	TRXFACODE	CHAR(1),	/* Transaction func ack code */
3	IUSEREXIT	CHAR(8),	/* User exit for VA processing */
3	IUSERAREA	FIXED BIN(31),	/* User exit user area */
3	IUSERACCESS	CHAR(1),	/* User exit access type */
3	IUSERTYPE	CHAR(2),	/* User exit program type */
3	MAPCHAIN	CHAR(1),	/* Mapchaining active flag */
3	FORCETEST	CHAR(1),	/* Use inbound test usage */
3	ENVPRBRK	CHAR(1),	/* Envelope profile break */
3	SUSBLKF	CHAR(1),	/* CICS SUSPEND block flag */
3	CLRERRS	CHAR(1),	/* Clear ERRCDES array flag */
3	FARC	FIXED BIN(31),	/* Func ack return code */


```

|      3 FAERC      FIXED BIN(31), /* Func ack ext return code */
|      3 SAPUPDT    CHAR(1),      /* SAP status tracking wanted */
|      3 SAPTRX     CHAR(1),      /* SAP transaction flag */
|      3 SAPCLIENT CHAR(3),      /* SAP client */
|      3 SAPDOCNUM  CHAR(16),     /* SAP document number */
|      3 SAPSEQ     CHAR(6),      /* SAP error sequence number */
|      3 ROUTCODE   CHAR(3),      /* Generic usage routing code */
|      3 FAEREQ     CHAR(1),      /* Func ack file required */
|      3 BOUNDARY   CHAR(1),      /* Incremental translation */
|      3 SUSBLKP    POINTER,      /* CICS SUSPEND block pointer */
|      3 CUSERDATA  CHAR(256),    /* C record user data area */
|      3 APPLTPID   CHAR(16),     /* Application trading partner */
|      3 EXTENDC    CHAR(1),      /* Extend the C record flag */
|      3 VAXFLAG    CHAR(1),      /* Pageable translation */
|      3 GDATES     CHAR(8),      /* Group envelope 8-byte date */
|      3 RECOVBAD   CHAR(1),      /* Recover from bad qdata */
|      3 RSRVD4     CHAR(61);     /* Reserved for future use */

```

PL/I NPDB

```

| /*****
| /*                               */
| /*                               */
| /* Release Level: Version 3, Release 1, Modification Level 0 */
| /*                               */
| /* Name: Network Profile Data Block (NPDB) */
| /*                               */
| /* Length: 300 bytes */
| /*                               */
| /*****
| DCL 1 NPDB,
|      3 BLKLEN     FIXED BIN(15), /* NPDB block length */
|      3 RESERV1    FIXED BIN(15), /* Reserved for future use */
|      3 BLKNME     CHAR(8),      /* NPDB block name */
|      3 NETID      CHAR(8),      /* Network identification */
|      3 NETNME     CHAR(30),     /* Network name */
|      3 COMROT     CHAR(8),      /* Communication routine name */
|      3 NETPGM     CHAR(8),      /* Network program name */
|      3 PGMPARM    CHAR(57),     /* Network program parameters */
|      3 CMDIN      CHAR(8),      /* Network command input file */
|      3 CMDLRECL   CHAR(4),      /* Network command rec length */
|      3 QDATA      CHAR(8),      /* Transaction data queue file */
|      3 DATLRECL   CHAR(4),      /* Transaction data rec length */
|      3 TMZONE     CHAR(5),      /* Time zone */
|      3 SYSTYP     CHAR(8),      /* System type */
|      3 SYSLVL     CHAR(4),      /* System level */
|      3 TXTHDR     CHAR(1),      /* Text header character */
|      3 CMDOUT     CHAR(8),      /* Network pgm cmd output file */
|      3 MSGROUT    CHAR(8),      /* Message handler program */
|      3 SEQNUM     CHAR(5),      /* Network sequence number */
|      3 NETACKFILE CHAR(8),      /* Net acknowledgment file */
|      3 NETPHONE   CHAR(32),     /* Dial connection phone */
|      3 SCRIPT     CHAR(8),      /* Script name */
|      3 FILLER     CHAR(14),     /* Reserved for future use */
|      3 DESCRIPT   CHAR(30),     /* Profile member description */

```

	3 LOGLOCK	CHAR(1),	/* Logical lock flag	*/
	3 LASTUID	CHAR(17),	/* Last update user ID	*/
	3 LASTUDT	FIXED BIN(31);	/* Last update date/time	*/

| PL/I TPPDB

```

| /*****
| /*                      DatInterchange/MVS                      */
| /*                      */
| /* Release Level: Version 3, Release 1, Modification Level 0    */
| /*                      */
| /* Name: Trading Partner Profile Data Block (TPPDB)              */
| /*                      */
| /* Length: 1532 bytes                                           */
| /*                      */
| *****/
| DCL 1 TPPDB,
|   3 BLKLEN          FIXED BIN(15), /* TPPDB block length      */
|   3 RESERV1         FIXED BIN(15), /* Reserved for future use  */
|   3 BLKNME          CHAR(8),        /* TPPDB block name        */
|   3 TPNICKNM        CHAR(16),       /* Trading partner nickname */
|   3 NETID           CHAR(8),        /* Network profile member   */
|   3 SYSQUAL         CHAR(1),        /* Inter-system qualifier   */
|   3 SYSID           CHAR(8),        /* Inter-system ID         */
|   3 ACCTNUM         CHAR(32),       /* Network account number   */
|   3 USERID          CHAR(32),       /* Network user ID         */
|   3 ENVLQUAL        CHAR(4),        /* Interchange qualifier    */
|   3 ENVLID          CHAR(35),       /* Interchange ID          */
|   3 CONAME          CHAR(40),       /* Company name            */
|   3 ADDR1           CHAR(40),       /* Company address line 1   */
|   3 ADDR2           CHAR(40),       /* Company address line 2   */
|   3 PHONE           CHAR(25),       /* Contact phone number     */
|   3 CONTACT         CHAR(30),       /* Contact name            */
|   3 PASSWORD        CHAR(14),       /* Password for sending     */
|   3 RCVPASS         CHAR(14),       /* Password for receiving   */
|   3 SECUID          CHAR(8),        /* Security profile member  */
|   3 NETCLS          CHAR(1),        /* Network message class    */
|   3 NETCHG          CHAR(1),        /* Network charge code      */
|   3 NETACK          CHAR(1),        /* Network acknowledgment code */
|   3 NETVCHK         CHAR(1),        /* Destination verification */
|   3 NETRETN         CHAR(3),        /* Retention period         */
|   3 NETEDIO         CHAR(1),        /* EDI processing option    */
|   3 NETEDIP         CHAR(1),        /* EDI processing override  */
|   3 STGFRMT         CHAR(1),        /* Storage format override  */
|   3 MACHTYPE        CHAR(1),        /* Machine type            */
|   3 STGFRMT         CHAR(1),        /* Storage format          */
|   3 EOTID           CHAR(1),        /* End of text delimiter    */
|   3 LOGENV          CHAR(1),        /* Log envelope data       */
|   3 FNGRPEN         CHAR(1),        /* Functional group wanted  */
|   3 SEDELIM         CHAR(1),        /* Sub element delimiter    */
|   3 DEDELIM         CHAR(1),        /* Data element delimiter   */
|   3 SGDELIM         CHAR(1),        /* Segment delimiter       */
|   3 SGSEP           CHAR(1),        /* Segment ID separator     */
|   3 DECNOT          CHAR(1),        /* Decimal notation        */
|   3 RLSCHAR         CHAR(1),        /* Release character        */
|   3 TPICTLNO        CHAR(9),        /* Interchange mask        */
|   3 TPGCTLNO        CHAR(9),        /* Group mask              */

```

	3	TPCTLNO	CHAR(9),	/* Transaction mask	*/
	3	COMMENT1	CHAR(40),	/* Comment line 1	*/
	3	COMMENT2	CHAR(40),	/* Comment line 2	*/
	3	NETCMDS	CHAR(8),	/* Network command file member	*/
	3	TPDATA LINE	CHAR(32),	/* Direct dial phone number	*/
	3	TIMEOUT	CHAR(4),	/* Communication line timeout	*/
	3	SEGMENTED	CHAR(1),	/* Segmented output	*/
	3	SUFFIX	CHAR(2),	/* File suffix	*/
	3	TPENV SUF	CHAR(2),	/* Envelope profile suffix	*/
	3	TPGENRCV	CHAR(1),	/* Allow generic recv usages	*/
	3	TPCMPRES	CHAR(1),	/* Compression flag	*/
	3	TPRSRV1	CHAR(8),	/* Reserved for future use	*/
	3	TPSUPAD3	CHAR(40),	/* Company address line 3	*/
	3	TPSUPCTY	CHAR(30),	/* City name	*/
	3	TPSUPST	CHAR(2),	/* State code	*/
	3	TPSUPPST	CHAR(15),	/* Postal code	*/
	3	TPSUPCON	CHAR(30),	/* Country	*/
	3	TPSUPFAX	CHAR(25),	/* Contact fax number	*/
	3	TPSUPU3	CHAR(40),	/* Comment line 3	*/
	3	TPSUPU4	CHAR(40),	/* Comment line 4	*/
	3	TPSUPU5	CHAR(40),	/* Comment line 5	*/
	3	TPSUPU6	CHAR(40),	/* Comment line 6	*/
	3	TPSUPU7	CHAR(40),	/* Comment line 7	*/
	3	TPSUPU8	CHAR(40),	/* Comment line 8	*/
	3	TPSUPU9	CHAR(40),	/* Comment line 9	*/
	3	TPSUPU10	CHAR(40),	/* Comment line 10	*/
	3	PRIORITY	CHAR(1),	/* Delivery priority	*/
	3	TPRSRV2	CHAR(3),	/* Reserved for future use	*/
	3	DESCRIPT	CHAR(30),	/* Profile member description	*/
	3	LOGLOCK	CHAR(1),	/* Logical lock flag	*/
	3	LASTUID	CHAR(17),	/* Last update user ID	*/
	3	LASTUDT	FIXED BIN(31),	/* Last update date/time	*/
	3	TPTYPE	CHAR(1),	/* Trading partner type	*/
	3	TPRSRV3	CHAR(467);	/* Reserved for future use	*/

PLI REQDB

```

| /*****
| /*                               */
| /*                               */
| /* Release Level: Version 3, Release 1, Modification Level 0 */
| /*                               */
| /* Name: Requestor Profile Data Block (REQDB) */
| /*                               */
| /* Length: 264 bytes */
| /*                               */
| *****/
| DCL 1 REQDB,
|     3 BLKLEN      FIXED BIN(15), /* REQDB block length */
|     3 RESERV1     FIXED BIN(15), /* Reserved for future use */
|     3 BLKNME      CHAR(8),       /* REQDB block name */
|     3 REQID       CHAR(16),      /* Requestor ID */
|     3 NETID       CHAR(8),       /* Network profile member */
|     3 ACCTNO      CHAR(32),      /* Network account number */
|     3 USERID      CHAR(32),      /* Network user ID */
|     3 PASSWD      CHAR(16),      /* Network password */
|     3 MSGUCL      CHAR(8),       /* Network message user class */

```

```

|      3 INDDNAME      CHAR(8),      /* Receive file name      */
|      3 NETCLS        CHAR(1),      /* Network message class  */
|      3 NETCHG        CHAR(1),      /* Network charge code    */
|      3 NETACK        CHAR(1),      /* Network acknowledgment */
|      3 NETVCHK       CHAR(1),      /* Destination verification */
|      3 NETRETN       CHAR(3),      /* Retention period       */
|      3 NETEDIO       CHAR(1),      /* EDI processing option  */
|      3 NETEDIP       CHAR(1),      /* EDI processing override */
|      3 STGFRMTO      CHAR(1),      /* Storage format override */
|      3 STGFRMT       CHAR(1),      /* Storage format         */
|      3 NETCMDMBR     CHAR(8),      /* Network command file member */
|      3 TIMEOUT      CHAR(4),      /* Communication line timeout */
|      3 NTACKPGM      CHAR(8),      /* Remote message handler */
|      3 ALTNETPHONE   CHAR(32),     /* Alternate dial phone number */
|      3 COMPRESS      CHAR(1),      /* Compression flag       */
|      3 PRIORITY      CHAR(1),      /* Delivery priority      */
|      3 FILLER        CHAR(15),     /* Reserved for future use */
|      3 DESCRIPT      CHAR(30),     /* Profile member description */
|      3 LOGLOCK       CHAR(1),      /* Logical lock flag      */
|      3 LASTUID       CHAR(17),     /* Last update user ID    */
|      3 LASTUDT       FIXED BIN(31); /* Last update date/time  */

```

PL/I DATBLK

```

| /*****
| /*                      DataInterchange/MVS                      */
| /*                      */
| /* Release Level: Version 3, Release 1, Modification Level 0      */
| /*                      */
| /* Name: Translator Data Block (DATBLK)                          */
| /*                      */
| /* Length: Variable                                             */
| /*                      */
| *****/
| DCL 1 DATBLK,
|      3 BLKLEN        FIXED BIN(15), /* DATBLK block length    */
|      3 RESERV1       FIXED BIN(15), /* Reserved for future use */
|      3 BLKNME        CHAR(8),      /* DATBLK block name      */
|      3 DATLEN        FIXED BIN(15), /* Length of data in block */
|      3 DATA         CHAR(1);      /* First byte of data     */

```

PL/I HUGEBLK

```

| /*****
| /*                      DataInterchange/MVS                      */
| /*                      */
| /* Release Level: Version 3, Release 1, Modification Level 0      */
| /*                      */
| /* Name: Translator Huge Block (HUGEBLK)                        */
| /*                      */
| /* Length: Variable (greater than 32 K)                         */
| /*                      */
| *****/
| DCL 1 HUGEBLK,
|      3 BLKLEN        FIXED BIN(31), /* HUGEBLK block length   */
|      3 NEXTBUF       POINTER,      /* Pointer to next buffer  */
|      3 PREVBUF       POINTER,      /* Pointer to previous buffer */

```

```
|      3 DATLEN      FIXED BIN(31), /* Length of data in block */
|      3 DATA       CHAR(1);      /* First byte of data */
```

PL/I SPDB

```
| /*****
| /*          DataInterchange/MVS          */
| /*          */
| /* Release Level: Version 3, Release 1, Modification Level 0 */
| /*          */
| /* Name: Security Profile Data Block (SPDB) */
| /*          */
| /* Length: 156 bytes */
| /*          */
| *****/
| DCL 1 SPDB,
|      3 BLKLEN      FIXED BIN(15), /* SPDB block length */
|      3 RESERV1     FIXED BIN(15), /* Reserved for future use */
|      3 BLKNAME     CHAR(8),       /* SPDB block name */
|      3 SECID       CHAR(8),       /* Security ID */
|      3 ORIGINAME   CHAR(16),      /* Security originator */
|      3 RECPNAME    CHAR(16),      /* Security recipient */
|      3 AUTHTYPE    CHAR(1),       /* Authorization type */
|      3 AUTHCODE    CHAR(1),       /* Authorization code */
|      3 ENCRTYPE    CHAR(1),       /* Encryption type */
|      3 FLTRTYPE    CHAR(1),       /* Filtering type */
|      3 ENCRPROG    CHAR(8),       /* Encryption program */
|      3 AUTHPROG    CHAR(8),       /* Authentication program */
|      3 COMPPROG    CHAR(8),       /* Compression program */
|      3 FLTRPROG    CHAR(8),       /* Filtering program */
|      3 BUFSIZE     CHAR(5),       /* Buffer size */
|      3 FILLER      CHAR(11),      /* Reserved for future use */
|      3 DESCRIPT    CHAR(30),      /* Profile member description */
|      3 LOGLOCK     CHAR(1),       /* Logical lock flag */
|      3 LASTUID     CHAR(17),      /* Last update user ID */
|      3 LASTUDT     FIXED BIN(31); /* Last update date/time */
```

PL/I Utility Control Information Block

```
| /*****
| /*          DataInterchange/MVS          */
| /*          */
| /* Release Level: Version 3, Release 1, Modification Level 0 */
| /*          */
| /* Name: Utility Control Block (FFUS) */
| /*          */
| /* Length: 248 bytes */
| /*          */
| *****/
| DCL 1 FFUS,
|      3 SYNCVAL     FIXED BIN(31), /* Syncpoint interval */
|      3 CMDP        POINTER,       /* Command string address */
|      3 CMDLEN      FIXED BIN(31), /* Command string length */
|      3 CMDNAME     CHAR(8),       /* Command file name */
|      3 CMDTYPE     CHAR(2),       /* Command file type */
|      3 DELIMITER   CHAR(1),       /* Command value delimiter */
|      3 PRTNAME     CHAR(8),       /* Print file name */
```

	3	PRTTYPE	CHAR(2),	/* Print file type	*/
	3	RPTNAME	CHAR(8),	/* Report file name	*/
	3	RPTTYPE	CHAR(2),	/* Report file type	*/
	3	EXCPNAME	CHAR(8),	/* Exception file name	*/
	3	EXCPTYPE	CHAR(2),	/* Exception file type	*/
	3	TRAKNAME	CHAR(8),	/* Tracking file name	*/
	3	TRAKTYPE	CHAR(2),	/* Tracking file type	*/
	3	QRYNAME	CHAR(8),	/* Query file name	*/
	3	QRYTYPE	CHAR(2),	/* Query file type	*/
	3	APPLID	CHAR(8),	/* Application ID override	*/
	3	LANGID	CHAR(6),	/* Language ID override	*/
	3	RESPID	CHAR(8),	/* Response pgm/trx name	*/
	3	RESPTYP	CHAR(2),	/* Response type	*/
	3	RTERMID	CHAR(4),	/* Response terminal ID	*/
	3	USRFLD	CHAR(16),	/* User field to pass data	*/
	3	SYSID	CHAR(8),	/* CICS system ID override	*/
	3	RESPFLAG	CHAR(1),	/* Response invocation reason	*/
	3	FILETYP	CHAR(2),	/* Network ack file type	*/
	3	ECBP	POINTER,	/* ECB address	*/
	3	CCBRC	FIXED BIN(31),	/* Return code	*/
	3	CCBERC	FIXED BIN(31),	/* Extended return code	*/
	3	FILEID	CHAR(8),	/* Envelope TSQ (or net acks)	*/
	3	APPFILE	CHAR(8),	/* Translated data TSQ	*/
	3	ABNDPCODE	CHAR(4),	/* CICS abend code	*/
	3	THANDLE	CHAR(20),	/* Translation handle	*/
	3	FFIEHDR	POINTER,	/* IE long header address	*/
	3	FARC	FIXED BIN(31),	/* Func ack return code	*/
	3	FAERC	FIXED BIN(31),	/* Func ack extend return code	*/
	3	FABUILT	CHAR(1),	/* Func ack built flag	*/
	3	FFMTFLG	CHAR(1),	/* Multi-TSQ indicator	*/
	3	USERSYNC	CHAR(1),	/* User syncpoint flag	*/
	3	RES1	CHAR(1),	/* Reserved for future use	*/
	3	USERCOND	POINTER,	/* User condition code pointer	*/
	3	RES2	CHAR(23),	/* Reserved for future use	*/
	3	RESPACTV	CHAR(1),	/* Response pgm active flag	*/
	3	WORKNAME	CHAR(8),	/* Workfile name	*/
	3	BATFLG	CHAR(1),	/* Batch or UTILSRV API flag	*/
	3	NOEXCP	CHAR(1),	/* No exception file	*/
	3	INVPARM	CHAR(1),	/* Network pgm invalid parm	*/
	3	LOGACTV	CHAR(1),	/* Logging CE0050 active	*/
	3	FFUSADDR	POINTER,	/* Pointer to this FFUS block	*/
	3	CCBP	POINTER,	/* Pointer to the CCB	*/
	3	FFMTCBP	POINTER;	/* Multi-TSQ control block ptr	*/

Include Files for C

C CCB

```
/*
 * DataInterchange/MVS
 */
/*
 * Release Level: Version 3, Release 1, Modification Level 0
 */
/*
 * Name: Common Control Block (CCB)
 */
/*
 * Length: 608 bytes
 */
/*****
typedef struct CCB ccb;          /* Provide "ccb" data type */
struct CCB {
    short zccbll;               /* CCB block length */
    short zccbid;               /* Reserved for future use */
    char zccbeye[8];            /* CCB block name */
    long zccbrc;                /* Return code */
    long zccberc;               /* Extended return code */
    char zccbsid[8];            /* System ID */
    char zccbuid[8];            /* User ID */
    char zccbaid[8];            /* Application ID */
    char zccbcid[8];            /* Error module ID */
    short zccbxfid;             /* Component function ID */
    char zccblpid[6];           /* Language profile ID */
    long zccbcpid;              /* Code page ID */
    long zccbrsv;               /* Reserved for future use */
    void *zccbccxp;              /* CCB extension pointer */
    void *zccbcabp;              /* Common area block pointer */
    char zccbdbid[4];           /* MVS DB2 subsystem ID */
    char zccbdbpl[8];           /* MVS DB2 plan / AIX alias */
    char zccbdbui[8];           /* AIX DB2 user ID */
    char zccbdbpw[18];          /* AIX DB2 password */
    char zccbrsv1[26];          /* DI internal use only */
    long zccbrsv2[117];         /* DI internal use only */
};
```

C SNB

```
/*
 * DataInterchange/MVS
 */
/*
 * Release Level: Version 3, Release 1, Modification Level 0
 */
/*
 * Name: Service Name Block (SNB)
 */
/*
 * Length: 32 bytes
 */
/*****
typedef struct SNB snb;          /* Provide "snb" data type */
struct SNB {
    short zsnbll;               /* SNB block length */
    short zsnbid;               /* Reserved for future use */
    char zsnbeye[8];            /* SNB block name */
    char zsnbname[8];           /* Service name */
};
```

```

|     long  zsnbndx;           /* Service index          */
|     short zsnbpc;           /* Service parameter count */
|     char  zsnbflg0;         /* First flag byte         */
|     char  zsnbflg1;         /* Second flag byte        */
|     void  *zsnbfanc;        /* First anchor pointer    */
| };

```

C FCB

```

| /*****
| /*                      DataInterchange/MVS                      */
| /*                      */
| /* Release Level: Version 3, Release 1, Modification Level 0      */
| /*                      */
| /* Name: Function Code Block (FCB)                                */
| /*                      */
| /* Length: 4 bytes                                              */
| /*                      */
| /*****/
| typedef struct FCB fcb;           /* Provide "fcb" data type */
| struct FCB {
|     short zfcbl1;               /* FCB block length         */
|     short zfcbfunc;             /* Service function code    */
| };

```

C CMCB

```

| /*****
| /*                      DataInterchange/MVS                      */
| /*                      */
| /* Release Level: Version 3, Release 1, Modification Level 0      */
| /*                      */
| /* Name: Communications Control Block (CMCB)                      */
| /*                      */
| /* Length: 254 bytes                                            */
| /*                      */
| /*****/
| typedef struct CMCB CMcb;        /* Provide "CMcb" data type */
| struct CMCB {
|     short blklen;               /* CMCB block length        */
|     short reserv1;              /* Reserved for future use   */
|     char  blkname[8];           /* CMCB block name          */
|     char  tpnicknm[16];         /* Trading partner nickname  */
|     char  netid[8];             /* Network ID                */
|     char  netop[8];             /* Network operation         */
|     char  reqid[16];            /* Requestor ID              */
|     char  seqnum[5];            /* Network sequence number   */
|     char  contrcv;              /* Continuous receive file type*/
|     char  ftype[2];             /* Send/receive file type    */
|     char  filercvd;             /* File received indicator   */
|     char  reserv2[5];           /* Reserved for future use   */
|     char  clrfile[1];           /* Clear file after processing */
|     char  datafmt[1];           /* Data format               */
|     char  acctyp[1];            /* Account type              */
|     char  datatyp[1];           /* Data type ddname/dataset  */
|     char  recvtyp[1];           /* Receive first or all files */
|     char  dcind[1];             /* Delivery priority         */

```



```

| char ackind[1]; /* Acknowledgment request code */
| char resrec[1]; /* Resolve record length */
| char script[8]; /* Script name */
| char ename[8]; /* Message user class */
| char msgname[8]; /* Message name */
| char filename[56]; /* Ddname or dataset name */
| char cansd[6]; /* Cancellation start date */
| char canst[6]; /* Cancellation start time */
| char caned[6]; /* Cancellation end date */
| char canet[6]; /* Cancellation end time */
| char tmzone[1]; /* Time zone for time interval */
| char modem[1]; /* Modem type */
| char npsscde[5]; /* Start session response code */
| char npescde[5]; /* End session response code */
| char nperrcd[5]; /* Network error code */
| char npsever[2]; /* Network error code severity */
| char blktype; /* Data block type huge or VA */
| char fqueued; /* Queue functions supported */
| char fmsgs; /* Free-form msgs supported */
| char ffile; /* Nonstandard files supported */
| char fedix; /* EDI ISA/IEA envs supported */
| char fedie; /* EDI UNB/UNZ envs supported */
| char fediu; /* EDI BG/EG envs supported */
| char fedig; /* EDI GS/GE envs supported */
| char fedii; /* EDI ICS/ICE envs supported */
| char fedit; /* EDI STX/END envs supported */
| char fcancel; /* CANCEL function supported */
| char fclass; /* User msg classes supported */
| char fack; /* Network acks supported */
| char fsysmsg; /* System messages supported */
| char frcvbtp; /* Receive by trading partner */
| char frestart; /* Restart supported */
| char fnouserid; /* Acct number only supported */
| char facctsep; /* Account/user ID separator */
| char ddcolon[3]; /* DD: string for IEBASE */
| char reserv3[2]; /* Reserved for future use */
| char admtype[2]; /* Admin response file type */
| char uniqid[8]; /* Unique ID returned by Exp */
| char sapupdt; /* VANI SAP status tracking */
| char fsentnet; /* Skip send-requested status */
| char reserv4[14]; /* Reserved for future use */
| };

```

C TRCB

```

| /*****
| /* DataInterchange/MVS */
| /*
| /* Release Level: Version 3, Release 1, Modification Level 0
| /*
| /* Name: Translator Control Block (TRCB)
| /*
| /* Length: 1536 bytes
| /*
| *****/
| typedef struct TRCB TRcb; /* Provide "TRcb" data type */
| struct TRCB {

```

short blklen;	/* TRCB block length */
short rsrvd1;	/* Reserved for future use */
char blkname[8];	/* TRCB block name */
char reqid[16];	/* Requestor ID */
char appfile[8];	/* Application file written to */
char atfid[16];	/* Application data format ID */
char atsid[16];	/* Structure name returned */
char eject;	/* End of current transaction */
char intpid[35];	/* Internal trading partner ID */
char appctlnum[32];	/* Application control number */
char trnid[16];	/* Standard transaction ID */
char test;	/* Usage indicator (I,P,T,U) */
char ihctl[9];	/* Interchange control number */
char ghctl[9];	/* Group control numbersav */
char thctl[9];	/* Transaction control number */
char envtype;	/* Envelope type (E,I,T,U,X) */
char blktype;	/* TRIDB and TRODB format */
char duptran;	/* Duplicate transaction flag */
char itpbreak;	/* ITP change causes new env */
long reqsize;	/* Output buffer required size */
char xpanded;	/* Expanded TRCB block flag */
char newenv;	/* Transaction caused new env */
char newgrp;	/* Transaction caused new grp */
char newtrn;	/* Transaction caused new tran */
long qsize;	/* Total queued envelope size */
long esize;	/* Current envelope size */
long grpnum;	/* Number of groups in env */
long trnnum;	/* Number of trans in env */
long segnum;	/* Number of segments in env */
long trngrp;	/* Number of trans in group */
long segtrn;	/* Number of segments in tran */
long errnum;	/* Number of translate errors */
char qbt[8];	/* QSIZE character equivalent */
char ihxctl[14];	/* Interchange hdr control num */
char isyntaxid[4];	/* Syntax ID for E,T envelopes */
char isyntaxver[1];	/* Syntax verification for E,T */
char isidqual[4];	/* Sender qualifier for E,I,X */
char isid[35];	/* Interchange sender ID */
char isendname[14];	/* Sender name for U,T envs */
char irevrout[14];	/* Reverse routing addr E envs */
char iridqual[4];	/* Receiver qual for E,I,X */
char irid[35];	/* Interchange receiver ID */
char irecvname[14];	/* Receiver name for U,T envs */
char irouteaddr[14];	/* Routing address for E envs */
char idate[6];	/* Interchange date */
char itime[6];	/* Interchange time */
char iverrel[5];	/* Interchange version/release */
char igt[6];	/* GRPNUM character equivalent */
char itt[6];	/* TRNNUM character equivalent */
char ist[10];	/* SEGNUM character equivalent */
char ibt[8];	/* ESIZE character equivalent */
char ispw[14];	/* Interchange password */
char iapref[14];	/* Application reference */
char istdid[4];	/* Interchange standard ID */
char rsrvd2[1];	/* Reserved for future use */
char iprior[1];	/* Interchange priority code */
char icommagree[35];	/* Communication agreement */

	char	ghxctl[14];	/* Group hdr control number */
	char	gfgid[6];	/* Group functional group ID */
	char	gsidqual[4];	/* Group sender ID qualifier */
	char	gsid[35];	/* Group sender ID */
	char	gridqual[4];	/* Group receiver ID qualifier */
	char	grid[35];	/* Group receiver ID */
	char	gdate[6];	/* Group date */
	char	gtime[6];	/* Group time */
	char	gver[12];	/* Group version */
	char	grel[12];	/* Group release */
	char	gtt[6];	/* TRNGRP character equivalent */
	char	gapw[14];	/* Group password */
	char	grespagency[2];	/* Group responsible agency */
	char	rsrvd3[12];	/* Reserved for future use */
	char	thxctl[14];	/* Transaction control number */
	char	ttc[6];	/* Current transaction ID */
	char	tver[6];	/* Transaction version */
	char	trel[6];	/* Transaction release */
	char	tst[10];	/* SEGTRN character equivalent */
	char	lastinenv[1];	/* Last transaction in env */
	char	xacfield[35];	/* Application control field */
	char	fabuilt[1];	/* Functional ack built flag */
	long	qgtnum;	/* Queued group total */
	long	qttnum;	/* Queued transaction total */
	long	qstnum;	/* Queued segment total */
	char	qgt[6];	/* QGTNUM character equivalent */
	char	qtt[6];	/* QTTNUM character equivalent */
	char	qst[10];	/* QSTNUM character equivalent */
	char	fasp[8];	/* FA standard profile member */
	char	envchk;	/* Verify previous enveloping */
	char	trnstat;	/* Transaction status */
	char	aptype[2];	/* Application file type */
	char	tskey[10];	/* Packed transaction handle */
	char	tskeyu[20];	/* Unpacked transaction handle */
	char	mapkey[16];	/* Map used for translation */
	char	batchid[8];	/* Batch ID */
	char	envldate[8];	/* Earliest envelope date */
	short	trxlfe;	/* Trans store tran life */
	short	imglfe;	/* Trans store image life */
	char	holdflag[1];	/* Trans store hold status */
	char	bndlflag[1];	/* Trx starts a new bundle */
	char	rawdata[1];	/* Raw data indicator */
	char	envldelay[1];	/* Delay enveloping flag */
	char	trxaccept[1];	/* Trx acceptable flag */
	char	trabort[1];	/* Translator terminated flag */
	char	fileid[8];	/* Standard data file name */
	char	dsname[56];	/* Physical dataset name */
	char	qnetid[8];	/* Trading partner's NETID */
	char	qpttopt[1];	/* Point-to-point QNETID flag */
	char	qsrpgm[1];	/* Send/recv prog QNETID flag */
	char	qddname[8];	/* Standard data file name */
	char	funackfle[8];	/* Functional ack file name */
	char	qtpnick[16];	/* Trading partner nickname */
	short	qrc;	/* Queueing return code */
	short	qerc;	/* Queueing ext return code */
	short	errcdes[10];	/* First 10 errors encountered */
	short	inmemtrans;	/* Number of trxs in storage */

```

| char nocommit; /* Database commits allowed */
| char scope; /* Env vs trx level recovery */
| char tpnick[16]; /* Trading partner nickname */
| char concatenate; /* Concatenate records flag */
| char assertlvl; /* Assertion level */
| char rawdataout; /* Raw data output */
| char fixedtrx; /* Fixed-to-fixed transaction */
| char mrreqid[16]; /* Management reporting REQID */
| char errfilter[80]; /* Error filter values */
| char ffileid[8]; /* Fixed-to-fixed file name */
| char vartrace; /* Variable trace wanted */
| char exptrace; /* Expression trace wanted */
| char ssegval; /* Service segment validation */
| char trxfacode; /* Transaction func ack code */
| char iuserexit[8]; /* User exit for VA processing */
| long iuserarea; /* User exit user area */
| char iuseraccess; /* User exit access type */
| char iusertype[2]; /* User exit program type */
| char mapchain; /* Mapchaining active flag */
| char forcetest; /* Use inbound test usage */
| char envprbrk; /* Envelope profile break */
| char susblkf; /* CICS SUSPEND block flag */
| char clrerrs; /* Clear ERRCDES array flag */
| long farc; /* Func ack return code */
| long faerc; /* Func ack ext return code */
| char sapupdt; /* SAP status tracking wanted */
| char saptrx; /* SAP transaction flag */
| char sapclient[3]; /* SAP client */
| char sapdocnum[16]; /* SAP document number */
| char sapseq[6]; /* SAP error sequence number */
| char routcode[3]; /* Generic usage routing code */
| char faereq; /* Func ack file required */
| char boundary; /* Incremental translation */
| void *susblkp; /* CICS SUSPEND block pointer */
| char cuserdata[256]; /* C record user data area */
| char appltpid[16]; /* Application trading partner */
| char extendc; /* Extend the C record flag */
| char vaxflag; /* Pageable translation */
| char gdate[8]; /* Group envelope 8-byte date */
| char recovbad; /* Recover from bad qdata */
| char rsrvd4[61]; /* Reserved for future use */
| };

```

C NPDB

```

| /*****
| /* DataInterchange/MVS */
| /* */
| /* Release Level: Version 3, Release 1, Modification Level 0 */
| /* */
| /* Name: Network Profile Data Block (NPDB) */
| /* */
| /* Length: 300 bytes */
| /* */
| /*****/
| typedef struct NPDB NPdb; /* Provide "NPdb" data type */
| struct NPDB {

```

```

|     short blklen;           /* NPDB block length          */
|     short reserv1;         /* Reserved for future use    */
|     char blkname[8];       /* NPDB block name           */
|     char netid[8];         /* Network identification     */
|     char netname[30];      /* Network name              */
|     char comrot[8];        /* Communication routine name */
|     char netpgm[8];        /* Network program name      */
|     char pgmparm[57];      /* Network program parameters */
|     char cmdin[8];         /* Network command input file */
|     char cmdlrecl[4];      /* Network command rec length */
|     char qdata[8];         /* Transaction data queue file */
|     char datlrecl[4];      /* Transaction data rec length */
|     char tmzone[5];        /* Time zone                 */
|     char systyp[8];        /* System type               */
|     char syslvl[4];        /* System level              */
|     char txthdr[1];        /* Text header character     */
|     char cmdout[8];        /* Network pgm cmd output file */
|     char msgrou[8];        /* Message handler program   */
|     char seqnum[5];        /* Network sequence number    */
|     char netackfile[8];    /* Net acknowledgment file    */
|     char netphone[32];     /* Dial connection phone     */
|     char script[8];        /* Script name               */
|     char filler[14];       /* Reserved for future use    */
|     char descript[30];     /* Profile member description */
|     char loglock;          /* Logical lock flag         */
|     char lastuid[17];      /* Last update user ID       */
|     long lastudt;          /* Last update date/time     */
| };

```

C TPPDB

```

| \
| /*****
| /*                               DataInterchange/MVS          */
| /*                               */
| /* Release Level: Version 3, Release 1, Modification Level 0 */
| /*                               */
| /* Name: Trading Partner Profile Data Block (TPPDB)          */
| /*                               */
| /* Length: 1532 bytes                                         */
| /*                               */
| /*****/
| typedef struct TPPDB TPPdb;      /* Provide "TPPdb" data type */
| struct TPPDB {
|     short blklen;           /* TPPDB block length          */
|     short reserv1;         /* Reserved for future use    */
|     char blkname[8];       /* TPPDB block name           */
|     char tpnickm[16];      /* Trading partner nickname   */
|     char netid[8];         /* Network profile member     */
|     char sysqual[1];       /* Inter-system qualifier     */
|     char sysid[8];         /* Inter-system ID           */
|     char acctnum[32];      /* Network account number     */
|     char userid[32];       /* Network user ID           */
|     char envlqual[4];      /* Interchange qualifier     */
|     char envlid[35];       /* Interchange ID            */
|     char coname[40];       /* Company name              */
|     char addr1[40];        /* Company address line 1     */

```

	char addr2[40];	/* Company address line 2	*/
	char phone[25];	/* Contact phone number	*/
	char contact[30];	/* Contact name	*/
	char password[14];	/* Password for sending	*/
	char rcvpass[14];	/* Password for receiving	*/
	char secuid[8];	/* Security profile member	*/
	char netcls[1];	/* Network message class	*/
	char netchg[1];	/* Network charge code	*/
	char netack[1];	/* Network acknowledgment code	*/
	char netvchk[1];	/* Destination verification	*/
	char netretn[2];	/* Retention period	*/
	char netedio[1];	/* EDI processing option	*/
	char netedip[1];	/* EDI processing override	*/
	char stgfrmt[1];	/* Storage format override	*/
	char machtype[1];	/* Machine type	*/
	char stgfrmt[1];	/* Storage format	*/
	char eotid[1];	/* End of text delimiter	*/
	char logenv[1];	/* Log envelope data	*/
	char fngrpenv[1];	/* Functional group wanted	*/
	char sedelim[1];	/* Sub element delimiter	*/
	char dedelim[1];	/* Data element delimiter	*/
	char sgdelim[1];	/* Segment delimiter	*/
	char sgsep[1];	/* Segment ID separator	*/
	char decnot[1];	/* Decimal notation	*/
	char rlschar[1];	/* Release character	*/
	char tpictlno[9];	/* Interchange mask	*/
	char tpgctlno[9];	/* Group mask	*/
	char tptctlno[9];	/* Transaction mask	*/
	char comment1[40];	/* Comment line 1	*/
	char comment2[40];	/* Comment line 2	*/
	char netcmds[8];	/* Network command file member	*/
	char tpdataline[32];	/* Direct dial phone number	*/
	char timeout[4];	/* Communication line timeout	*/
	char segmented[1];	/* Segmented output	*/
	char suffix[2];	/* File suffix	*/
	char tpenvsuf[2];	/* Envelope profile suffix	*/
	char tpgenrcv;	/* Allow generic rcv usages	*/
	char tpcmpres;	/* Compression flag	*/
	char tprsrv1[8];	/* Reserved for future use	*/
	char tpsupad3[40];	/* Company address line 3	*/
	char tpsupcty[30];	/* City name	*/
	char tpsupst[2];	/* State code	*/
	char tpsuppst[15];	/* Postal code	*/
	char tpsupcon[30];	/* Country	*/
	char tpsupfax[25];	/* Contact fax number	*/
	char tpsupu3[40];	/* Comment line 3	*/
	char tpsupu4[40];	/* Comment line 4	*/
	char tpsupu5[40];	/* Comment line 5	*/
	char tpsupu6[40];	/* Comment line 6	*/
	char tpsupu7[40];	/* Comment line 7	*/
	char tpsupu8[40];	/* Comment line 8	*/
	char tpsupu9[40];	/* Comment line 9	*/
	char tpsupu10[40];	/* Comment line 10	*/
	char priority;	/* Delivery priority	*/
	char tprsrv2[3];	/* Reserved for future use	*/
	char descript[30];	/* Profile member description	*/
	char loglock;	/* Logical lock flag	*/

```

|     char lastuid[17];           /* Last update user ID      */
|     long lastudt;              /* Last update date/time   */
|     char tptype;               /* Trading partner type    */
|     char tprsrv3[467];         /* Reserved for future use */
| };

```

C REQDB

```

| /*****
| /*                               DataInterchange/MVS                               */
| /*                               */
| /* Release Level: Version 3, Release 1, Modification Level 0 */
| /*                               */
| /* Name: Requestor Profile Data Block (REQDB) */
| /*                               */
| /* Length: 264 bytes */
| /*                               */
| *****/
| typedef struct REQDB REQdb;      /* Provide "REQdb" data type */
| struct REQDB {
|     short blklen;               /* REQDB block length      */
|     short reserv1;              /* Reserved for future use */
|     char blkname[8];            /* REQDB block name        */
|     char reqid[16];             /* Requestor ID            */
|     char netid[8];              /* Network profile member  */
|     char acctno[32];            /* Network account number  */
|     char userid[32];            /* Network user ID         */
|     char passwd[16];            /* Network password        */
|     char msguc1[8];             /* Network message user class */
|     char inddname[8];           /* Receive file name       */
|     char netcls[1];             /* Network message class   */
|     char netchg[1];             /* Network charge code     */
|     char netack[1];             /* Network acknowledgment code */
|     char netvchk[1];            /* Destination verification */
|     char netretn[3];            /* Retention period        */
|     char netedio[1];            /* EDI processing option   */
|     char netedip[1];            /* EDI processing override */
|     char stgfrmt0[1];           /* Storage format override */
|     char stgfrmt[1];            /* Storage format          */
|     char netcmdmbr[8];          /* Network command file member */
|     char timeout[4];            /* Communication line timeout */
|     char ntaskpgm[8];           /* Remote message handler  */
|     char altnetphone[32];       /* Alternate dial phone number */
|     char compress;              /* Compression flag        */
|     char priority;              /* Delivery priority       */
|     char filler[15];            /* Reserved for future use */
|     char descript[30];           /* Profile member description */
|     char loglock;               /* Logical lock flag       */
|     char lastuid[17];           /* Last update user ID     */
|     long lastudt;              /* Last update date/time   */
| };

```

C DATBLK

```
/*  
*****  
/* DataInterchange/MVS */  
/* */  
/* Release Level: Version 3, Release 1, Modification Level 0 */  
/* */  
/* Name: Translator Data Block (DATBLK) */  
/* */  
/* Length: Variable */  
/* */  
/*  
*****  
typedef struct DATBLK DATAblk; /* Provide "DATAblk" data type */  
struct DATBLK {  
    short blklen; /* DATBLK block length */  
    short reserv1; /* Reserved for future use */  
    char blkname[8]; /* DATBLK block name */  
    short datlen; /* Length of data in block */  
    char data[1]; /* First byte of data */  
};
```

C HUGEBLK

```
/*  
*****  
/* DataInterchange/MVS */  
/* */  
/* Release Level: Version 3, Release 1, Modification Level 0 */  
/* */  
/* Name: Translator Huge Block (HUGEBLK) */  
/* */  
/* Length: Variable */  
/* */  
/*  
*****  
typedef struct HUGEBLK HUGEblk; /* Provide "HUGEblk" data type */  
struct HUGEBLK {  
    long blklen; /* HUGEBLK block length */  
    void *nextbuf; /* Pointer to next buffer */  
    void *prevbuf; /* Pointer to previous buffer */  
    long datlen; /* Length of data in block */  
    char data[1]; /* First byte of data */  
};
```

C SPDB

```
/*  
*****  
/* DataInterchange/MVS */  
/* */  
/* Release Level: Version 3, Release 1, Modification Level 0 */  
/* */  
/* Name: Security Profile Data Block (SPDB) */  
/* */  
/* Length: 156 bytes */  
/* */  
/*  
*****  
typedef struct SPDB SPdb; /* Provide "SPdb" data type */  
struct SPDB {  
    short blklen; /* SPDB block length */  
    short reserv1; /* Reserved for future use */  
};
```



```

|   char blkname[8];           /* SPDB block name           */
|   char secid[8];            /* Security ID               */
|   char origname[16];        /* Security originator       */
|   char recpname[16];        /* Security recipient        */
|   char authtype;            /* Authorization type        */
|   char authcode;            /* Authorization code        */
|   char encrtype;            /* Encryption type           */
|   char fltrtype;            /* Filtering type            */
|   char encrprog[8];         /* Encryption program        */
|   char authprog[8];         /* Authentication program    */
|   char compprog[8];         /* Compression program       */
|   char fltrprog[8];         /* Filtering program         */
|   char bufsize[5];          /* Buffer size               */
|   char filler[11];          /* Reserved for future use   */
|   char descrpt[30];         /* Profile member description */
|   char loglock;             /* Logical lock flag         */
|   char lastuid[17];         /* Last update user ID       */
|   long lastudt;             /* Last update date/time     */
| };

```

C Utility Control Information Block

```

| /*****
| /*                               DataInterchange/MVS                */
| /*                               */
| /* Release Level: Version 3, Release 1, Modification Level 0      */
| /*                               */
| /* Name: Utility Control Block (FFUS)                             */
| /*                               */
| /* Length: 248 bytes                                              */
| /*                               */
| *****/
| typedef struct FFUS FFus;      /* Provide "FFus" data type */
| struct FFUS {
|     long   syncval;           /* Syncpoint interval       */
|     char   *cmdp;              /* Command string address    */
|     long   cmdlen;            /* Command string length     */
|     char   cmdname[8];        /* Command file name         */
|     char   cmdtype[2];        /* Command file type         */
|     char   delimiter;         /* Command value delimiter   */
|     char   prtname[8];        /* Print file name           */
|     char   prttype[2];        /* Print file type           */
|     char   rptname[8];        /* Report file name          */
|     char   rpttype[2];        /* Report file type          */
|     char   excpname[8];       /* Exception file name       */
|     char   excptype[2];       /* Exception file type       */
|     char   trakname[8];       /* Tracking file name        */
|     char   traktype[2];       /* Tracking file type        */
|     char   qryname[8];        /* Query file name           */
|     char   qrytype[2];        /* Query file type           */
|     char   applid[8];         /* Application ID override   */
|     char   langid[6];         /* Language ID override     */
|     char   respid[8];         /* Response pgm/trx name     */
|     char   resptyp[2];        /* Response type             */
|     char   rtermid[4];        /* Response terminal ID      */
|     char   usrflid[16];       /* User field to pass data   */
|     char   sysid[8];          /* CICS system ID override   */

```

```

| char respflag; /* Response invocation reason */
| char filetype[2]; /* Network ack file type */
| long *ecbp; /* ECB address */
| long ccbr; /* Return code */
| long ccberc; /* Extended return code */
| char fileid[8]; /* Envelope TSQ (or net acks) */
| char appfile[8]; /* Translated data TSQ */
| char abndcode[4]; /* CICSabend code */
| char thandle[20]; /* Transaction handle */
| void *ffiehdr; /* IE long header address */
| long farc; /* Func ack return code */
| long faerc; /* Func ack ext return code */
| char fabuilt; /* Func ack built flag */
| char ffmtflg; /* Multi-TSQ indicator */
| char usersync; /* User syncpoint flag */
| char res1; /* Reserved for future use */
| void *usercond; /* User condition code pointer */
| char res2[23]; /* Reserved for future use */
| char respactv; /* Response pgm active flag */
| char workname[8]; /* Workfile name */
| char batflg; /* Batch or UTILSRV API flag */
| char noexcp; /* No exception file */
| char invparm; /* Network pgm invalid parm */
| char logactv; /* Logging CE0050 active */
| void *ffusaddr; /* Pointer to this FFUS block */
| void *ccbp; /* Pointer to the CCB */
| void *ffmtcbp; /* Multi-TSQ control blk ptr */
| };

```

Copy Books for ASSEMBLER

ASSEMBLER CCB

```

| *****
| *                               *
| * DataInterchange/MVS          *
| *                               *
| * Release Level: Version 3, Release 1, Modification Level 0 *
| *                               *
| * Name: Common Control Block (CCB) *
| *                               *
| * Length: 608 bytes *
| *                               *
| *****
| CCB      DSECT ,
|          DS    0D
| ZCCBLL   DS    H          CCB block length
| ZCCBID   DS    H          Reserved for future use
| ZCCBEYE  DS    CL8        CCB block name
| ZCCBRC   DS    F          Return code
| ZCCBERC  DS    F          Extended return code
| ZCCBSID  DS    CL8        System ID
| ZCCBUID  DS    CL8        User ID
| ZCCBAID  DS    CL8        Application ID
| ZCCBCID  DS    CL8        Error module ID
| ZCCBXFID DS    H          Component function ID
| ZCCBLPID DS    CL6        Language profile ID

```

ZCCBCPID DS	F	Code page ID
ZCCBRVS DS	F	Reserved for future use
ZCCBCCXP DS	F	CCB extension pointer
ZCCBCABP DS	F	Common area block pointer
ZCCBDBID DS	CL4	MVS DB2 subsystem ID
ZCCBDBPL DS	CL8	MVS DB2 plan / AIX alias
ZCCBDBUI DS	CL8	AIX DB2 user ID
ZCCBDBPW DS	CL18	AIX DB2 password
ZCCBRVS1 DS	CL26	DI internal use only
ZCCBRVS2 DS	117F	DI internal use only

ASSEMBLER SNB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Service Name Block (SNB)                                                 *
*                                                                                   *
* Length: 32 bytes                                                                *
*                                                                                   *
*****
SNB      DSECT ,
          DS    0D
ZSNBLL   DS    H          SNB block length
ZSNBID   DS    H          Reserved for future use
ZSNBEYE  DS    CL8        SNB block name
ZSNBNAME DS    CL8        Service name
ZSNBNDX  DS    F          Service index
ZSNBPC   DS    H          Service parameter count
ZSNBFLG0 DS    CL1        First flag byte
ZSNBFLG1 DS    CL1        Second flag byte
ZSNBFANC DS    F          First anchor pointer

```

ASSEMBLER FCB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Function Code Block (FCB)                                                 *
*                                                                                   *
* Length: 4 bytes                                                                *
*                                                                                   *
*****
FCB      DSECT ,
          DS    0D
ZFCBLL   DS    H          FCB block length
ZFCBFUNC DS    H          Service function code

```

ASSEMBLER CMCB

```

*****
*                               DataInterchange/MVS                               *
*                               *                                                 *
* Release Level: Version 3, Release 1, Modification Level 0                     *
*                               *                                                 *
* Name: Communications Control Block (CMCB)                                     *
*                               *                                                 *
* Length: 254 bytes                                                         *
*                               *                                                 *
*****
CMCB      DSECT ,
          DS      0D
BLKLEN    DS      H                  CMCB block length
RESERV1    DS      H                  Reserved for future use
BLKNME     DS      CL8                CMCB block name
TPNICKNM   DS      CL16               Trading partner nickname
NETID      DS      CL8                Network ID
NETOP      DS      CL8                Network operation
REQID      DS      CL16               Requestor ID
SEQNUM     DS      CL5                Network sequence number
CONTRCV    DS      CL1                Continuous receive flag
FTYPE      DS      CL2                Send/receive file type
FILERCVD   DS      CL1                File received indicator
RESERV2    DS      CL5                Reserved for future use
CLRFILE    DS      CL1                Clear file after processing
DATAFMT    DS      CL1                Data format
ACCTYP     DS      CL1                Account type
DATATYP    DS      CL1                Data type ddname/dataset
RECVTYP    DS      CL1                Receive first or all files
DCIND      DS      CL1                Delivery priority
ACKIND     DS      CL1                Acknowledgment request code
RESRECL    DS      CL1                Resolve record length
SCRIPT     DS      CL8                Script name
ENAME      DS      CL8                Message user class
MSGNAME    DS      CL8                Message name
FILENAME   DS      CL56               Ddname or dataset name
CANSND     DS      CL6                Cancellation start date
CANST      DS      CL6                Cancellation start time
CANED      DS      CL6                Cancellation end date
CANET      DS      CL6                Cancellation end time
TMZONE     DS      CL1                Time zone for time interval
MODEM      DS      CL1                Modem type
NPSSCDE    DS      CL5                Start session response code
NPESCDE    DS      CL5                End session response code
NPERRCD    DS      CL5                Network error code
NPSEVER    DS      CL2                Network error code severity
BLKTYPE    DS      CL1                Data block type huge or VA
FQUEUED    DS      CL1                Queue functions supported
FMSG      DS      CL1                Free-form msgs supported
FFILE      DS      CL1                Nonstandard files supported
FEDIX      DS      CL1                EDI ISA/IEA envs supported
FEDIE      DS      CL1                EDI UNB/UNZ envs supported
FEDIU      DS      CL1                EDI BG/EG envs supported
FEDIG      DS      CL1                EDI GS/GE envs supported
FEDII      DS      CL1                EDI ICS/ICE envs supported
FEDIT      DS      CL1                EDI STX/END envs supported

```

	FCANCEL	DS	CL1	Cancel function supported
	FCLASS	DS	CL1	User msg classes supported
	FAK	DS	CL1	Network acks supported
	FSYSMSG	DS	CL1	System messages supported
	FRCVBTP	DS	CL1	Receive by trading partner
	FRESTART	DS	CL1	Restart supported
	FNOUSERI	DS	CL1	Acct number only supported
	FACCTSEP	DS	CL1	Account/user ID separator
	DDCOLON	DS	CL3	DD: string for IEBASE
	RESERV3	DS	CL2	Reserved for future use
	ADMTYPE	DS	CL2	Admin response file type
	UNIQID	DS	CL8	Unique ID returned by Exp
	SAPUPDT	DS	CL1	VANI SAP status tracking
	FSENTNET	DS	CL1	Skip send-requested status
	RESERV4	DS	CL14	Reserved for future use

ASSEMBLER TRCB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Translator Control Block (TRCB)                                           *
*                                                                                   *
* Length: 1536 bytes                                                             *
*                                                                                   *
*****
TRCB      DSECT ,
          DS      0D
BLKLEN    DS      H                    TRCB block length
RSRVD1    DS      H                    Reserved for future use
BLKNME    DS      CL8                  TRCB block name
REQID     DS      CL16                 Requestor ID
APPFID    DS      CL8                  Application file written to
ATFID     DS      CL16                 Application data format ID
ATSID     DS      CL16                 Structure name returned
EJECT     DS      CL1                  End of current transaction
INTPID    DS      CL35                 Internal trading partner ID
APPCTLNU  DS      CL32                 Application control number
TRNID     DS      CL16                 Standard transaction ID
TEST      DS      CL1                  Usage indicator (I,P,T,U)
IHCTL     DS      CL9                  Interchange control number
GHCTL     DS      CL9                  Group control number
THCTL     DS      CL9                  Transaction control number
ENVTYPE   DS      CL1                  Envelope type (E,I,T,U,X)
BLKTYPE   DS      CL1                  TRIDB and TRODB format
DUPTRAN   DS      CL1                  Duplicate transaction flag
ITPBREAK  DS      CL1                  ITP change causes new env
REQSIZE   DS      F                    Output buffer required size
XPANDED   DS      CL1                  Expanded TRCB block flag
NEWENV    DS      CL1                  Transaction caused new env
NEWGRP    DS      CL1                  Transaction caused new grp
NEWTRN    DS      CL1                  Transaction caused new tran
QSIZE     DS      F                    Total queued envelope size
ESIZE     DS      F                    Current envelope size
GRPNUM    DS      F                    Number of groups in env

```

TRNNUM	DS	F	Number of trans in env
SEGNUM	DS	F	Number of segments in env
TRNGRP	DS	F	Number of trans in group
SEGTRN	DS	F	Number of segments in tran
ERRNUM	DS	F	Number of translate errors
QBT	DS	CL8	QSIZE character equivalent
IHXCTL	DS	CL14	Interchange hdr control num
ISYNTAXI	DS	CL4	Syntax ID for E,T envelopes
ISYNTAXV	DS	CL1	Syntax verification for E,T
ISIDQUAL	DS	CL4	Sender qualifier for E,I,X
ISID	DS	CL35	Interchange sender ID
ISENDNAM	DS	CL14	Sender name for U,T envs
IREVROUT	DS	CL14	Reverse routing addr E envs
IRIDQUAL	DS	CL4	Receiver qual for E,I,X
IRID	DS	CL35	Interchange receiver ID
IRECVNAM	DS	CL14	Receiver name for U,T envs
IROUTEAD	DS	CL14	Routing address for E envs
IDATE	DS	CL6	Interchange date
ITIME	DS	CL6	Interchange time
IVERREL	DS	CL5	Interchange version/release
IGT	DS	CL6	GRPNUM character equivalent
ITT	DS	CL6	TRNNUM character equivalent
IST	DS	CL10	SEGNUM character equivalent
IBT	DS	CL8	ESIZE character equivalent
ISPW	DS	CL14	Interchange password
IAPREF	DS	CL14	Application reference
ISTDID	DS	CL4	Interchange standard ID
RSRVD2	DS	CL1	Reserved for future use
IPRIOR	DS	CL1	Interchange priority code
ICOMMAGR	DS	CL35	Communucation agreement
GHXCTL	DS	CL14	Group hdr control number
GFGID	DS	CL6	Group functional group ID
GSIDQUAL	DS	CL4	Group sender ID qualifier
GSID	DS	CL35	Group sender ID
GRIDQUAL	DS	CL4	Group receiver ID qualifier
GRID	DS	CL35	Group receiver ID
GDATE	DS	CL6	Group date
GTIME	DS	CL6	Group time
GVER	DS	CL12	Group version
GREL	DS	CL12	Group release
GTT	DS	CL6	TRNGRP character equivalent
GAPW	DS	CL14	Group password
GRESPAGE	DS	CL2	Group responsible agency
RSRVD3	DS	CL12	Reserved for future use
THXCTL	DS	CL14	Transaction control number
TTC	DS	CL6	Current transaction ID
TVER	DS	CL6	Transaction version
TREL	DS	CL6	Transaction release
TST	DS	CL10	SEGTRN character equivalent
LASTINEN	DS	CL1	Last transaction in env
XACFIELD	DS	CL35	Application control field
FABUILT	DS	CL1	Functional ack built flag
QGTNUM	DS	F	Queued group total
QTTNUM	DS	F	Queued transaction total
QSTNUM	DS	F	Queued segment total
QGT	DS	CL6	QGTNUM character equivalent
QTT	DS	CL6	QTTNUM character equivalent

QST	DS	CL10	QSTNUM character equivalent
FASPM	DS	CL8	FA standard profile member
ENVCHK	DS	CL1	Verify previous enveloping
TRNSTAT	DS	CL1	Transaction status
APTYPE	DS	CL2	Application file type
TSKEY	DS	CL10	Packed transaction handle
TSKEYU	DS	CL20	Unpacked transaction handle
MAPKEY	DS	CL16	Map used for translation
BATCHID	DS	CL8	Batch ID
ENVLDATE	DS	CL8	Earliest envelope date
TRXLIFE	DS	H	Trans store tran life
IMGLIFE	DS	H	Trans store image life
HOLDFLAG	DS	CL1	Trans store hold status
BNDLFLAG	DS	CL1	Trx starts a new bundle
RAWDATA	DS	CL1	Raw data indicator
ENVLELA	DS	CL1	Delay enveloping flag
TRXACCEP	DS	CL1	Trx acceptable flag
TRABORT	DS	CL1	Translator terminated flag
FILEID	DS	CL8	Standard data file name
DSNAME	DS	CL56	Physical dataset name
QNETID	DS	CL8	Trading partner's NETID
QPTTOPT	DS	CL1	Point-to-point QNETID flag
QSRPGM	DS	CL1	Send/recv prog QNETID flag
QDDNAME	DS	CL8	Standard data file name
FUNACKFL	DS	CL8	Functional ack file name
QTPNICK	DS	CL16	Trading partner nickname
QRC	DS	H	Queueing return code
QERC	DS	H	Queueing ext return code
ERRCDDES	DS	10H	First 10 errors encountered
INMEMTRA	DS	H	Number of trxs in storage
NOCOMMIT	DS	CL1	Database commits allowed
SCOPE	DS	CL1	Env vs trx level recovery
TPNICK	DS	CL16	Trading partner nickname
CONCATEN	DS	CL1	Concatenate records flag
ASSERTLV	DS	CL1	Assertion level
RAWDATAO	DS	CL1	Raw data output
FIXEDTRX	DS	CL1	Fixed-to-fixed transaction
MRREQID	DS	CL16	Management reporting REQID
ERRFILTE	DS	CL80	Error filter values
FFILEID	DS	CL8	Fixed-to-fixed file name
VARTRACE	DS	CL1	Variable trace wanted
EXPTRACE	DS	CL1	Expression trace wanted
SSEGVAL	DS	CL1	Service segment validation
TRXFACOD	DS	CL1	Transaction func ack code
IUSEREXI	DS	CL8	User exit for VA processing
IUSERARE	DS	F	User exit user area
IUSERACC	DS	CL1	User exit access type
IUSERTYP	DS	CL2	User exit program type
MAPCHAIN	DS	CL1	Mapchaining active flag
FORCETES	DS	CL1	Use inbound test usage
ENVPRBRK	DS	CL1	Envelope profile break
SUSBLKF	DS	CL1	CICS SUSPEND block flag
CLRERRS	DS	CL1	Clear ERRCDDES array flag
FARC	DS	F	Func ack return code
FAERC	DS	F	Func ack ext return code
SAPUPDT	DS	CL1	SAP status tracking wanted
SAPTRX	DS	CL1	SAP transaction flag

	SAPCLIEN	DS	CL3	SAP client
	SAPDOCNU	DS	CL16	SAP document number
	SAPSEQ	DS	CL6	SAP error sequence number
	ROUTCODE	DS	CL3	Generic usage routing code
	FAEREQ	DS	CL1	Func ack file required
	BOUNDARY	DS	CL1	Incremental translation
	SUSBLKP	DS	F	CICS SUSPEND block pointer
	CUSERDAT	DS	CL256	C record user data area
	APPLTPID	DS	CL16	Application trading partner
	EXTENDC	DS	CL1	Extend the C record flag
	VAXFLAG	DS	CL1	Pageable translation
	GDATE8	DS	CL8	Group envelope 8-byte date
	RECOVBAD	DS	CL1	Recover from bad qdata
	RSRVD4	DS	CL61	Reserved for future use

ASSEMBLER NPDB

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Network Profile Data Block (NPDB)                                         *
*                                                                                   *
* Length: 300 bytes                                                                *
*                                                                                   *
*****
NPDB      DSECT ,
          DS      0D
BLKLEN    DS      H                      NPDB block length
RESERV1    DS      H                      Reserved for future use
BLKNME     DS      CL8                    NPDB block name
NETID      DS      CL8                    Network identification
NETNME     DS      CL30                   Network name
COMROT     DS      CL8                    Communication routine name
NETPGM     DS      CL8                    Network program name
PGMPARM    DS      CL57                   Network program parameters
CMDIN      DS      CL8                    Network command input file
CMDLRECL   DS      CL4                    Network command rec length
QDATA      DS      CL8                    Transaction data queue file
DATLRECL   DS      CL4                    Transaction data rec length
TMZONE     DS      CL5                    Time zone
SYSTYP     DS      CL8                    System type
SYSLVL     DS      CL4                    System level
TXTHDR     DS      CL1                    Text header character
CMDOUT     DS      CL8                    Network pgm cmd output file
MSGROUT    DS      CL8                    Message handler program
SEQNUM     DS      CL5                    Network sequence number
NETACKFI   DS      CL8                    Net acknowledgment file
NETPHONE   DS      CL32                   Dial connection phone
SCRIPT     DS      CL8                    Script name
FILLER     DS      CL14                   Reserved for future use
DESCRIPT   DS      CL30                   Profile member description
LOGLOCK    DS      CL1                    Logical lock flag
LASTUID    DS      CL17                   Last update user ID
LASTUDT    DS      F                      Last update date/time

```


ASSEMBLER TPPDB

```

*****
*                               DataInterchange/MVS                               *
*                               *                                                 *
* Release Level: Version 3, Release 1, Modification Level 0                     *
*                               *                                                 *
* Name: Trading Partner Profile Data Block (TPPDB)                             *
*                               *                                                 *
* Length: 1532 bytes                                                         *
*                               *                                                 *
*****
TPPDB    DSECT ,
        DS      0D
BLKLEN   DS      H                TPPDB block length
RESERV1  DS      H                Reserved for future use
BLKNME   DS      CL8              TPPDB block name
TPNICKNM DS      CL16             Trading partner nickname
NETID    DS      CL8              Network profile member
SYSQUAL  DS      CL1              Inter-system qualifier
SYSID    DS      CL8              Inter-system ID
ACCTNUM  DS      CL32             Network account number
USERID   DS      CL32             Network user ID
ENVLQUAL DS      CL4              Interchange qualifier
ENVLID   DS      CL35             Interchange ID
CONAME   DS      CL40             Company name
ADDR1    DS      CL40             Company address line 1
ADDR2    DS      CL40             Company address line 2
PHONE    DS      CL25             Contact phone number
CONTACT  DS      CL30             Contact name
PASSWORD DS      CL14             Password for sending
RCVPASS  DS      CL14             Password for receiving
SECUID   DS      CL8              Security profile member
NETCLS   DS      CL1              Network message class
NETCHG   DS      CL1              Network charge code
NETACK   DS      CL1              Network acknowledgment code
NETVCHK  DS      CL1              Destination verification
NETRETN  DS      CL3              Retention period
NETEDIO  DS      CL1              EDI processing option
NETEDIP  DS      CL1              EDI processing override
STGFRMTO DS      CL1              Storage format override
MACHTYPE DS      CL1              Machine type
STGFRMT  DS      CL1              Storage format
EOTID    DS      CL1              End of text delimiter
LOGENV   DS      CL1              Log envelope data
FNGRPENV DS      CL1              Functional group wanted
SEDELIM  DS      CL1              Sub element delimiter
DEDELIM  DS      CL1              Data element delimiter
SGDELIM  DS      CL1              Segment delimiter
SGSEP    DS      CL1              Segment ID separator
DECNOT   DS      CL1              Decimal notation
RLSCHAR  DS      CL1              Release character
TPICTLNO DS      CL9              Interchange mask
TPGCTLNO DS      CL9              Group mask
TPTCTLNO DS      CL9              Transaction mask
COMMENT1 DS      CL40             Comment line 1
COMMENT2 DS      CL40             Comment line 2
NETCMDS  DS      CL8              Network comand file member

```

	TPDATAI	DS	CL32	Direct dial phone number
	TIMEOUT	DS	CL4	Communication line timeout
	SEGMENTE	DS	CL1	Segmented output
	SUFFIX	DS	CL2	File suffix
	TPENVISUF	DS	CL2	Envelope profile suffix
	TPGENRCV	DS	CL1	Allow generic recv usages
	TPCMPRES	DS	CL1	Compression flag
	TPRSRV1	DS	CL8	Reserved for future use
	TPSUPAD3	DS	CL40	Company address line 3
	TPSUPCTY	DS	CL30	City name
	TPSUPST	DS	CL2	State code
	TPSUPPST	DS	CL15	Postal code
	TPSUPCON	DS	CL30	Country
	TPSUPFAX	DS	CL25	Contact fax number
	TPSUPU3	DS	CL40	Comment line 3
	TPSUPU4	DS	CL40	Comment line 4
	TPSUPU5	DS	CL40	Comment line 5
	TPSUPU6	DS	CL40	Comment line 6
	TPSUPU7	DS	CL40	Comment line 7
	TPSUPU8	DS	CL40	Comment line 8
	TPSUPU9	DS	CL40	Comment line 9
	TPSUPU10	DS	CL40	Comment line 10
	PRIORITY	DS	CL1	Delivery priority
	TPRSRV2	DS	CL3	Reserved for future use
	DESCRIPT	DS	CL30	Profile member description
	LOGLOCK	DS	CL1	Logical lock flag
	LASTUID	DS	CL17	Last update user ID
	LASTUDT	DS	F	Last update date/time
	TPTYPE	DS	CL1	Trading partner type
	TPRSRV3	DS	CL467	Reserved for future use

ASSEMBLER REQDB

```

*****
*                               DataInterchange/MVS                               *
*                               *                                                 *
* Release Level: Version 3, Release 1, Modification Level 0                     *
*                               *                                                 *
* Name: Requestor Profile Data Block (REQDB)                                   *
*                               *                                                 *
* Length: 264 bytes                                                         *
*                               *                                                 *
*****
REQDB    DSECT ,
         DS    0D
BLKLEN   DS    H                    REQDB block length
RESERV1  DS    H                    Reserved for future use
BLKNME   DS    CL8                  REQDB block name
REQID    DS    CL16                 Requestor ID
NETID    DS    CL8                  Network profile member
ACCTNO   DS    CL32                 Network account number
USERID   DS    CL32                 Network user ID
PASSWD   DS    CL16                 Network password
MSGUCL   DS    CL8                  Network message user class
INDDNAME DS    CL8                  Receive file name
NETCLS   DS    CL1                  Network message class
NETCHG   DS    CL1                  Network charge code

```

	NETACK	DS	CL1	Network acknowledgment code
	NETVCHK	DS	CL1	Destination verification
	NETRETN	DS	CL3	Retention period
	NETEDIO	DS	CL1	EDI processing option
	NETEDIP	DS	CL1	EDI processing override
	STGFRMTO	DS	CL1	Storage format override
	STGFRMT	DS	CL1	Storage format
	NETCMDMB	DS	CL8	Network command file member
	TIMEOUT	DS	CL4	Communication line timeout
	NTACKPGM	DS	CL8	Remote message handler
	ALTNETPH	DS	CL32	Alternate dial phone number
	COMPRESS	DS	CL1	Compression flag
	PRIORITY	DS	CL1	Delivery priority
	FILLER	DS	CL15	Reserved for future use
	DESCRIPT	DS	CL30	Profile member description
	LOGLOCK	DS	CL1	Logical lock flag
	LASTUID	DS	CL17	Last update user ID
	LASTUDT	DS	F	Last update date/time

ASSEMBLER DATBLK

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Translator Data Block (DATBLK)                                           *
*                                                                                   *
* Length: Variable                                                                *
*                                                                                   *
*****
DATBLK DSECT ,
      DS      0D
BLKLEN DS      H      DATBLK block length
RESERV1 DS      H      Reserved for future use
BLKNME DS      CL8     DATBLK block name
DATLEN DS      H      Length of data in block
DATA   DS      CL1     First byte of data

```

ASSEMBLER HUGEBLK

```

*****
*                               DataInterchange/MVS                               *
*                                                                                   *
* Release Level: Version 3, Release 1, Modification Level 0                       *
*                                                                                   *
* Name: Translator Huge Block (HUGEBLK)                                          *
*                                                                                   *
* Length: Variable (greater than 32 K)                                           *
*                                                                                   *
*****
HUGEBLK DSECT ,
      DS      0D
BLKLEN DS      F      HUGEBLK block length
NEXTBUF DS      F      Pointer to next buffer
PREVBUF DS      F      Pointer to previous buffer

```

DATLEN	DS	F	Length of data in block
DATA	DS	CL1	First byte of data

ASSEMBLER SPDB

```

*****
*                               DataInterchange/MVS                               *
*                               *                                                 *
* Release Level: Version 3, Release 1, Modification Level 0                     *
*                               *                                                 *
* Name: Security Profile Data Block (SPDB)                                     *
*                               *                                                 *
* Length: 156 bytes                                                         *
*                               *                                                 *
*****
SPDB      DSECT ,
          DS      0D
BLKLEN    DS      H                SPDB block length
RESERV1    DS      H                Reserved for future use
BLKNME     DS      CL8             SPDB block name
SECID      DS      CL8             Security ID
ORIGNAME   DS      CL16            Security originator
RECPNAME   DS      CL16            Security recipient
AUTHTYPE   DS      CL1            Authorization type
AUTHCODE   DS      CL1            Authorization code
ENCRTYPE   DS      CL1            Encryption type
FLTRTYPE   DS      CL1            Filtering type
ENCRPROG   DS      CL8            Encryption program
AUTHPROG   DS      CL8            Authentication program
COMPPROG   DS      CL8            Compression program
FLTRPROG   DS      CL8            Filtering program
BUFSIZE    DS      CL5            Buffer size
FILLER     DS      CL11           Reserved for future use
DESCRIPT   DS      CL30           Profile member description
LOGLOCK    DS      CL1            Logical lock flag
LASTUID    DS      CL17           Last update user ID
LASTUDT    DS      F              Last update date/time

```

ASSEMBLER Utility Control Information Block

```

*****
*                               DataInterchange/MVS                               *
*                               *                                                 *
* Release Level: Version 3, Release 1, Modification Level 0                     *
*                               *                                                 *
* Name: Utility Control Block (FFUS)                                           *
*                               *                                                 *
* Length: 248 bytes                                                         *
*                               *                                                 *
*****
FFUS      DSECT ,
          DS      0D
SYNCVAL    DS      F                Syncpoint interval
CMDP       DS      F                Command string address
CMDLEN     DS      F                Command string length
CMDNAME    DS      CL8             Command file name
CMDTYPE    DS      CL2             Command file type

```

DELIMITE	DS	CL1	Command value delimiter
PRTNAME	DS	CL8	Print file name
PRTTYPE	DS	CL2	Print file type
RPTNAME	DS	CL8	Report file name
RPTTYPE	DS	CL2	Report file type
EXCPNAME	DS	CL8	Exception file name
EXCPTYPE	DS	CL2	Exception file type
TRAKNAME	DS	CL8	Tracking file name
TRAKTYPE	DS	CL2	Tracking file type
QRYNAME	DS	CL8	Query file name
QRYTYPE	DS	CL2	Query file type
APPLID	DS	CL8	Application ID override
LANGID	DS	CL6	Language ID override
RESPID	DS	CL8	Response pgm/trx name
RESPTYP	DS	CL2	Response type
RTERMID	DS	CL4	Response terminal ID
USRFLD	DS	CL16	User field to pass data
SYSID	DS	CL8	CICS system ID override
RESPFLAG	DS	CL1	Response invocation reason
FILETYP	DS	CL2	Network ack file type
ECBP	DS	F	ECB address
CCBRC	DS	F	Return code
CCBERC	DS	F	Extended return code
FILEID	DS	CL8	Envelope TSQ (or net acks)
APPFILE	DS	CL8	Translated data TSQ
ABNDCODE	DS	CL4	CICS abend code
THANDLE	DS	CL20	Transaction handle
FFIEHDR	DS	F	IE long header address
FARC	DS	F	Func ack return code
FAERC	DS	F	Func ack extend return code
FABUILT	DS	CL1	Func ack built flag
FFMTFLG	DS	CL1	Multi-TSQ indicator
USERSYNC	DS	CL1	User syncpoint flag
RES1	DS	CL1	Reserved for future use
USERCOND	DS	F	User condition code pointer
RES2	DS	CL23	Reserved for future use
RESPACTV	DS	CL1	Response pgm active flag
WORKNAME	DS	CL8	Workfile name
BATFLG	DS	CL1	Batch or UTILSRV API flag
NOEXCP	DS	CL1	No exception file
INVPARM	DS	CL1	Network pgm invalid parm
LOGACTV	DS	CL1	Logging CE0050 active
FFUSADDR	DS	F	Pointer to this FFUS block
CCBP	DS	F	Pointer to the CCB
FFMTCBP	DS	F	Multi-TSQ control block ptr

Appendix D. Sample Programs

This Appendix provides sample programs and exit routines.

Creating Tagged Import Files from Fixed Flat Files

Sample programs included in the product enable you to convert a fixed flat file into a tagged import file for importing trading partner profile (TPPROF) members, send transaction usages, and receive transaction usages.

These sample programs and JCL contain extensive documentation within the programs describing how they work and how they can be used.

These programs are:

- A COBOL program named EDIXF2T, which is the main program executed.
- An Assembler program named EDIXTAGF, which is link edited with EDIXTAGF and is a formatting service to create the tags.
- Sample JCL named EDIXF2T used to define the fixed flat file and to execute the conversion program EDIXF2T.

These programs and JCL are provided as source and can be modified, compiled, and link edited to suit your individual needs.

Initializing and Terminating DataInterchange

The following three sections show examples of initializing and terminating DataInterchange with COBOL, PL/I, and C programs.

COBOL Initialization/Termination Example

The following COBOL program illustrates how to initialize and terminate DataInterchange. For more information on the SNB definition, see "COBOL SNB" on page C-3, for the CCB definition, see "COBOL CCB" on page C-2, and for the FCB definition, see "COBOL FCB" on page C-3.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. EDICOB.
```

```
*****  
*                                                                 *  
* This COBOL Command Level CICS program illustrates how        *  
* to define the Application Programming Interface Control        *  
* Blocks and call the DataInterchange Service Director to      *  
* Initialize and Terminate DataInterchange.                     *  
*                                                                 *  
*****
```

```
ENVIRONMENT DIVISION.
```

```
DATA DIVISION.
```

```
WORKING-STORAGE SECTION.
```

```

01 WS-APPLID                PIC X(08) VALUE 'EDIFFS '.

01 WS-MSG.
   05 FILLER                 PIC X(11) VALUE 'DI INIT RC='.
   05 WS-RC1                 PIC 9(04).
   05 FILLER                 PIC X(05) VALUE ' ERC='.
   05 WS-ERC1                PIC 9(04).
   05 FILLER                 PIC X(11) VALUE ' TERM RC='.
   05 WS-RC2                 PIC 9(04).
   05 FILLER                 PIC X(05) VALUE ' ERC='.
   05 WS-ERC2                PIC 9(04).

```

```

*****
*   Service Name Block - SNB                               *
*****
01 SNB-DATA.
COPY DISNB.

```

```

*****
*   Common Control Block - CCB                             *
*****
01 CCB-DATA.
COPY DICCB.

```

```

*****
*   Function Code Block - FUNCBLK                         *
*****
01 FCB-DATA.
COPY DIFCB.

```

LINKAGE SECTION.

```

01 DFHCOMMAREA              PIC X(01).

```

PROCEDURE DIVISION.

```

*****
*   Clear the Control Blocks                               *
*****

```

```

      MOVE LOW-VALUES TO SNB-DATA.
      MOVE LOW-VALUES TO CCB-DATA.
      MOVE LOW-VALUES TO FCB-DATA.

```

```

*****
*   Initialize DataInterchange                             *
*****

```

```

      MOVE 32           TO SNB-ZSNBLL.
      MOVE 'ENVSERV'    TO SNB-ZSNBNAME.
      MOVE 4            TO SNB-ZSNBPC.

```

```

      MOVE 608          TO CCB-ZCCBLL.
      MOVE 'ENU'        TO CCB-ZCCBLPID.

```



```

MOVE 4          TO FCB-ZFCBLL.
MOVE 1          TO FCB-ZFCBFUNC.

CALL 'FXXZCBL'  USING SNB-DATA
                   CCB-DATA
                   FCB-DATA
                   WS-APPLID.

MOVE CCB-ZCCBRC TO WS-RC1.
MOVE CCB-ZCCBERC TO WS-ERC1.

*****
*   Application specific code would be placed here   *
*****

*****
*   Terminate DataInterchange                         *
*****

MOVE 'ENVSERV'  TO SNB-ZSNBNAME.
MOVE 3          TO SNB-ZSNBPC.

MOVE 2          TO FCB-ZFCBFUNC.

CALL 'FXXZCBL'  USING SNB-DATA
                   CCB-DATA
                   FCB-DATA.

MOVE CCB-ZCCBRC TO WS-RC2.
MOVE CCB-ZCCBERC TO WS-ERC2.

*****
*   Write Return Code message to terminal             *
*****

EXEC CICS
  SEND FROM (WS-MSG) LENGTH (48) ERASE
END-EXEC.

EXEC CICS
  RETURN
END-EXEC.

STOP RUN.

```

PL/I Initialization/Termination Example

The following PL/I program illustrates how to initialize and terminate DataInterchange. For more information on the SNB definition, see “PL/I SNB” on page C-14, for the CCB definition, see “PL/I CCB” on page C-13, and for the FCB definition, see “PL/I FCB” on page C-14.

```
EDIPLIA: PROC OPTIONS(MAIN,REENTRANT) REORDER;
```

```

/*****/
/*
/* This PLI Command Level CICS program illustrates how
/* to define the Application Programming Interface Control */

```

```

/* Blocks and call the DataInterchange Service Director to */
/* Initialize and Terminate DataInterchange.                */
/*                                                          */
/*****

DCL FXXZPLI ENTRY OPTIONS(ASSEMBLER,INTER);

DCL APP          CHAR(08) INIT ('EDIFFS');

DCL ADDR         BUILTIN;

DCL 1 MSG,
    2 FILL01      CHAR(11) INIT ('DI INIT RC='),
    2 RC1         PIC '9999',
    2 FILL02      CHAR(05) INIT (' ERC='),
    2 ERC1        PIC '9999',
    2 FILL03      CHAR(11) INIT (' TERM RC='),
    2 RC2         PIC '9999',
    2 FILL04      CHAR(05) INIT (' ERC='),
    2 ERC2        PIC '9999';

/*****
/* Service Name Block - SNB                                */
/*****

DCL 1 SNB,
    %INCLUDE DISNB;

/*****
/* Common Control Block - CCB                                */
/*****

DCL 1 CCB,
    %INCLUDE DICCB;

/*****
/* Function Code Block - FUNCBLK                            */
/*****

DCL 1 FUNCBLK,
    %INCLUDE DIFCB;

/*****
/* Initialize DataInterchange                                */
/*****

SNB = '';
SNB.ZSNBLL = 32;
SNB.ZSNBNAME = 'ENVSERV ';
SNB.ZSNBPC = 4;

CCB = '';
CCB.ZCCBLL = 608;
CCB.ZCCBLPID = 'ENU  ';

```

```

FUNCBLK.ZFCBLL = 4;
FUNCBLK.ZFCBFUNC = 1;

CALL FXXZPLI(SNB,CCB,FUNCBLK,APP);

MSG.RC1 = CCB.ZCCBRC;
MSG.ERC1 = CCB.ZCCBERC;

/*****
/* Application specific code would be placed here */
*****/

/*****
/* Terminate DataInterchange */
*****/

SNB.ZSNBNAME = 'ENVSERV ';
SNB.ZSNBPC = 3;

FUNCBLK.ZFCBFUNC = 2;

CALL FXXZPLI(SNB,CCB,FUNCBLK);

MSG.RC2 = CCB.ZCCBRC;
MSG.ERC2 = CCB.ZCCBERC;

/*****
/* Write Return Code message to terminal */
*****/

EXEC CICS SEND FROM (MSG) LENGTH (48) ERASE;

EXEC CICS RETURN;

END EDIPLIA;

```

C Initialization/Termination Example

The following C program illustrates how to initialize and terminate DataInterchange. For more information on the SNB definition, see “C SNB” on page C-25, for the CCB definition, see “C CCB” on page C-25, and for the FCB definition, see “C FCB” on page C-26.

```

/* ----- */
/* Get the control block definitions */
/* ----- */
#include "stdio.h"           /* C I/O routines */
#include "disnb.h"           /* get SNB definition */
#include "diccb.h"           /* get CCB definition */
#include "difcb.h"           /* get FCB definition */

/* ----- */
/* Prototypes for internal functions */
/* ----- */
static int check_ierr(ccb*);
static int check_terr(ccb*);

```

```

    int
main()
{
    ccb  DIccb;                /* DataInterchange common block */

    /* ----- */
    /* Call a routine to Initialize DataInterchange */
    /* ----- */
    if (!DIinit(&DIccb)) {
        /* ----- */
        /* Application logic goes here */
        /* ----- */
        /* ----- */
        /* Call a routine to Terminate DataInterchange */
        /* ----- */
        DIterm(&DIccb);
    }

    /* ----- */
    /* A subroutine to initialize DataInterchange */
    /* ----- */
    int
DIinit(CCBptr)
    ccb *CCBptr;                /* Pointer to Common Control Block */
{
    snb  SNB;                  /* Local snb */
    fcb  FUNCBLK;              /* Local fcb */

    /* ----- */
    /* Initialize the CCB */
    /* 1. Clear it */
    /* 2. Set CCB length */
    /* 3. Move in language indicator */
    /* ----- */
    memset(CCBptr,'\0',sizeof(ccb));
    CCBptr->zccbll=sizeof(ccb);
    memcpy(CCBptr->zccbldid,"ENU  ",6);

    /* ----- */
    /* Initialize the SNB for environmental services */
    /* 1. Clear it */
    /* 2. Set SNB length */
    /* 3. Set the number of parameters that will be passed */
    /* 4. Set the name of the SERVICE being called */
    /* ----- */
    memset(&SNB,'\0',sizeof(SNB));
    SNB.zsnbll=sizeof(SNB);
    SNB.zsnbpc=5;
    memcpy(SNB.zsnbname,"ENVSERV ",8);

    /* ----- */
    /* Initialize the FUNCBLK for the function wanted */
    /* 1. Set the length of the function block */
    /* 2. Set the function value to 1 indicating INITIALIZATION */
    /* ----- */
    FUNCBLK.zfcbll=sizeof(fcb);
    FUNCBLK.zfcbfunc=1;

```

```

/* ----- */
/* Make the call to FXXZC to initialize the ENVIRONMENT and */
/* check for any errors after the call. The fourth parameter to */
/* FXXZC must be changed to the APPLICATION NAME. This */
/* determines which activity log the transactions and/or error */
/* messages are logged to. */
/* ----- */
printf("INITIALIZE - SET UP DataInterchange ENVIRONMENT\n");
fxxzc(&SNB,CCBptr,&FUNCBLK,"APPLNAME","SYSID");
return check_ierr(CCBptr);
}
/* ----- */
/* A subroutine to terminate DataInterchange */
/* ----- */
int
DInterm(CCBptr)
    ccb *CCBptr;          /* Pointer to Common Control Block */
{
    snb  SNB;              /* Local snb */
    fcb  FUNCBLK;          /* Local fcb */

    /* ----- */
    /* Initialize the SNB for environmental services */
    /* 1. Clear it */
    /* 2. Set SNB length */
    /* 3. Set the number of parameters that will be passed */
    /* 4. Set the name of the SERVICE being called */
    /* ----- */
    memset(&SNB,'\0',sizeof(SNB));
    SNB.zsnbll=sizeof(SNB);
    SNB.zsnbpc=3;
    memcpy(SNB.zsnbname,"ENVSERV ",8);

    /* ----- */
    /* Initialize the FUNCBLK for the function wanted */
    /* 1. Set the length of the function block */
    /* 2. Set the function value to 2 indicating TERMINATION */
    /* ----- */
    FUNCBLK.zfcbll=sizeof(fcb);
    FUNCBLK.zfcbfunc=2;

    /* ----- */
    /* Make the call to FXXZC to terminate the ENVIRONMENT and */
    /* check for any errors after the call. */
    /* ----- */
    printf("TERMINATE the DataInterchange ENVIRONMENT\n");
    fxxzc(&SNB,CCBptr,&FUNCBLK);
    return check_terr(CCBptr);
}

/* ----- */
/* A subroutine to check the results of a DataInterchange terminate */
/* ----- */

```

```

    int
check_terr(CCBptr)
    ccb *CCBptr;          /* Pointer to CCB control block      */
{
    int    lrc;           /* Local return value      */

    /* ----- */
    /* Check Return Codes received from TERMINATION      */
    /* return 1 if there are errors                        */
    /* return 0 if there are NO errors                    */
    /* ----- */
    lrc=1;
    switch ((int)CCBptr->zccbrc) {

        case 0:
            /* ----- */
            /* ZERO indicates termination was successful    */
            /* ----- */
            lrc=0;
            printf("DATAINTERCHANGE TERMINATION SUCCESSFUL\n");
            break;

        default:
            /* ----- */
            /* Invalid return code. Display the return codes and */
            /* terminate processing                               */
            /* ----- */
            printf("INVALID RETURN CODE FROM DataInterchange\n");
            printf("RETURN CODE = %ld, EXTENDED RETURN CODE = %ld\n",
                CCBptr->zccbrc, CCBptr->zccberc);
            break;
    }
    return lrc;
}
/* ----- */
/* A subroutine to check the results of a DataInterchange initialize */
/* ----- */
int
check_ierr(CCBptr)
    ccb *CCBptr;          /* Pointer to CCB control block      */
{
    int    lrc;           /* Local return value      */

    /* ----- */
    /* Check Return Codes received from Initialization      */
    /* return 1 if there are errors                        */
    /* return 0 if there are NO errors                    */
    /* ----- */
    lrc=1;
    switch ((int)CCBptr->zccbrc) {

        case 0:
            /* ----- */
            /* ZERO indicates initialization was successful and */
            /* regular processing can continue                  */
            /* ----- */

```

```

        lrc=0;
        printf("DATAINTERCHANGE INITIALIZATION SUCCESSFUL\n");
        break;

case 4:
    /* ----- */
    /* FOUR indicates initialization was not successful. */
    /* Display the return code and extended return code and */
    /* terminate processing. */
    /* ----- */
    printf("DATAINTERCHANGE INITIALIZATION FAILED\n");
    printf("RETURN CODE = %ld, EXTENDED RETURN CODE = %ld\n",
          CCBptr->zccbrc,CCBptr->zccberc);
    break;

default:
    /* ----- */
    /* Invalid return code. Display the return codes and */
    /* terminate processing */
    /* ----- */
    printf("INVALID RETURN CODE FROM DataInterchange\n");
    printf("RETURN CODE = %ld, EXTENDED RETURN CODE = %ld\n",
          CCBptr->zccbrc,CCBptr->zccberc);
    break;
}
return lrc;
}

```

Query the Transaction Store using COBOL

The following sample COBOL program illustrates how to initialize the DataInterchange Utility data block and LINK to EDIFFUT to query the Transaction Store. For more information on the FFUS definition, see "COBOL Utility Control Information Block" on page C-12.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. EDICOBU.

```

```

*****
*
* This COBOL Command Level CICS program illustrates how
* to invoke the DataInterchange Transaction Store Utility
* by initializing the Utility data block and LINKing to
* program EDIFFUT. The Utility is called to PERFORM a
* QUERY of the Transaction Store and the results will be
* written to a TS Queue named EDIQUERY.
*
*****

```

```

ENVIRONMENT DIVISION.

```

```

DATA DIVISION.

```

```

WORKING-STORAGE SECTION.

```

```

COPY EDIFFDU.

```

```

01  CMD-STRING          PIC X(41)  VALUE
    'PERFORM QUERY WHERE HANDLE(1992) TO(1993)'.

01  WS-MSG.
    05  FILLER  PIC X(16) VALUE 'SEVERITY CODE = '.
    05  WS-SEV  PIC 9(04).
    05  FILLER  PIC X(28) VALUE ' - UTILITY CONDITION CODE = '.
    05  WS-CC   PIC 9(05).

```

LINKAGE SECTION.

```

01  DFHCOMMAREA        PIC X(01).

01  LINKAGE-POINTERS.
    05  FILLER          PIC S9(08) COMP.
    05  CMD-POINTER     PIC S9(08) COMP.

01  CMD-AREA           PIC X(41).

```

PROCEDURE DIVISION.

```

EXEC CICS
  GETMAIN SET (CMD-POINTER) LENGTH (41)
END-EXEC.

```

```

MOVE CMD-STRING TO CMD-AREA.

```

```

MOVE LOW-VALUES TO FFUS-DATA.
MOVE CMD-POINTER TO FFUS-CMDP.
MOVE 41          TO FFUS-CMDLEN.
MOVE 'EDIQUERY'  TO FFUS-QRYNAME.
MOVE 'TS'        TO FFUS-QRYTYPE.

```

```

EXEC CICS LINK
  PROGRAM ('EDIFFUT')
  COMMAREA (FFUS-DATA)
  LENGTH (248)
END-EXEC.

```

```

MOVE FFUS-CCBRC TO WS-SEV.
MOVE FFUS-CCBERC TO WS-CC.

```

```

EXEC CICS
  SEND FROM (WS-MSG) LENGTH (53) ERASE
END-EXEC.

```

```

EXEC CICS
  RETURN
END-EXEC.

```

```

STOP RUN.

```

Query the Transaction Store in PL/I

The following sample PL/I program illustrates how to initialize the DataInterchange Utility data block and LINK to EDIFFUT to query the Transaction Store. For more information on the FFUS definition, see “COBOL Utility Control Information Block” on page C-12.

```
EDIPLIU: PROC OPTIONS(MAIN,REENTRANT) REORDER;

/*****
/*
/* This PLI Command Level CICS program illustrates how
/* to invoke the DataInterchange Transaction Store Utility
/* by initializing the Utility data block and LINKing to
/* program EDIFFUT. The Utility is called to PERFORM a
/* QUERY of the Transaction Store and the results will be
/* written to a TS Queue named EDIQUERY.
/*
*****/

DCL LOW          BUILTIN;

%INCLUDE EDIFFDU;

DCL CMD_STRING    CHAR(41) INIT
    ('PERFORM QUERY WHERE HANDLE(1992) TO(1993)');

DCL ADDR          BUILTIN;

DCL 1 MSG,
    2 FILL01      CHAR(16) INIT ('SEVERITY CODE = '),
    2 SEV         PIC '9999',
    2 FILL02      CHAR(28) INIT (' - UTILITY CONDITION CODE = '),
    2 CONDCODE    PIC '99999';

FFUS = '';
FFUS.CMDP = ADDR(CMD_STRING);
FFUS.CMDLEN = 41;
FFUS.QRYNAME = 'EDIQUERY';
FFUS.QRYTYPE = 'TS';

EXEC CICS LINK PROGRAM ('EDIFFUT') COMMAREA (FFUS) LENGTH (248);

MSG.SEV = FFUS.CCBRC;
MSG.CONDCODE = FFUS.CCBERC;

EXEC CICS SEND FROM (MSG) LENGTH (53) ERASE;

EXEC CICS RETURN;

END EDIPLIU;
```

Translate and Queue for Send

The following C program translates your application data. It illustrates function code 0131, which queues the transaction data to a network after translation. The transactions remain in the queue until sending is requested. For more information on the SNB definition, see "C SNB" on page C-25, for the CCB definition, see "C CCB" on page C-25, for the FCB definition, see "C FCB" on page C-26, for the TRCB definition, see "C TRCB" on page C-27, and for the TRIDB and TRODB definitions, see "C DATBLK" on page C-34.

```
/* ----- */
/* Include the definition files */
/* ----- */
#include "stdio.h"          /* C I/O definitions */
#include "disnb.h"          /* SNB definition */
#include "diccb.h"          /* CCB definition */
#include "difcb.h"          /* FCB definition */
#include "ditrcb.h"         /* TRCB definition */
#include "didblk.h"         /* TRIDB and TRODB definitions */

/* ----- */
/* Prototype internal functions */
/* ----- */
static int check_error(ccb*);

int
main()
{
    snb   TRsnb;             /* SNB for translator */
    ccb   DIccb;             /* DataInterchange common block */
    fcb   TRfcb;             /* Translator function block */
    TRcb   TPCB;             /* Translator control block */
    DATAblk *TPIDB,*TPODB;  /* Pointers to data blocks */

    /* ----- */
    /* Call a routine to Initialize DataInterchange */
    /* ----- */
    if (!DIinit(&DIccb)) {

        /* ----- */
        /* Initialize was successful so continue processing */
        /* ----- */

        /* ----- */
        /* Prepare the translator SNB */
        /* 1. Initialize to zeros */
        /* 2. Set the SNB length */
        /* 3. Set the number parameters passed to the translator */
        /* 4. Set the name of the translator service (TRANPROC) */
        /* ----- */
        memset(&TRsnb,'\0',sizeof(TRsnb));
        TRsnb.zsnbll=sizeof(snb);
        TRsnb.zsnbpc=6;
        memcpy(TRsnb.zsnbname,"TRANPROC",8);

        /* ----- */
        /* Prepare the translator FCB */
        /* 1. Set the FCB length */
        /* ----- */
    }
}
```

```

/* 2. Set the function for PRODUCTION SEND TRANSLATE (131) */
/* ----- */
TRfcb.zfcbll=sizeof(fcb);
TRfcb.zfcbfunc=131;

/* ----- */
/* Prepare the translator control block */
/* ----- */
memset(&TPCB,' ',sizeof(TPCB));
TPCB.blklen = sizeof(TPCB);
memcpy(TPCB.blkme,"EDITRCB ",8);

/* ----- */
/* set ATFID to data format ID that was defined using the */
/* Application Data Format screens. This format describes the */
/* structure of your Application data. */
/* ----- */
memcpy(TPCB.atfid,"POSEND",6);

/* ----- */
/* Set the internal trading partner ID for this trading partner */
/* that was defined using Trading Partner transaction screens */
/* ----- */
memcpy(TPCB.intpid,"XYZCOMPANYINTPID",16);

/* ----- */
/* Prepare the INPUT and OUTPUT data blocks */
/* ----- */
TPIDB = (DATAblk*) malloc(32767);
TPIDB->blklen = 32767;
memcpy(TPIDB->blkme,"EDITRIN ",8);

/* MOVE APPLICATION DATA TO TPIDB.data */
/* MOVE APPLICATION DATA length to TPIDB.datlen */

TPODB = (DATAblk*) malloc(32767);
TPODB->blklen = 32767;
memcpy(TPODB->blkme,"EDITROUT",8);

/* ----- */
/* Call the translator to TRANSLATE application data */
/* ----- */
printf("TRANSLATE - TRANSLATE AND QUEUE TO SEND\n");
fxzc(&TRsnb,&DIccb,&TRfcb,&TPCB,TPIDB,TPODB);
if (!check_error(&DIccb)) {
    /* ----- */
    /* There were no errors, call translator with a TERMINATION */
    /* request to finish building and to queue the transaction */
    /* data */
    /* ----- */
    TRterm(&TRsnb,&DIccb,&TRfcb,&TPCB,TPIDB,TPODB);
}
/* ----- */
/* Terminate Data Interchange */
/* ----- */
DIterm(&DIccb);
}

```

```

}
static int
check_error(CCBptr)
    ccb *CCBptr;          /* Pointer to common block */
{
    int lrc;               /* return code */
    /* ----- */
    /* Check the return codes from a SEND TRANSLATE request */
    /* ----- */
    lrc=0;
    switch ((int)CCBptr->zccbrc) {
        case 0:
            /* ----- */
            /* Return code is zero, the translator was successful */
            /* ----- */
            printf("Translation was successful\n");
            break;

        case 8:
            /* ----- */
            /* If the return code is EIGHT and the Extended Return Code */
            /* is from ONE to SIX, this indicates a Translation Error. */
            /* If the Extended Return Code is greater than or equal to */
            /* TEN, a non-translation error has occurred and */
            /* translator will have terminated itself automatically */
            /* ----- */
            switch ((int)CCBptr->zccberc) {
                case 1:          /* data element error */
                    printf("Data element level error occurred\n");
                    break;
                case 2:          /* segment level error */
                    printf("Segment level occurred\n");
                    break;
                case 3:          /* transaction level error */
                case 4:          /* function group level error */
                case 5:          /* envelope level error */
                case 6:          /* invalid data in input file */
                    printf("Unexpected error during send processing\n");
                    printf("Return code = %ld, extended return code = %ld",
                        CCBptr->zccbrc, CCBptr->zccberc);
                    break;
                default:
                    lrc = 1;
                    printf("Non translation error occurred\n");
                    printf("Return code = %ld, extended return code = %ld",
                        CCBptr->zccbrc, CCBptr->zccberc);
                    break;
            }
        case 12:
            /* ----- */
            /* A severe error occurred */
            /* ----- */
            lrc = 1;
            printf("A severe error occurred in translation\n");
            printf("Return code = %ld, extended return code = %ld",
                CCBptr->zccbrc, CCBptr->zccberc);
            break;
    }
}

```

```

    default:
        /* ----- */
        /* An invalid return code was returned */
        /* ----- */
        lrc = 1;
        printf("An invalid return code from the translator\n");
        printf("Return code = %ld, extended return code = %ld",
            CCBptr->zccbrc, CCBptr->zccberc);
        break;
}

return lrc;
}

```

Send Queued Data

The following C program calls communications to send previously translated transactions to a network. For more information on the SNB definition, see “C SNB” on page C-25, for the CCB definition, see “C CCB” on page C-25, for the FCB definition, see “C FCB” on page C-26, for the CMCB definition, see “C CMCB” on page C-26, and for the TPPDB definitions, see “C TPPDB” on page C-31.

```

/* ----- */
/* Get the control block definitions */
/* ----- */
#include "stdio.h"           /* C I/O library */
#include "disnb.h"           /* SNB definition */
#include "diccb.h"           /* CCB definition */
#include "difcb.h"           /* FCB definition */
#include "dicmcb.h"          /* TRCB definition */
#include "ditpdb.h"          /* TPPDB definition */

/* ----- */
/* Provide function prototypes */
/* ----- */
static int check_error(ccb*);

int
main()
{
    ccb  DIccb;      /* Common Control Block */
    snb  CMsnb;      /* SNB for communications */
    fcb  CMfcb;      /* FCB for communications */
    CMcb CMCB;       /* Communication control block */
    TPPdb CMTTPDB;   /* Trading Partner Data Block */

    /* ----- */
    /* Call function for DataInterchange initialization */
    /* ----- */
    if (DIinit(&DIccb))
        return 1;

    /* ----- */
    /* Initialize the SNB for communications support */
    /* 1. Initialize the block to zeros */
    /* 2. Set block length */
    /* ----- */
}

```

```

/* 3. Set the number of parameters to communications */
/* 4. Set the name of the communications service (COMM) */
/* ----- */
memset(&CMsnb,'\0',sizeof(snb));
CMsnb.zsnbll=sizeof(snb);
CMsnb.zsnbpc=6;
memcpy(CMsnb.zsnbname,"COMM      ",8);

/* ----- */
/* Initialize the function block for communications */
/* 1. Set the length of the block */
/* 2. Set the function to SEND TRANSACTION DATA */
/* ----- */
CMfcb.zfcbll=sizeof(fcb);
CMfcb.zfcbfunc=211;

/* ----- */
/* Initialize the control block for communications */
/* ----- */
memset(&CMCB,' ',sizeof(CMCB));
CMCB.blklen=sizeof(CMCB);
memcpy(CMCB.blkme,"EDICMCB",7);

/* ----- */
/* Set the network operation (netop) to indicate that we want to */
/* send X12 data (SENDX12) */
/* ----- */
memcpy(CMCB.netop,"SENDX12",7);

/* ----- */
/* Set the requestor id (reqid) equal to the member ID of an */
/* entry in the REQUESTOR profile. */
/* ----- */
memcpy(CMCB.reqid,"XYZCOMPANYPOSEND",16);

/* ----- */
/* NULLS in the ename field indicates the message user class from */
/* the REQUESTOR profile will be used */
/* ----- */
memset(CMCB.ename,'\0',sizeof(CMCB.ename));

/* ----- */
/* Initialize the trading partner Data block */
/* ----- */
memset(&CMTPPDB,' ',sizeof(CMTPPDB));
CMTPPDB.blklen=sizeof(CMTPPDB);
memcpy(CMTPPDB.blkme,"EDITPPDB",8);

/* ----- */
/* Issue the call to send the X12 data */
/* ----- */
printf("SEND TRANSACTION DATA\n");
fxxzc(&CMsnb,&DIccb,&CMfcb,&CMCB,&CMTPPDB,(void*)0);
if (check_error(&DIccb)) {
/* ----- */
/* Add code here to process a failed SEND request */
/* ----- */
}

```

```

    }
    /* ----- */
    /* Terminate DataInterchange */
    /* ----- */
    DItem(&DIccb);
    return 0;
}
static int
check_error(CCBptr)
    ccb *CCBptr; /* Pointer to the common block */
{
    int lrc; /* return code */

    lrc=1;

    /* ----- */
    /* Process according to the return code in the CCB */
    /* ----- */
    switch ((int)CCBptr->zccbrc) {
        case 0:
            /* ----- */
            /* Data was successfully sent */
            /* ----- */
            printf("Data successfully sent to the network\n");
            lrc=0;
            break;

        case 4:
            /* ----- */
            /* Warning from communications */
            /* ----- */
            printf("A warning was received when SENDING data\n");
            printf("Return code = %ld, extended return code = %ld\n",
                CCBptr->zccbrc, CCBptr->zccberc);
            lrc=0;
            break;

        case 8:
        case 12:
            /* ----- */
            /* An error was returned by communications */
            /* ----- */
            printf("An error was received when SENDING data\n");
            printf("Return code = %ld, extended return code = %ld\n",
                CCBptr->zccbrc, CCBptr->zccberc);
            break;

        default:
            /* ----- */
            /* An invalid return code from communications */
            /* ----- */
            printf("An invalid return code when SENDING data\n");
            printf("Return code = %ld, extended return code = %ld\n",
                CCBptr->zccbrc, CCBptr->zccberc);
    }
}

```

```

        break;
    }
    return lrc;
}

```

End Translation

The following C program ends data translation. After determining that your transaction translated correctly or with minor errors, the program then ends the translation. If the translator encounters a nontranslation error then the translator stops on its own and a termination call should not be made. This sample application sets the function code to 1000, which tells the translator that translation is complete. If DataInterchange is building an interchange when this request is received, DataInterchange finishes the interchange and writes it to the transaction data queue.

This program is an example of a subroutine called from another program. For more information on the SNB definition, see "C SNB" on page C-25, for the CCB definition, see "C CCB" on page C-25, for the FCB definition, see "C FCB" on page C-26, for the TRCB definition, see "C TRCB" on page C-27, and for the TRIDB and TRODB definitions, see "C DATBLK" on page C-34.

```

/* ----- */
/* Get control block definitions */
/* ----- */
#include "stdio.h"           /* C I/O library */
#include "disnb.h"           /* SNB definition */
#include "diccb.h"           /* CCB definition */
#include "difcb.h"           /* FCB definition */
#include "ditrcb.h"          /* TRCB definition */
#include "didblk.h"          /* TRIDB and TRODB definitions */

/* ----- */
/* Prototype for internal function */
/* ----- */
static void check_error(ccb*);

int
TRterm(SNBptr,CCBptr,TPCBptr,TPIDB,TPODB)
    snb *SNBptr;           /* Pointer to SNB for translator */
    ccb *CCBptr;           /* Pointer to common block */
    TRcb *TPCBptr;         /* Pointer to translator control block */
    DATAblk *TPIDB;       /* Pointer to input data block */
    DATAblk *TPODB;       /* Pointer to output data block */
{
    fcb TRfcb;             /* function block for translator termination */

    /* ----- */
    /* Set up the function block for TERMINATION of the translator */
    /* ----- */
    TRfcb.zfcbll = sizeof(fcb);
    TRfcb.zfcbfunc = 1000;

    /* ----- */
    /* Make the call to terminate translation */
    /* ----- */
    printf("END_TRANSLATOR - Terminate call to translator\n");
    fxxzc(SNBptr,CCBptr,&TRfcb,TPCBptr,TPIDB,TPODB);
}

```



```

    check_error(CCBptr);
}
static void
check_error(CCBptr)
    ccb *CCBptr;          /* Pointer to common block          */
{
    switch ((int)CCBptr->zccbrc) {
        case 0:
            /* ----- */
            /* Termination of the translator was successful */
            /* ----- */
            printf("TRANSACTION PROCESSOR TERMINATED SUCCESSFULLY\n");
            break;

        case 8:
        case 12:
            /* ----- */
            /* Termination failed */
            /* ----- */
            printf("TRANSACTION PROCESSOR TERMINATION FAILED\n");
            printf("Return code = %ld , extended return code = %ld\n",
                CCBptr->zccbrc, CCBptr->zccbrc);
            break;

        default:
            /* ----- */
            /* Invalid return code */
            /* ----- */
            printf("Invalid return code from TERMINATION call\n");
            printf("Return code = %ld , extended return code = %ld\n",
                CCBptr->zccbrc, CCBptr->zccbrc);
            break;
    }
}

```

Receive From the Network

The following C program calls communications to receive data.

After this code, you can add the code that translates the standard data format to your application's data format. For more information on the SNB definition, see "C SNB" on page C-25, for the CCB definition, see "C CCB" on page C-25, for the FCB definition, see "C FCB" on page C-26, for the CMCB definition, see "C CMCB" on page C-26, and for the TPPDB definitions, see "C TPPDB" on page C-31.

```

/* ----- */
/* Get the control block definitions */
/* ----- */
#include "stdio.h"          /* C I/O library */
#include "disnb.h"          /* SNB definition */
#include "diccb.h"          /* CCB definition */
#include "difcb.h"          /* FCB definition */
#include "dicmcb.h"         /* TRCB definition */
#include "ditpdb.h"         /* TPPDB definition */

```

```

/* ----- */
/* Prologues for internal functions */
/* ----- */
static int check_error(ccb*);

int
main()
{
    ccb  DIccb;      /* Common Control Block */
    snb  CMsnb;      /* SNB for communications */
    fcb  CMfcb;      /* FCB for communications */
    CMcb  CMCB;      /* Communication control block */
    TPPdb CMTPPDB;   /* Trading Partner Data Block */

    /* ----- */
    /* Call function for DataInterchange initialization */
    /* ----- */
    if (DIinit(&DIccb))
        return 1;

    /* ----- */
    /* Initialize the SNB for communications support */
    /* 1. Initialize the block to zeros */
    /* 2. Set block length */
    /* 3. Set the number of parameters to communications */
    /* 4. Set the name of the communications service (COMM) */
    /* ----- */
    memset(&CMsnb,'\0',sizeof(snb));
    CMsnb.zsnbll=sizeof(snb);
    CMsnb.zsnbpc=6;
    memcpy(CMsnb.zsnbname,"COMM",8);

    /* ----- */
    /* Initialize the function block for communications */
    /* 1. Set the length of the block */
    /* 2. Set the function to RECEIVE */
    /* ----- */
    CMfcb.zfcbll=sizeof(fcb);
    CMfcb.zfcbfunc=232;

    /* ----- */
    /* Initialize the control block for communications */
    /* ----- */
    memset(&CMCB,' ',sizeof(CMCB));
    CMCB.blklen=sizeof(CMCB);
    memcpy(CMCB.blkname,"EDICMCB",7);

    /* ----- */
    /* Set the network operation (netop) to indicate that we want to */
    /* receive X12 data (RCVX12) */
    /* ----- */
    memcpy(CMCB.netop,"RCVX12",7);

    /* ----- */
    /* Set the requestor id (reqid) equal to the member ID of an */
    /* entry in the REQUESTOR profile. */
    /* ----- */

```

```

memcpy(CMCB.reqid,"XYZCOMPANYPODEPT",16);

/* ----- */
/* acctyp of "D" means to receive for this trading partner only */
/* ----- */
*CMCB.acctyp='D';

/* ----- */
/* datatype of "D" indicates a DDNAME is used rather than DSNAME */
/* ----- */
*CMCB.datatyp='D';

/* ----- */
/* NULLS in the ename field indicates the message user class from */
/* the REQUESTOR profile will be used */
/* ----- */
memset(CMCB.ename,'\0',sizeof(CMCB.ename));

/* ----- */
/* Set record length indicator (RESRECL) to indicate that the */
/* lrecl of the output file should be used as the record length */
/* ----- */
*CMCB.resrecl = 'S';

/* ----- */
/* Initialize the trading partner Data block */
/* ----- */
memset(&CMTPPDB,' ',sizeof(CMTPPDB));
CMTPPDB.blklen=sizeof(CMTPPDB);
memcpy(CMTPPDB.blkname,"EDITPPDB",8);

/* ----- */
/* Issue the call to receive the X12 data */
/* ----- */
printf("RECEIVE IMMEDIATE\n");
fxxzc(&CMSnb,&DIccb,&CMfcb,&CMCB,&CMTPPDB,(void*)0);
if (!check_error(&DIccb)) {
    /* ----- */
    /* Add code here to call translator to processed received data */
    /* ----- */
}
/* ----- */
/* Terminate DataInterchange */
/* ----- */
DIterm(&DIccb);
return 0;
}

static int
check_error(CCBptr)
    ccb    *CCBptr;    /* Pointer to the common block */
{
    int    lrc;        /* return code */

    lrc=1;

    /* ----- */
    /* Process according to the return code in the CCB */
    /* ----- */

```

```

/* ----- */
switch ((int)CCBptr->zccbrc) {
    case 0:
        /* ----- */
        /* Data was successfully received */
        /* ----- */
        printf("Data successfully received from the network\n");
        lrc=0;
        break;

    case 4:
        /* ----- */
        /* Warning from communications */
        /* ----- */
        printf("A warning was received when RECEIVING data\n");
        printf("Return code = %ld, extended return code = %ld\n",
            CCBptr->zccbrc,CCBptr->zccberc);
        lrc=0;
        break;

    case 8:
    case 12:
        /* ----- */
        /* An error was returned by communications */
        /* ----- */
        printf("An error was received when RECEIVING data\n");
        printf("Return code = %ld, extended return code = %ld\n",
            CCBptr->zccbrc,CCBptr->zccberc);
        break;

    default:
        /* ----- */
        /* An invalid return code from communications */
        /* ----- */
        printf("An invalid return code when RECEIVING data\n");
        printf("Return code = %ld, extended return code = %ld\n",
            CCBptr->zccbrc,CCBptr->zccberc);
        break;
}
return lrc;
}

```

Translate Received Data

The following C program calls the translator to receive data, then translates the data to application data format. For more information on the SNB definition, see "C SNB" on page C-25, for the CCB definition, see "C CCB" on page C-25, for the FCB definition, see "C FCB" on page C-26, for the TRCB definition, see "C TRCB" on page C-27, and for the TRIDB and TRODB definitions, see "C DATBLK" on page C-34.

```

/* ----- */
/* Include the definition files */
/* ----- */
#include "stdio.h"           /* C I/O definitions */
#include "disnb.h"           /* SNB definition */
#include "diccb.h"           /* CCB definition */

```

```

#include "difcb.h"           /* FCB definition */
#include "ditrcb.h"          /* TRCB definition */
#include "didblk.h"          /* TRIDB and TRODB definitions */

/* ----- */
/* prototypes */
/* ----- */
static int check_error(ccb*);

int
main()
{
    snb    TRsnb;           /* SNB for translator */
    ccb    DIccb;           /* DataInterchange common block */
    fcb    TRfcb;           /* Translator function block */
    TRcb    TPCB;           /* Translator control block */
    DATAblk *TPIDB,*TPODB; /* Pointers to data blocks */

    /* ----- */
    /* Call a routine to Initialize DataInterchange */
    /* ----- */
    if (!DIinit(&DIccb)) {

        /* ----- */
        /* Initialize was successful so continue processing */
        /* ----- */

        /* ----- */
        /* Prepare the translator SNB */
        /* 1. Initialize to zeros */
        /* 2. Set the SNB length */
        /* 3. Set the number parameters passed to the translator */
        /* 4. Set the name of the translator service (TRANPROC) */
        /* ----- */
        memset(&TRsnb,'\0',sizeof(TRsnb));
        TRsnb.zsnbll=sizeof(snb);
        TRsnb.zsnbpc=6;
        memcpy(TRsnb.zsnbname,"TRANPROC",8);

        /* ----- */
        /* Prepare the translator FCB */
        /* 1. Set the FCB length */
        /* 2. Set the function for PRODUCTION SEND TRANSLATE (131) */
        /* ----- */
        TRfcb.zfcbll=sizeof(fcb);
        TRfcb.zfcbfunc=212;

        /* ----- */
        /* Prepare the translator control block */
        /* ----- */
        memset(&TPCB,' ',sizeof(TPCB));
        TPCB.blklen = sizeof(TPCB);
        memcpy(TPCB.blkname,"EDITRCB ",8);

        /* ----- */
        /* set REQID to REQUESTOR profile that was defined using */
        /* profile screens. The requestor profile contains the */

```

```

/* DDNAME of the file that was received and is to be translated.*/
/* ----- */
memcpy(TPCB.reqid,"XYZCOMPANYPODEPT",16);

/* ----- */
/* Prepare the INPUT and OUTPUT data blocks */
/* ----- */
TPIDB = (DATAblk*) malloc(32767);
TPIDB->blklen = 32767;
memcpy(TPIDB->blkme,"EDITRIN ",8);

TPODB = (DATAblk*) malloc(32767);
TPODB->blklen = 32767;
memcpy(TPODB->blkme,"EDITROUT",8);

/* ----- */
/* Call the translator to TRANSLATE STANDARD data to APPL format*/
/* ----- */
for (;!Diccb.zccbrc;) {
    printf("TRANSLATE STANDARD DATA to APPLICATION FORMAT\n");
    fxxzc(&Trsnb,&Diccb,&Trfcb,&TPCB,TPIDB,TPODB);
    if (!check_error(&Diccb)) {
        /* ----- */
        /* Add code here to process the application data */
        /* placed in TRODB by the translator. */
        /* ----- */
    }
}
/* ----- */
/* Terminate DataInterchange */
/* ----- */
DIterm(&Diccb);
}
}

static int
check_error(CCBptr)
    ccb *CCBptr;          /* Pointer to common block */
{
    int lrc;              /* return code */
    /* ----- */
    /* Check the return codes from a RECEIVE TRANSLATE request */
    /* ----- */
    lrc=0;
    switch ((int)CCBptr->zccbrc) {
        case 0:
            /* ----- */
            /* Return code is zero, the translator was successful */
            /* ----- */
            printf("Translation was successful\n");
            break;

        case 8:
            /* ----- */
            /* If the return code is EIGHT and the Extended Return Code*/
            /* is from ONE to SIX, this indicates a Translation Error. */
            /* If the Extended Return Code is greater than or equal to */
            /* TEN, a non-translation error has occurred and */
            /* ----- */

```

```

/* translator will have terminated itself automatically */
/* ----- */
switch ((int)CCBptr->zccberc) {
    case 1: /* data element error */
        printf("Data element level error occurred\n");
        CCBptr->zccbrc=CCBptr->zccberc=0;
        break;
    case 2: /* segment level error */
        printf("Segment level occurred\n");
        CCBptr->zccbrc=CCBptr->zccberc=0;
        break;
    case 3: /* transaction level error */
    case 4: /* function group level error */
    case 5: /* envelope level error */
    case 6: /* invalid data in input file */
        lrc=1;
        printf("Error in Envelope format or content\n");
        printf("Return code = %ld, extended return code = %ld",
            CCBptr->zccbrc,CCBptr->zccberc);
        CCBptr->zccbrc=CCBptr->zccberc=0;
        break;
    default:
        lrc = 1;
        printf("Non translation error occurred\n");
        printf("Return code = %ld, extended return code = %ld",
            CCBptr->zccbrc,CCBptr->zccberc);
        break;
}
case 12:
/* ----- */
/* A severe error occurred */
/* ----- */
lrc = 1;
printf("A severe error occurred in translation\n");
printf("Return code = %ld, extended return code = %ld",
    CCBptr->zccbrc,CCBptr->zccberc);
break;

default:
/* ----- */
/* An invalid return code was returned */
/* ----- */
lrc = 1;
printf("An invalid return code from the translator\n");
printf("Return code = %ld, extended return code = %ld",
    CCBptr->zccbrc,CCBptr->zccberc);
break;
}

return lrc;
}

```

End DataInterchange

When you are finished using DataInterchange services, you must end your DataInterchange session. The following C program ends a DataInterchange session. It also illustrates a function call from an application program. For more information on the SNB definition, see "C SNB" on page C-25, for the CCB definition, see "COBOL CCB" on page C-2, and for the FCB definition, see "C FCB" on page C-26.

```
/* ----- */
/* Include the definition files                                */
/* ----- */
#include "stdio.h"          /* C I/O definitions          */
#include "disnb.h"          /* SNB definition            */
#include "diccb.h"          /* CCB definition            */
#include "difcb.h"          /* FCB definition            */

int
DIterm(CCBptr)
    ccb *CCBptr;           /* Pointer to Common Control Block */
{
    snb SNB;               /* Local snb                  */
    fcb FUNCBLK;           /* Local fcb                  */

    /* ----- */
    /* Initialize the SNB for environmental services          */
    /* 1. Clear it                                           */
    /* 2. Set SNB length                                     */
    /* 3. Set the number of parameters that will be passed  */
    /* 4. Set the name of the SERVICE being called           */
    /* ----- */
    memset(&SNB, '\0', sizeof(SNB));
    SNB.zsnbll = sizeof(SNB);
    SNB.zsnbpc = 3;
    memcpy(SNB.zsnbname, "ENVSERV ", 8);

    /* ----- */
    /* Initialize the FUNCBLK for the function wanted        */
    /* 1. Set the length of the function block               */
    /* 2. Set the function value to 2 indicating TERMINATION */
    /* ----- */
    FUNCBLK.zfcbll = sizeof(fcb);
    FUNCBLK.zfcbfunc = 2;

    /* ----- */
    /* Make the call to FXXZC to terminate the ENVIRONMENT and */
    /* check for any errors after the call.                     */
    /* ----- */
    printf("TERMINATE the DataInterchange ENVIRONMENT\n");
    fxxzc(&SNB, CCBptr, &FUNCBLK);
    return check_error(CCBptr);
}
/*
int
check_error(CCBptr)
    ccb *CCBptr;           /* Pointer to CCB control block */
{
    int lrc;               /* Local return value          */

```



```

/* ----- */
/* Check Return Codes received from TERMINATION */
/* return 1 if there are errors */
/* return 0 if there are NO errors */
/* ----- */
lrc=1;
switch ((int)CCBptr->zccbrc) {

    case 0:
        /* ----- */
        /* ZERO indicates termination was successful */
        /* ----- */
        lrc=0;
        printf("DATAINTERCHANGE TERMINATION SUCCESSFUL\n");
        break;

    default:
        /* ----- */
        /* Invalid return code. Display the return codes and */
        /* terminate processing */
        /* ----- */
        printf("INVALID RETURN CODE FROM DataInterchange\n");
        printf("RETURN CODE = %ld, EXTENDED RETURN CODE = %ld\n",
              CCBptr->zccbrc,CCBptr->zccberc);
        break;
}
return lrc;
}

```

Data Extract Report Generator (EDISAMR1)

The following routine reads in a flat file that was created using the Perform Data Extract command in a JCL file. It then reads the records in and sorts them by the Application Control ID and writes them back out to the same flat file. It then reads them in and writes them back out to the specified device in the defined report format.

In this example 3 records of length 1024 are concatenated together to form one record of length 3072 using the following command in the run JCL file.

```

PERFORM ENVELOPE DATA EXTRACT SELECTING
CONCATENATE(Y) INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y)
WHERE HANDLE(19930201) TO(19930228)

```

Once this command has been executed an unsorted flat file will exist with all the fields specified in IN-RECORD.

```

*****
* Module Name: EDISAMR1                                     *
*                                                         *
* Descriptive Name: Data Extract Report Generator          *
*                                                         *
* Dependencies: None                                       *
*                                                         *
* Restrictions: None                                       *
*                                                         *
* Language: COBOL                                          *
*                                                         *
*****

```

IDENTIFICATION DIVISION.

PROGRAM-ID. EDISAMR1.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

```

        SELECT SORT-FILE ASSIGN TO UT-S-SORTWK.
        SELECT IN-FILE ASSIGN TO UT-S-SQFILE1.
        SELECT OUT-FILE ASSIGN TO UT-S-SQFILE2.

```

DATA DIVISION.

FILE SECTION.

SD SORT-FILE.

```

01 SORT-RECORD.
   05 FILLER                                PIC X(2390).
   05 TRANSACTION-ID                       PIC X(0008).
   05 FILLER                                PIC X(0070).
   05 APPLICATION-ID                       PIC X(0035).
   05 FILLER                                PIC X(0569).

```

FD IN-FILE

```

        RECORDING MODE F
        BLOCK CONTAINS 0 RECORDS
        RECORD CONTAINS 3072 CHARACTERS
        LABEL RECORDS ARE STANDARD.

```

```

*****
* In record definition                                     *
*****

```

```

01 IN-RECORD.
   05 IN-E-REC-ID                          PIC X(003).
   05 IN-E-NICKNAME                        PIC X(016).
   05 IN-E-DIREC-SR                        PIC X(001).
   05 IN-E-INT-CNTRLNUM                     PIC X(014).
   05 IN-E-INT-RCVR-ID                      PIC X(035).

```

```

05 IN-E-FILLER PIC X(072).
05 IN-E-SENDER-ID PIC X(035).
05 IN-E-FAKE-FLAG PIC X(001).
05 IN-E-SEQ-ERROR-FLAG PIC X(001).
05 IN-E-DPLCTE-INT-FLAG PIC X(001).
05 IN-E-TEST-INT-FLAG PIC X(001).
05 IN-E-ENV-DATE PIC X(014).
05 E-FILLER-1 PIC X(002).
05 TRANSMISSION-YEAR PIC X(002).
05 TRANSMISSION-MONTH PIC X(002).
05 TRANSMISSION-DAY PIC X(002).
05 TRANSMISSION-HOUR PIC X(002).
05 TRANSMISSION-MIN PIC X(002).
05 E-FILLER-2 PIC X(002).
05 IN-E-TA1-ACK PIC X(001).
05 IN-E-TA2-DATE PIC X(014).
05 IN-E-NETWORK-STAT PIC X(002).
05 IN-E-NETWORK-STAT-TEXT PIC X(020).
05 IN-E-ACK-EXPECTED PIC X(001).
05 IN-E-ACK-RECEIVED PIC X(001).
05 IN-E-ACK-DATE PIC X(014).
05 IN-E-MES-USER-CLASS PIC X(008).
05 IN-E-MESSAGE-NAME PIC X(008).
05 IN-E-SEQ-NUM PIC X(005).
05 IN-E-MESSAGE-ID PIC X(008).
05 IN-E-PHY-DATA-SET-NAME PIC X(056).
05 IN-E-GROUP-CNT PIC X(011).
05 IN-E-TRAN-CNT PIC X(011).
05 IN-E-SEG-CNT PIC X(011).
05 IN-E-INT-SIZE PIC X(011).
05 IN-E-INT-HEADER PIC X(250).
05 IN-E-INT-TRAILER PIC X(030).
05 IN-E-MRTCPAD1 PIC X(354).

*****
*GROUP RECORD DEFINITION *
*****

05 IN-G-RECORD-ID PIC X(003).
05 IN-G-NICKNAME PIC X(016).
05 IN-G-DIRECTION PIC X(001).
05 IN-G-INT-CNTRL-NUM PIC X(014).
05 IN-G-INT-RECIEVER-ID PIC X(035).
05 IN-G-GROUP-CNTRL-NUM PIC X(014).
05 IN-G-FILLER PIC X(058).
05 IN-G-FAKE-FLAG PIC X(001).
05 IN-G-SENDER-ID PIC X(035).
05 IN-G-RECEIVER-ID PIC X(035).
05 IN-G-ACK-EXPECTED PIC X(001).
05 IN-G-ACK-RECEIVED PIC X(001).
05 IN-G-ACK-DATE PIC X(014).
05 IN-G-ACK-HANDLE PIC X(020).
05 IN-G-TRANSACTION-COUNT PIC X(011).
05 IN-G-SEGMENT-COUNT PIC X(011).
05 IN-G-GROUP-SIZE PIC X(011).
05 IN-G-GROUP-HEADER PIC X(153).
05 IN-G-GROUP-TAILER PIC X(026).
05 IN-G-PADD PIC X(564).

*****

```

*TRANSACTIONS RECORD DEFINITION

*

05	IN-T-REC-ID	PIC X(003).
05	IN-T-NICKNAME	PIC X(016).
05	IN-T-DIRECTION-SR	PIC X(001).
05	IN-T-ENV-CNTRL-NUM	PIC X(014).
05	IN-T-INT-RCVR-ID	PIC X(035).
05	IN-T-GROUP-NUMBER	PIC X(014).
05	IN-T-TRANSACTION-NUMBER	PIC X(014).
05	IN-T-CNTRL-HANDLE	PIC X(020).
05	IN-T-TRAN-HANDLE	PIC X(020).
05	IN-T-FILLER-PAD1	PIC X(004).
05	IN-T-ENV-DATA	PIC X(014).
05	T-BLANKS	PIC X(002).
05	TRANSACTION-YEAR	PIC X(002).
05	TRANSACTION-MONTH	PIC X(002).
05	TRANSACTION-DAY	PIC X(002).
05	TRANSACTION-HOUR	PIC X(002).
05	TRANSACTION-MIN	PIC X(002).
05	T-BLANKS	PIC X(002).
05	IN-T-TRAN-STAT-CODE	PIC X(002).
05	IN-T-TRAN-STAT-TEXT	PIC X(020).
05	IN-T-ACK-RCVED	PIC X(001).
05	IN-T-ACD-RCVED-TEXT	PIC X(020).
05	IN-T-ACK-DATE	PIC X(014).
05	IN-T-TRX-ACK-EXPECTED	PIC X(001).
05	IN-T-TRX-ACK-RCVD	PIC X(001).
05	IN-T-TRX-ACK-RCVED-TEXT	PIC X(020).
05	IN-T-TRX-ACK-DATE	PIC X(014).
05	IN-T-ACK-HANDLE	PIC X(020).
05	IN-T-SGMT-CNT	PIC X(011).
05	IN-T-TRAN-SIZE	PIC X(011).
05	IN-T-ENV-SEG-CNT	PIC X(011).
05	IN-T-ENV-TRAN-SIZE	PIC X(011).
05	IN-T-FORMAT-ID	PIC X(016).
05	IN-T-TRANSACTION-TYPE	PIC X(008).
05	IN-T-FNC-GRP-ID	PIC X(006).
05	IN-T-ENV-TYPE	PIC X(001).
05	IN-T-ENV-MEMBER	PIC X(008).
05	IN-T-NETWORK-ID	PIC X(008).
05	IN-T-TRAN-STAN-ID	PIC X(008).
05	IN-T-STAN-VERSION	PIC X(002).
05	IN-T-TAN-LEVEL	PIC X(002).
05	IN-T-INT-TP-ID	PIC X(035).
05	IN-T-APP-CNTRL-FLD	PIC X(035).
05	IN-T-DLVRY-DATE	PIC X(014).
05	IN-T-EARLIEST-ENV-DATE	PIC X(008).
05	IN-T-EARLIEST-PRG-DATE	PIC X(008).
05	IN-T-TRAN-ERROR-LVL	PIC X(001).
05	IN-T-STORE-STAT	PIC X(002).
05	IN-T-STORE-STAT-TEXT	PIC X(020).
05	IN-T-OVERRIDE-FLAG	PIC X(001).
05	IN-T-HELD-FLAG	PIC X(001).
05	IN-T-TEST-FLAG	PIC X(001).
05	IN-T-DUPLICATE-FLAG	PIC X(001).
05	IN-T-PURGE-FLAG	PIC X(001).
05	IN-T-TRANSLATE-FLAG	PIC X(001).

```

05 IN-T-DETACHED-FLAG      PIC X(001).
05 IN-T-OVERRIDE-HANDLE    PIC X(020).
05 IN-T-SECURITY-PROFILE    PIC X(008).
05 IN-T-ENCRYPTION-KEY      PIC X(016).
05 IN-T-AUTHENTICATION-KEY  PIC X(016).
05 IN-T-APPLICATION-NUMBER  PIC X(014).
05 IN-T-DATA-DELLIMITER     PIC X(001).
05 IN-T-SUB-DELLIMITER      PIC X(001).
05 IN-T-SEGMENT-TERMINATOR  PIC X(001).
05 IN-T-DECIMAL-NOTAION     PIC X(001).
05 IN-T-RELEASE-CHARACTER   PIC X(001).
05 IN-T-SEGMENT-ID-SEP       PIC X(001).
05 IN-T-TRANSACTION-HEADER  PIC X(085).
05 IN-T-TRANSACTION-TRAILER PIC X(026).
05 IN-T-FILLER-PAD3         PIC X(318).

```

```

FD OUT-FILE
  RECORDING MODE F
  BLOCK CONTAINS 0 RECORDS
  RECORD CONTAINS 132 CHARACTERS
  LABEL RECORDS ARE STANDARD.

```

```

01 OUT-RECORD              PIC X(132).

```

WORKING-STORAGE SECTION.

```

*****
* Define report heading records                                     *
*****

```

```

01 REPORT-TITLE.
  05 FILLER              PIC X(053) VALUE SPACES.
  05 FILLER              PIC X(026) VALUE
    'CONTINENTAL BAKING COMPANY'.
  05 FILLER              PIC X(044) VALUE SPACES.
  05 WS-DATE2.
    10 WS-MONTH2         PIC X(002).
    10 FILLER            PIC X(001) VALUE '/'.
    10 WS-DAY2           PIC X(002).
    10 FILLER            PIC X(001) VALUE '/'.
    10 WS-YEAR2          PIC X(002).

```

```

01 REPORT-TITLE-2.
  05 FILLER              PIC X(050) VALUE SPACES.
  05 FILLER              PIC X(033) VALUE
    'OUTSTANDING ACKNOWLEDGMENT REPORT'.
  05 FILLER              PIC X(048) VALUE SPACES.

```

```

*****
* Category report heading record                                   *
*****

```

```

01 HEADING-TITLE.
  05 FILLER              PIC X(057) VALUE
    '*-----TRANSMISSION-----*'.
  05 FILLER              PIC X(074) VALUE SPACES.

```

```

01 COLUMN-HEADING-TOP.
    05 FILLER PIC X(065) VALUE SPACES.
    05 FILLER PIC X(007) VALUE 'TRADING'.
    05 FILLER PIC X(059) VALUE SPACES.

01 COLUMN-HEADING-MIDDLE.
    05 FILLER PIC X(009) VALUE SPACES.
    05 FILLER PIC X(007) VALUE 'CONTROL'.
    05 FILLER PIC X(010) VALUE SPACES.
    05 FILLER PIC X(003) VALUE 'SET'.
    05 FILLER PIC X(036) VALUE SPACES.
    05 FILLER PIC X(007) VALUE 'PARTNER'.
    05 FILLER PIC X(059) VALUE SPACES.

01 COLUMN-HEADING-BOTTOM.
    05 FILLER PIC X(001) VALUE SPACES.
    05 FILLER PIC X(004) VALUE 'TYPE'.
    05 FILLER PIC X(004) VALUE SPACES.
    05 FILLER PIC X(006) VALUE 'NUMBER'.
    05 FILLER PIC X(010) VALUE SPACES.
    05 FILLER PIC X(006) VALUE 'NUMBER'.
    05 FILLER PIC X(008) VALUE SPACES.
    05 FILLER PIC X(004) VALUE 'DATE'.
    05 FILLER PIC X(006) VALUE SPACES.
    05 FILLER PIC X(004) VALUE 'TIME'.
    05 FILLER PIC X(010) VALUE SPACES.
    05 FILLER PIC X(011) VALUE 'DUNS NUMBER'.
    05 FILLER PIC X(013) VALUE SPACES.
    05 FILLER PIC X(015) VALUE 'TRANSACTION KEY'.
    05 FILLER PIC X(013) VALUE SPACES.
    05 FILLER PIC X(006) VALUE 'STATUS'.
    05 FILLER PIC X(010) VALUE SPACES.

01 REPORT-UNDERLINE.
    05 FILLER PIC X(001) VALUE SPACES.
    05 FILLER PIC X(004) VALUE '----'.
    05 FILLER PIC X(001) VALUE SPACES.
    05 FILLER PIC X(014) VALUE '-----'.
    05 FILLER PIC X(001) VALUE SPACES.
    05 FILLER PIC X(014) VALUE '-----'.
    05 FILLER PIC X(001) VALUE SPACES.
    05 FILLER PIC X(010) VALUE '-----'.
    05 FILLER PIC X(001) VALUE SPACES.
    05 FILLER PIC X(008) VALUE '-----'.
    05 FILLER PIC X(001) VALUE SPACES.
    05 FILLER PIC X(025) VALUE
        '-----'.
    05 FILLER PIC X(001) VALUE SPACES.
    05 FILLER PIC X(025) VALUE
        '-----'.
    05 FILLER PIC X(001) VALUE SPACES.
    05 FILLER PIC X(020) VALUE
        '-----'.
    05 FILLER PIC X(003) VALUE SPACES.

```

```

01 UNDERLINE                PIC X(131) VALUE SPACES.
01 TRANSMISSION-DATE-FILLER  PIC X(010) VALUE SPACES.
01 TRANSMISSION-TIME-FILLER  PIC X(008) VALUE SPACES.

01 TRANSMISSION-DATE.
   05 TRANSMISSION-MONTH     PIC X(002).
   05 FILLER                 PIC X(001) VALUE '/'.
   05 TRANSMISSION-DAY       PIC X(002).
   05 FILLER                 PIC X(001) VALUE '/'.
   05 TRANSMISSION-YEAR      PIC X(004).

01 TRANSMISSION-TIME.
   05 TRANSMISSION-HOUR      PIC X(002).
   05 FILLER                 PIC X(001) VALUE ':'.
   05 TRANSMISSION-MIN       PIC X(002).
   05 FILLER                 PIC X(001) VALUE ':'.
   05 TRANSMISSION-SEC       PIC X(002).

01 APP-ID.
   05 IN-T-APP-CNTRL-FLD     PIC X(025).

01 INTERNAL-TP-ID.
   05 IN-T-INT-TP-ID         PIC X(035).

01 REPORT-DATA-1.
   05 FILLER                 PIC X(002) VALUE SPACES.
   05 IN-T-TRANSACTION-TYPE  PIC X(001).
   05 FILLER                 PIC X(003) VALUE SPACES.
   05 IN-T-ENV-CNTRL-NUM     PIC X(014).
   05 FILLER                 PIC X(001) VALUE SPACES.
   05 IN-T-TRANSACTION-NUMBER PIC X(014).
   05 FILLER                 PIC X(001) VALUE SPACES.
   05 WS-TRANSMISSION-DATE   PIC X(010) VALUE SPACES.
   05 FILLER                 PIC X(001) VALUE SPACES.
   05 WS-TRANSMISSION-TIME   PIC X(008) VALUE SPACES.
   05 FILLER                 PIC X(001) VALUE SPACES.
   05 WS-T-INT-TP-ID         PIC X(025).
   05 FILLER                 PIC X(001) VALUE SPACES.
   05 WS-T-APP-CNTRL-FLD     PIC X(025).
   05 FILLER                 PIC X(001) VALUE SPACES.
   05 IN-T-TRAN-STAT-TEXT    PIC X(020).
   05 FILLER                 PIC X(003) VALUE SPACES.

01 REPORT-FOOTER.
   05 FILLER                 PIC X(050) VALUE SPACES.
   05 FILLER                 PIC X(029) VALUE
                                'COMPANY ABC INTERNAL USE ONLY'.
   05 FILLER                 PIC X(045) VALUE SPACES.
   05 FILLER                 PIC X(005) VALUE 'PAGE '.
   05 PAGE-COUNT             PIC 9(002) VALUE ZERO.

```

```

*****
* Miscellaneous program variables                                     *
*****

```

```

01 WS-DATE.
   05 WS-YEAR1               PIC X(002).

```

```

05 WS-MONTH1          PIC X(002).
05 WS-DAY1            PIC X(002).

01 END-OF-FILE-FLAG    PIC 9(001) VALUE 1.
01 LINE-COUNT          PIC 9(002) VALUE ZERO.
01 BLANK-LINE          PIC X(132) VALUE SPACES.
01 LINE-BREAK-KEY      PIC X(025) VALUE LOW-VALUES.

PROCEDURE DIVISION.

*****
* Main Procedure.  Opens input, output files.                *
*                  Reads and stores Data.                    *
*                  Writes headings to out-file.              *
*                  Reads Data from in file and                *
*                  Writes it to out file while                *
*                  there are records to read.                 *
*                  Writes Footer and Page number.            *
*                  Closes input, output files.                *
*                  ends execution.                            *
*****

1000-PROGRAM-MAINLINE.

    SORT SORT-FILE
      ON ASCENDING KEY APPLICATION-ID
                          TRANSACTION-ID
    USING IN-FILE
    GIVING IN-FILE.

    OPEN INPUT IN-FILE.
    OPEN OUTPUT OUT-FILE.

    ACCEPT WS-DATE FROM DATE.
    MOVE WS-MONTH1 TO WS-MONTH2.
    MOVE WS-DAY1 TO WS-DAY2.
    MOVE WS-YEAR1 TO WS-YEAR2.
    READ IN-FILE AT END MOVE ZERO TO END-OF-FILE-FLAG.
    MOVE CORRESPONDING IN-RECORD TO INTERNAL-TP-ID.
    MOVE INTERNAL-TP-ID TO LINE-BREAK-KEY.
    PERFORM 2000-PROCESS-DATA UNTIL END-OF-FILE-FLAG = ZERO.
    CLOSE IN-FILE OUT-FILE.
    STOP RUN.

*****
* Main control loop.  Reads records and writes                *
*                  records to OUT-FILE while                  *
*                  there are records to read                  *
*****

2000-PROCESS-DATA.

    IF LINE-COUNT = 0
      WRITE OUT-RECORD FROM BLANK-LINE
        AFTER ADVANCING PAGE
      WRITE OUT-RECORD FROM REPORT-TITLE
        AFTER ADVANCING 4 LINES

```



```

WRITE OUT-RECORD FROM COLUMN-HEADING-TOP
  AFTER ADVANCING 3 LINES
WRITE OUT-RECORD FROM COLUMN-HEADING-MIDDLE
WRITE OUT-RECORD FROM COLUMN-HEADING-BOTTOM
WRITE OUT-RECORD FROM REPORT-UNDERLINE
MOVE 9 TO LINE-COUNT.

MOVE CORRESPONDING IN-RECORD TO REPORT-DATA-1.
MOVE CORRESPONDING IN-RECORD TO INTERNAL-TP-ID.
MOVE INTERNAL-TP-ID TO WS-T-INT-TP-ID.
MOVE CORRESPONDING IN-RECORD TO APP-ID.
MOVE APP-ID TO WS-T-APP-CNTRL-FLD.

IF LINE-BREAK-KEY NOT = WS-T-INT-TP-ID
  ADD 1 TO LINE-COUNT
  WRITE OUT-RECORD FROM BLANK-LINE
  MOVE INTERNAL-TP-ID TO LINE-BREAK-KEY.

IF TRANSMISSION-YEAR OF IN-RECORD = ' '
  MOVE TRANSMISSION-DATE-FILLER TO WS-TRANSMISSION-DATE
  MOVE TRANSMISSION-TIME-FILLER TO WS-TRANSMISSION-TIME
ELSE
  MOVE CORRESPONDING IN-RECORD TO TRANSMISSION-DATE
  MOVE CORRESPONDING IN-RECORD TO TRANSMISSION-TIME
  MOVE TRANSMISSION-DATE TO WS-TRANSMISSION-DATE
  MOVE TRANSMISSION-TIME TO WS-TRANSMISSION-TIME.

ADD 1 TO LINE-COUNT.
WRITE OUT-RECORD FROM REPORT-DATA-1.
READ IN-FILE AT END MOVE ZERO TO END-OF-FILE-FLAG.

*****
* If no more records exist and a full page of data          *
* has not been generated then fill the rest of the         *
* page with blanks and print out page number               *
* and report footer.                                       *
*****

IF LINE-COUNT > 61 OR END-OF-FILE-FLAG = 0
  COMPUTE LINE-COUNT = 66 - LINE-COUNT
  ADD 1 TO PAGE-COUNT
  WRITE OUT-RECORD FROM REPORT-FOOTER
  AFTER ADVANCING LINE-COUNT LINES
  MOVE 0 TO LINE-COUNT.

```

Data Extract Report Generator (EDISAMS1)

The following routine reads in a SQIFILE that was created using the Perform Data Extract command. It then reads the records in and sorts them by the Application Control ID and writes them back out to the the same flat file. It then reads them in and writes them back out to SQOFILE in the defined report format.

In this example three records of length 1024 are concatenated together to form one record of length 3072 using the following command in the run JCL file.

```
PERFORM ENVELOPE DATA EXTRACT SELECTING
CONCATENATE(Y) INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y)
WHERE HANDLE(19930201) TO(19930228)
```

Once this command has been executed an unsorted flat file will exist with all the fields specified in IN-RECORD.

```
*****
* Module Name: EDISAMS1 *
* * *
* Dependencies: None. *
* * *
* Restrictions: None. *
* * *
* Language: COBOL. *
*****
```

IDENTIFICATION DIVISION.

PROGRAM-ID. EDISAMS1

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

```
*****
* Define Data source file, Sort file and Final report file *
*****
```

```
        SELECT SORT-FILE    ASSIGN UT-S-SORTWK01.
        SELECT IN-FILE      ASSIGN UT-S-SQIFILE.
        SELECT OUT-FILE     ASSIGN UT-S-SQOFILE.
```

DATA DIVISION.

FILE SECTION.

SD SORT-FILE.

```
01 SORT-RECORD.
   05 FILLER                PIC X(2390).
   05 TRANSACTION-ID        PIC X(0008).
   05 FILLER                PIC X(0070).
   05 APPLICATION-ID        PIC X(0035).
   05 FILLER                PIC X(0569).
```

FD IN-FILE

```
        RECORDING MODE F
        BLOCK CONTAINS 0 RECORDS
        RECORD CONTAINS 3072 CHARACTERS
        LABEL RECORDS ARE STANDARD.
```

```
*****
* In record definition *
*****
```

```

01  IN-RECORD.
    05  IN-E-REC-ID          PIC X(003).
    05  IN-E-NICKNAME       PIC X(016).
    05  IN-E-DIREC-SR       PIC X(001).
    05  IN-E-INT-CNTRLNUM   PIC X(014).
    05  IN-E-INT-RCVR-ID    PIC X(035).
    05  IN-E-FILLER         PIC X(072).
    05  IN-E-SENDER-ID      PIC X(035).
    05  IN-E-FAKE-FLAG      PIC X(001).
    05  IN-E-SEQ-ERROR-FLAG PIC X(001).
    05  IN-E-DPLCTE-INT-FLAG PIC X(001).
    05  IN-E-TEST-INT-FLAG  PIC X(001).
    05  IN-E-ENV-DATE       PIC X(014).
    05  E-FILLER-1         PIC X(002).
    05  TRANSMISSION-YEAR   PIC X(002).
    05  TRANSMISSION-MONTH  PIC X(002).
    05  TRANSMISSION-DAY    PIC X(002).
    05  TRANSMISSION-HOUR   PIC X(002).
    05  TRANSMISSION-MIN    PIC X(002).
    05  E-FILLER-2         PIC X(002).
    05  IN-E-TA1-ACK        PIC X(001).
    05  IN-E-TA2-DATE       PIC X(014).
    05  IN-E-NETWORK-STAT   PIC X(002).
    05  IN-E-NETWORK-STAT-TEXT PIC X(020).
    05  IN-E-ACK-EXPECTED   PIC X(001).
    05  IN-E-ACK-RECEIVED   PIC X(001).
    05  IN-E-ACK-DATE       PIC X(014).
    05  IN-E-MES-USER-CLASS PIC X(008).
    05  IN-E-MESSAGE-NAME   PIC X(008).
    05  IN-E-SEQ-NUM        PIC X(005).
    05  IN-E-MESSAGE-ID     PIC X(008).
    05  IN-E-PHY-DATA-SET-NAME PIC X(056).
    05  IN-E-GROUP-CNT      PIC X(011).
    05  IN-E-TRAN-CNT       PIC X(011).
    05  IN-E-SEG-CNT        PIC X(011).
    05  IN-E-INT-SIZE       PIC X(011).
    05  IN-E-INT-HEADER     PIC X(250).
    05  IN-E-INT-TRAILER    PIC X(030).
    05  IN-E-MRTCPAD1       PIC X(354).

*****
*GROUP RECORD DEFINITION*
*****
    05  IN-G-RECORD-ID      PIC X(003).
    05  IN-G-NICKNAME       PIC X(016).
    05  IN-G-DIRECTION      PIC X(001).
    05  IN-G-INT-CNTRL-NUM  PIC X(014).
    05  IN-G-INT-RECIEVER-ID PIC X(035).
    05  IN-G-GROUP-CNTRL-NUM PIC X(014).
    05  IN-G-FILLER         PIC X(058).
    05  IN-G-FAKE-FLAG      PIC X(001).
    05  IN-G-SENDER-ID      PIC X(035).
    05  IN-G-RECEIVER-ID    PIC X(035).
    05  IN-G-ACK-EXPECTED   PIC X(001).
    05  IN-G-ACK-RECEIVED   PIC X(001).
    05  IN-G-ACK-DATE       PIC X(014).
    05  IN-G-ACK-HANDLE     PIC X(020).

```

```

05 IN-G-TRANSACTION-COUNT PIC X(011).
05 IN-G-SEGMENT-COUNT PIC X(011).
05 IN-G-GROUP-SIZE PIC X(011).
05 IN-G-GROUP-HEADER PIC X(153).
05 IN-G-GROUP-TAILER PIC X(026).
05 IN-G-PADD PIC X(564).
*****
*TRANSACTIONS RECORD DEFINITION*
*****
05 IN-T-REC-ID PIC X(003).
05 IN-T-NICKNAME PIC X(016).
05 IN-T-DIRECTION-SR PIC X(001).
05 IN-T-ENV-CNTRL-NUM PIC X(014).
05 IN-T-INT-RCVR-ID PIC X(035).
05 IN-T-GROUP-NUMBER PIC X(014).
05 IN-T-TRANSACTION-NUMBER PIC X(014).
05 IN-T-CNTRL-HANDLE PIC X(020).
05 IN-T-TRAN-HANDLE PIC X(020).
05 IN-T-FILLER-PAD1 PIC X(004).
05 IN-T-ENV-DATA PIC X(014).
05 T-BLANKS PIC X(002).
05 TRANSACTION-YEAR PIC X(002).
05 TRANSACTION-MONTH PIC X(002).
05 TRANSACTION-DAY PIC X(002).
05 TRANSACTION-HOUR PIC X(002).
05 TRANSACTION-MIN PIC X(002).
05 T-BLANKS PIC X(002).
05 IN-T-TRAN-STAT-CODE PIC X(002).
05 IN-T-TRAN-STAT-TEXT PIC X(020).
05 IN-T-ACK-RCVED PIC X(001).
05 IN-T-ACD-RCVED-TEXT PIC X(020).
05 IN-T-ACK-DATE PIC X(014).
05 IN-T-TRX-ACK-EXPECTED PIC X(001).
05 IN-T-TRX-ACK-RCVD PIC X(001).
05 IN-T-TRX-ACK-RCVED-TEXT PIC X(020).
05 IN-T-TRX-ACK-DATE PIC X(014).
05 IN-T-ACK-HANDLE PIC X(020).
05 IN-T-SGMT-CNT PIC X(011).
05 IN-T-TRAN-SIZE PIC X(011).
05 IN-T-ENV-SEG-CNT PIC X(011).
05 IN-T-ENV-TRAN-SIZE PIC X(011).
05 IN-T-FORMAT-ID PIC X(016).
05 IN-T-TRANSACTION-TYPE PIC X(008).
05 IN-T-FNC-GRP-ID PIC X(006).
05 IN-T-ENV-TYPE PIC X(001).
05 IN-T-ENV-MEMBER PIC X(008).
05 IN-T-NETWORK-ID PIC X(008).
05 IN-T-TRAN-STAN-ID PIC X(008).
05 IN-T-STAN-VERSION PIC X(002).
05 IN-T-TAN-LEVEL PIC X(002).
05 IN-T-INT-TP-ID PIC X(035).
05 IN-T-APP-CNTRL-FLD PIC X(035).
05 IN-T-DLVRY-DATE PIC X(014).
05 IN-T-EARLIEST-ENV-DATE PIC X(008).
05 IN-T-EARLIEST-PRG-DATE PIC X(008).
05 IN-T-TRAN-ERROR-LVL PIC X(001).
05 IN-T-STORE-STAT PIC X(002).

```

```

05 IN-T-STORE-STAT-TEXT      PIC X(020).
05 IN-T-OVERRIDE-FLAG       PIC X(001).
05 IN-T-HELD-FLAG          PIC X(001).
05 IN-T-TEST-FLAG          PIC X(001).
05 IN-T-DUPLICATE-FLAG      PIC X(001).
05 IN-T-PURGE-FLAG         PIC X(001).
05 IN-T-TRANSLATE-FLAG      PIC X(001).
05 IN-T-DETACHED-FLAG      PIC X(001).
05 IN-T-OVERRIDE-HANDLE     PIC X(020).
05 IN-T-SECURITY-PROFILE    PIC X(008).
05 IN-T-ENCRYPTION-KEY      PIC X(016).
05 IN-T-AUTHENTICATION-KEY  PIC X(016).
05 IN-T-APPLICATION-NUMBER  PIC X(014).
05 IN-T-DATA-DELLIMITER     PIC X(001).
05 IN-T-SUB-DELLIMITER      PIC X(001).
05 IN-T-SEGMENT-TERMINATOR  PIC X(001).
05 IN-T-DECIMAL-NOTAION     PIC X(001).
05 IN-T-RELEASE-CHARACTER   PIC X(001).
05 IN-T-SEGMENT-ID-SEP      PIC X(001).
05 IN-T-TRANSACTION-HEADER  PIC X(085).
05 IN-T-TRANSACTION-TRAILER PIC X(026).
05 IN-T-FILLER-PAD3         PIC X(318).

FD OUT-FILE
  RECORDING MODE F
  BLOCK CONTAINS 0 RECORDS
  RECORD CONTAINS 132 CHARACTERS
  LABEL RECORDS ARE STANDARD.

01 OUT-RECORD                PIC X(132).

WORKING-STORAGE SECTION.

*****
* Define variables.                                     *
*****

01 LINE-BREAK-KEY            PIC X(022) VALUE ' '.
01 END-OF-FILE-FLAG         PIC 9(001).
01 LINE-COUNT               PIC 9(002) VALUE ZERO.
01 BLANK-LINE               PIC X(131) VALUE ' '.

01 WS-DATE.
  05 WS-YEAR1               PIC X(002).
  05 WS-MONTH1             PIC X(002).
  05 WS-DAY1               PIC X(002).

*****
* Define report heading records                         *
*****

01 REPORT-TITLE.
  05 FILLER                 PIC X(047) VALUE SPACES.
  05 FILLER                 PIC X(038) VALUE
    'TRADING PARTNER CAPABILITY REPORT DATA'.
  05 FILLER                 PIC X(038) VALUE SPACES.
  05 WS-DATE2.

```

```

10 WS-MONTH2      PIC X(002).
10 FILLER          PIC X(001) VALUE '/'.
10 WS-DAY2         PIC X(002).
10 FILLER          PIC X(001) VALUE '/'.
10 WS-YEAR2        PIC X(002).

```

```

*****
* Category report heading record                                     *
*****

```

```

01 COLUMN-HEADING-TOP.
05 FILLER          PIC X(006) VALUE SPACES.
05 FILLER          PIC X(011) VALUE 'APPLICATION'.
05 FILLER          PIC X(015) VALUE SPACES.
05 FILLER          PIC X(007) VALUE 'TRADING'.
05 FILLER          PIC X(009) VALUE SPACES.
05 FILLER          PIC X(012) VALUE 'TRANSMISSION'.
05 FILLER          PIC X(004) VALUE SPACES.
05 FILLER          PIC X(011) VALUE 'TRANSACTION'.
05 FILLER          PIC X(005) VALUE SPACES.
05 FILLER          PIC X(008) VALUE 'ENVELOPE'.
05 FILLER          PIC X(009) VALUE SPACES.
05 FILLER          PIC X(005) VALUE 'GROUP'.
05 FILLER          PIC X(010) VALUE SPACES.
05 FILLER          PIC X(005) VALUE 'TRANS'.
05 FILLER          PIC X(007) VALUE SPACES.
05 FILLER          PIC X(005) VALUE 'TRANS'.
05 FILLER          PIC X(003) VALUE SPACES.

01 COLUMN-HEADING-MID.
05 FILLER          PIC X(008) VALUE SPACES.
05 FILLER          PIC X(007) VALUE 'CONTROL'.
05 FILLER          PIC X(017) VALUE SPACES.
05 FILLER          PIC X(007) VALUE 'PARTNER'.
05 FILLER          PIC X(011) VALUE SPACES.
05 FILLER          PIC X(009) VALUE 'DATE&TIME'.
05 FILLER          PIC X(006) VALUE SPACES.
05 FILLER          PIC X(009) VALUE 'DATE&TIME'.
05 FILLER          PIC X(005) VALUE SPACES.
05 FILLER          PIC X(011) VALUE 'INTERCHANGE'.
05 FILLER          PIC X(004) VALUE SPACES.
05 FILLER          PIC X(011) VALUE 'INTERCHANGE'.
05 FILLER          PIC X(008) VALUE SPACES.
05 FILLER          PIC X(003) VALUE 'SET'.
05 FILLER          PIC X(008) VALUE SPACES.
05 FILLER          PIC X(004) VALUE 'TYPE'.
05 FILLER          PIC X(004) VALUE SPACES.

01 COLUMN-HEADING-BOT.
05 FILLER          PIC X(010) VALUE SPACES.
05 FILLER          PIC X(002) VALUE 'ID'.
05 FILLER          PIC X(022) VALUE SPACES.
05 FILLER          PIC X(002) VALUE 'ID'.
05 FILLER          PIC X(041) VALUE SPACES.
05 FILLER          PIC X(014) VALUE 'CONTROL NUMBER'.
05 FILLER          PIC X(001) VALUE SPACES.
05 FILLER          PIC X(014) VALUE 'CONTROL NUMBER'.

```

05	FILLER	PIC X(001) VALUE SPACES.
05	FILLER	PIC X(014) VALUE 'CONTROL NUMBER'.
05	FILLER	PIC X(002) VALUE SPACES.
05	FILLER	PIC X(006) VALUE 'NUMBER'.
05	FILLER	PIC X(003) VALUE SPACES.
01	REPORT-UNDERLINE.	
05	FILLER	PIC X(006) VALUE SPACES.
05	FILLER	PIC X(011) VALUE ALL '- '.
05	FILLER	PIC X(015) VALUE SPACES.
05	FILLER	PIC X(007) VALUE ALL '- '.
05	FILLER	PIC X(009) VALUE SPACES.
05	FILLER	PIC X(012) VALUE '-----'.
05	FILLER	PIC X(003) VALUE SPACES.
05	FILLER	PIC X(012) VALUE '-----'.
05	FILLER	PIC X(002) VALUE SPACES.
05	FILLER	PIC X(014) VALUE '-----'.
05	FILLER	PIC X(001) VALUE SPACES.
05	FILLER	PIC X(014) VALUE '-----'.
05	FILLER	PIC X(001) VALUE SPACES.
05	FILLER	PIC X(014) VALUE '-----'.
05	FILLER	PIC X(002) VALUE SPACES.
05	FILLER	PIC X(006) VALUE '-----'.
05	FILLER	PIC X(003) VALUE SPACES.
01	UNDERLINE	PIC X(131) VALUE SPACES.
01	TRANSMISSION-DATE-FILLER	PIC X(014) VALUE SPACES.
01	TRANSACTION-DATE-FILLER	PIC X(014) VALUE SPACES.
01	TRANSMISSION-DATE.	
05	TRANSMISSION-YEAR	PIC X(002).
05	FILLER	PIC X(001) VALUE '/ '.
05	TRANSMISSION-MONTH	PIC X(002).
05	FILLER	PIC X(001) VALUE '/ '.
05	TRANSMISSION-DAY	PIC X(002).
05	FILLER	PIC X(001) VALUE ' '.
05	TRANSMISSION-HOUR	PIC X(002).
05	FILLER	PIC X(001) VALUE ': '.
05	TRANSMISSION-MIN	PIC X(002).
01	TRANSACTION-DATE.	
05	TRANSACTION-YEAR	PIC X(002).
05	FILLER	PIC X(001) VALUE '/ '.
05	TRANSACTION-MONTH	PIC X(002).
05	FILLER	PIC X(001) VALUE '/ '.
05	TRANSACTION-DAY	PIC X(002).
05	FILLER	PIC X(001) VALUE ' '.
05	TRANSACTION-HOUR	PIC X(002).
05	FILLER	PIC X(001) VALUE ': '.
05	TRANSACTION-MIN	PIC X(002).
01	APP-ID.	
05	IN-T-APP-CNTRL-FLD	PIC X(022).
01	REPORT-DATA-1.	
05	FILLER	PIC X(001) VALUE SPACES.
05	WS-T-APP-CNTRL-FLD	PIC X(022).

```

05 FILLER PIC X(001) VALUE SPACES.
05 IN-T-INT-TP-ID PIC X(022).
05 FILLER PIC X(001) VALUE SPACES.
05 WS-E-TRANSMISSION-DATE PIC X(014) VALUE SPACES.
05 FILLER PIC X(001) VALUE SPACES.
05 WS-T-TRANSACTION-DATE PIC X(014) VALUE SPACES.
05 FILLER PIC X(001) VALUE SPACES.
05 IN-T-ENV-CNTRL-NUM PIC X(014).
05 FILLER PIC X(001) VALUE SPACES.
05 IN-T-GROUP-NUMBER PIC X(014).
05 FILLER PIC X(001) VALUE SPACES.
05 IN-T-TRANSACTION-NUMBER PIC X(014).
05 FILLER PIC X(001) VALUE SPACES.
05 IN-T-TRANSACTION-TYPE PIC X(008).
05 FILLER PIC X(001) VALUE SPACES.

01 REPORT-FOOTER.
05 FILLER PIC X(050) VALUE SPACES.
05 FILLER PIC X(029) VALUE
    'COMPANY ABC INTERNAL USE ONLY'.
05 FILLER PIC X(045) VALUE SPACES.
05 FILLER PIC X(005) VALUE 'PAGE '.
05 PAGE-COUNT PIC 9(003) VALUE ZERO.

```

PROCEDURE DIVISION.

```

*****
* Main Procedure.  Opens input, output files. Reads and stores *
*                  Data. Writes headings to OUT-FILE. Reads   *
*                  Data from in file and writes it to OUT-FILE *
*                  while there are records to read. Writes     *
*                  footer, page number, closes input and       *
*                  closes output files.                         *
*****

```

1000-PROGRAM-MAINLINE.

```

SORT SORT-FILE
  ON ASCENDING KEY APPLICATION-ID
                  TRANSACTION-ID
  USING IN-FILE
  GIVING IN-FILE.

OPEN INPUT IN-FILE.
OPEN OUTPUT OUT-FILE.

ACCEPT WS-DATE FROM DATE.
MOVE WS-MONTH1 TO WS-MONTH2.
MOVE WS-DAY1 TO WS-DAY2.
MOVE WS-YEAR1 TO WS-YEAR2.
MOVE 1 TO END-OF-FILE-FLAG.
READ IN-FILE AT END MOVE ZERO TO END-OF-FILE-FLAG.
MOVE CORRESPONDING IN-RECORD TO APP-ID.
MOVE APP-ID TO LINE-BREAK-KEY.
PERFORM 2000-PROCESS-DATA UNTIL END-OF-FILE-FLAG = ZERO.
CLOSE IN-FILE OUT-FILE.
STOP RUN.

```



```

*****
* Main control loop. Reads records and writes                *
*                      records to OUT-FILE while              *
*                      there are records to read              *
*****
2000-PROCESS-DATA.
    IF LINE-COUNT = 0
        WRITE OUT-RECORD FROM BLANK-LINE
        AFTER ADVANCING PAGE
        WRITE OUT-RECORD FROM REPORT-TITLE
        AFTER ADVANCING 2 LINES
        WRITE OUT-RECORD FROM COLUMN-HEADING-TOP
        AFTER ADVANCING 3 LINES
        WRITE OUT-RECORD FROM COLUMN-HEADING-MID
        WRITE OUT-RECORD FROM COLUMN-HEADING-BOT
        WRITE OUT-RECORD FROM REPORT-UNDERLINE
        MOVE 9 TO LINE-COUNT.

*****
* Move corresponding data into indicated fields.              *
*****

    MOVE CORRESPONDING IN-RECORD TO REPORT-DATA-1.
    MOVE CORRESPONDING IN-RECORD TO APP-ID.
    MOVE APP-ID TO WS-T-APP-CNTRL-FLD.

    IF LINE-BREAK-KEY NOT = APP-ID
        MOVE CORRESPONDING IN-RECORD TO APP-ID
        MOVE APP-ID TO LINE-BREAK-KEY
        WRITE OUT-RECORD FROM BLANK-LINE
        ADD 1 TO LINE-COUNT.

*****
* If TRANSMISSION-YEAR IS BLANK THEN FILL TRANSMISSION-DATE *
* with blanks                                                *
*****

    IF TRANSMISSION-YEAR OF IN-RECORD = ' '
        MOVE TRANSMISSION-DATE-FILLER TO WS-E-TRANSMISSION-DATE
    ELSE
        MOVE CORRESPONDING IN-RECORD TO TRANSMISSION-DATE
        MOVE TRANSMISSION-DATE TO WS-E-TRANSMISSION-DATE.

*****
* If TRANSACTION-YEAR IS BLANK THEN FILL TRANSACTION-DATE   *
* with blanks                                                *
*****

    IF TRANSACTION-YEAR OF IN-RECORD = ' '
        MOVE TRANSACTION-DATE-FILLER TO WS-T-TRANSACTION-DATE
    ELSE
        MOVE CORRESPONDING IN-RECORD TO TRANSACTION-DATE
        MOVE TRANSACTION-DATE TO WS-T-TRANSACTION-DATE.

    WRITE OUT-RECORD FROM REPORT-DATA-1
    ADD 1 TO LINE-COUNT

```

```

        READ IN-FILE AT END MOVE ZERO TO END-OF-FILE-FLAG.
*****
* If no more records exist and a full page of data          *
* has not been generated then fill the rest of the         *
* page with blanks and print out page number               *
* and report footer.                                       *
*****
        IF LINE-COUNT > 061 OR END-OF-FILE-FLAG = 0
          COMPUTE LINE-COUNT = 66 - LINE-COUNT
          ADD 1 TO PAGE-COUNT
          WRITE OUT-RECORD FROM REPORT-FOOTER
            AFTER ADVANCING LINE-COUNT LINES
          MOVE 0 TO LINE-COUNT.

```

Network Activity Report Generator (EDISAMT1)

The following routine reads a flat file and generates a formatted report and stores the report in a flat file.

| This sample program demonstrates how to read the output file produced by the command PERFORM
 | TRADING PARTNER CAPABILITY DATA EXTRACT and create a report that shows which transactions your trading
 | partners are capable of exchanging with you. The report also shows how many transactions you have
 | exchanged with each trading partner in both test and production and the number of transactions in error.

```

*****
* Module Name: EDISAMT1                                     *
*                                                       *
* Dependencies: None.                                       *
*                                                       *
* Restrictions: None.                                       *
*                                                       *
* Language: COBOL.                                          *
*                                                       *
*****

```

IDENTIFICATION DIVISION.

PROGRAM-ID. EDISAMT1

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

```

        SELECT IN-FILE ASSIGN UT-S-MRFILE1.
        SELECT OUT-FILE ASSIGN UT-S-MRFILE2.

```

DATA DIVISION.

FILE SECTION.

FD IN-FILE

```

        RECORDING MODE F
        BLOCK CONTAINS 0 RECORDS
        RECORD CONTAINS 1024 CHARACTERS

```

LABEL RECORDS ARE STANDARD.

```
01 IN-RECORD.
   05 IN-REC_ID          PIC X(003).
   05 IN-NICKNAME        PIC X(016).
   05 INTERNAL-TP-ID.
       10 IN-INTERNAL-TP-ID PIC X(006).
       10 FILLER          PIC X(029).
   05 IN-COMPANY-NAME    PIC X(040).
   05 IN-ADDRESS-1ALL    PIC X(040).
   05 ADDRESS-2.
       10 IN-ADDRESS-2    PIC X(020).
       10 FILLER          PIC X(020).
   05 IN-COMMENT-1      PIC X(040).
   05 IN-COMMENT-2      PIC X(040).
   05 IN-DIRECTION      PIC X(001).
   05 IN-STANDARD-ID    PIC X(008).
   05 IN-VERSION-STANDARD PIC X(002).
   05 IN-MRTCLVL        PIC X(002).
   05 IN-DESCRIPTON-STND PIC X(050).
   05 STANDARD-TRA-ID.
       10 IN-STANDARD-TRA-ID PIC X(006).
       10 FILLER          PIC X(002).
   05 IN-MAP-TRANS-ID PIC X(016).
   05 IN-MEASUREMENT-DATE PIC X(008).
   05 FILLER            PIC X(001).
   05 IN-MEASUREMENT-ID  PIC X(001).
   05 FILLER            PIC X(002).
   05 NUMBER-TRANS.
       10 IN-NUMBER-TRANS  PIC X(007).
       10 FILLER          PIC X(004).
   05 NUMBER-ERRORS.
       10 IN-NUMBER-ERRORS PIC X(007).
       10 FILLER          PIC X(004).
   05 IN-FILLER-PAD      PIC X(652).
```

```
FD OUT-FILE
  RECORDING MODE F
  BLOCK CONTAINS 0 RECORDS
  RECORD CONTAINS 132 CHARACTERS
  LABEL RECORDS ARE STANDARD.
```

```
01 OUT-RECORD          PIC X(132).
```

WORKING-STORAGE SECTION.

```
*****
* Define report heading definition                                     *
*****
```

```
01 REPORT-HEADING-1.
   05 FILLER          PIC X(047) VALUE SPACES.
   05 FILLER          PIC X(038) VALUE
       'TRADING PARTNER CAPABILITY REPORT DATA'.
   05 FILLER          PIC X(039) VALUE SPACES.
   05 WS-DATE2.
```

```

10 WS-MONTH2      PIC X(002).
10 FILLER          PIC X(001) VALUE '/'.
10 WS-DAY2        PIC X(002).
10 FILLER          PIC X(001) VALUE '/'.
10 WS-YEAR2       PIC X(002).

```

```

*****
* Category column heading definition                                     *
*****

```

```

01 REPORT-HEADING-2.
05 FILLER          PIC X(003) VALUE SPACES.
05 FILLER          PIC X(012) VALUE 'TP NICK NAME'.
05 FILLER          PIC X(004) VALUE SPACES.
05 FILLER          PIC X(006) VALUE 'INT ID'.
05 FILLER          PIC X(002) VALUE SPACES.
05 FILLER          PIC X(006) VALUE 'TRX ID'.
05 FILLER          PIC X(002) VALUE SPACES.
05 FILLER          PIC X(003) VALUE 'DIR'.
05 FILLER          PIC X(007) VALUE SPACES.
05 FILLER          PIC X(006) VALUE 'MAP ID'.
05 FILLER          PIC X(007) VALUE SPACES.
05 FILLER          PIC X(012) VALUE 'TEST OR PROD'.
05 FILLER          PIC X(002) VALUE SPACES.
05 FILLER          PIC X(012) VALUE 'DATE STARTED'.
05 FILLER          PIC X(002) VALUE SPACES.
05 FILLER          PIC X(010) VALUE 'TOTAL TRXS'.
05 FILLER          PIC X(002) VALUE SPACES.
05 FILLER          PIC X(012) VALUE 'TOTAL ERRORS'.

```

```

*****
* Underline for column headings                                       *
*****

```

```

01 REPORT-UNDERLINE.
05 FILLER          PIC X(003) VALUE SPACES.
05 FILLER          PIC X(012) VALUE '-----'.
05 FILLER          PIC X(004) VALUE SPACES.
05 FILLER          PIC X(006) VALUE '-----'.
05 FILLER          PIC X(002) VALUE SPACES.
05 FILLER          PIC X(006) VALUE '-----'.
05 FILLER          PIC X(002) VALUE SPACES.
05 FILLER          PIC X(003) VALUE '---'.
05 FILLER          PIC X(007) VALUE SPACES.
05 FILLER          PIC X(006) VALUE '-----'.
05 FILLER          PIC X(007) VALUE SPACES.
05 FILLER          PIC X(012) VALUE '-----'.
05 FILLER          PIC X(002) VALUE SPACES.
05 FILLER          PIC X(012) VALUE '-----'.
05 FILLER          PIC X(002) VALUE SPACES.
05 FILLER          PIC X(010) VALUE '-----'.
05 FILLER          PIC X(002) VALUE SPACES.
05 FILLER          PIC X(012) VALUE '-----'.
05 FILLER          PIC X(022) VALUE SPACES.

```

```

*****
* Formatted report definition                                         *

```

```

*****

01  REPORT-DATA-1.
    05  FILLER                      PIC X(001).
    05  OUT-COMPANY-NAME             PIC X(016).
    05  FILLER                      PIC X(002).
    05  OUT-INTERNAL-TP-ID          PIC X(006).
    05  FILLER                      PIC X(002).
    05  OUT-STANDARD-TRA-ID         PIC X(006).
    05  FILLER                      PIC X(003).
    05  OUT-DIRECTION               PIC X(001).
    05  FILLER                      PIC X(003).
    05  OUT-MAP-TRANS-ID            PIC X(016).
    05  FILLER                      PIC X(007).
    05  OUT-MEASUREMENT-ID          PIC X(001).
    05  FILLER                      PIC X(010).
    05  OUT-MEASUREMENT-DATE        PIC X(008).
    05  FILLER                      PIC X(005).
    05  OUT-NUMBER-TRANS            PIC X(007).
    05  FILLER                      PIC X(006).
    05  OUT-NUMBER-ERRORS           PIC X(007).
    05  FILLER                      PIC X(003).
    05  FILLER                      PIC X(022).

01  REPORT-FOOTER.
    05  FILLER                      PIC X(050) VALUE SPACES.
    05  FILLER                      PIC X(029) VALUE
        'COMPANY ABC INTERNAL USE ONLY'.
    05  FILLER                      PIC X(045) VALUE SPACES.
    05  FILLER                      PIC X(005) VALUE 'PAGE '.
    05  PAGE-COUNT                  PIC 9(003) VALUE ZERO.

*****
* Miscellaneous program variables                                     *
*****

01  WS-DATE.
    05  WS-YEAR1                    PIC X(002).
    05  WS-MONTH1                  PIC X(002).
    05  WS-DAY1                    PIC X(002).

01  END-OF-FILE-FLAG                PIC 9(001) VALUE 1.

01  LINE-COUNT                      PIC 9(002) VALUE ZERO.

01  BLANK-LINE                      PIC X(132) VALUE SPACES.

01  COMPANY-NAME-TEMP               PIC X(040) VALUE LOW-VALUES.

PROCEDURE DIVISION.

*****
* Main Procedure.  Opens input, output files.                       *
*                  Writes Headings.                                  *
*                  Reads in records and writes                      *
*                  out report to report file.                       *
*                  Writes Footer and Page#.                          *
*****

```

```

*                               Closes input, output files.                               *
*                               ends execution                                           *
*****

1000-PROGRAM-MAINLINE.

    OPEN INPUT IN-FILE.
    OPEN OUTPUT OUT-FILE.
    ACCEPT WS-DATE FROM DATE.
    MOVE WS-MONTH1 TO WS-MONTH2.
    MOVE WS-DAY1 TO WS-DAY2.
    MOVE WS-YEAR1 TO WS-YEAR2.
    READ IN-FILE AT END MOVE ZERO TO END-OF-FILE-FLAG.
    PERFORM 2000-PROCESS-DATA UNTIL END-OF-FILE-FLAG = 0.
    CLOSE IN-FILE OUT-FILE.
    STOP RUN.

2000-PROCESS-DATA.

*****
* Print header, if necessary                                                           *
*****

    IF LINE-COUNT = 0
        WRITE OUT-RECORD FROM BLANK-LINE
            AFTER ADVANCING PAGE
        WRITE OUT-RECORD FROM REPORT-HEADING-1
            AFTER ADVANCING 2 LINES
        WRITE OUT-RECORD FROM REPORT-HEADING-2
            AFTER ADVANCING 3 LINES
        WRITE OUT-RECORD FROM REPORT-UNDERLINE
        MOVE 7 TO LINE-COUNT.

*****
* Build output data line                                                             *
*****

    MOVE SPACES          TO REPORT-DATA-1.
    MOVE IN-COMPANY-NAME  TO OUT-COMPANY-NAME.
    MOVE IN-INTERNAL-TP-ID TO OUT-INTERNAL-TP-ID.
    MOVE IN-STANDARD-TRA-ID TO OUT-STANDARD-TRA-ID.
    MOVE IN-DIRECTION     TO OUT-DIRECTION.
    MOVE IN-MAP-TRANS-ID  TO OUT-MAP-TRANS-ID.
    MOVE IN-MEASUREMENT-ID TO OUT-MEASUREMENT-ID.
    MOVE IN-MEASUREMENT-DATE TO OUT-MEASUREMENT-DATE.
    MOVE IN-NUMBER-TRANS   TO OUT-NUMBER-TRANS.
    MOVE IN-NUMBER-ERRORS  TO OUT-NUMBER-ERRORS.

*****
* Company name report break                                                         *
*****

    IF IN-COMPANY-NAME = COMPANY-NAME-TEMP AND LINE-COUNT > 7
        MOVE SPACES TO OUT-COMPANY-NAME
    ELSE
        ADD 1 TO LINE-COUNT
        WRITE OUT-RECORD FROM BLANK-LINE

```

```

        MOVE IN-COMPANY-NAME TO COMPANY-NAME-TEMP.

*****
* Print report data line                                     *
*****

        ADD 1 TO LINE-COUNT.
        WRITE OUT-RECORD FROM REPORT-DATA-1.

*****
* Read next record                                           *
*****

        READ IN-FILE AT END MOVE ZERO TO END-OF-FILE-FLAG.

*****
* Print footer, if necessary                                 *
*****

        IF LINE-COUNT > 61 OR END-OF-FILE-FLAG = 0
            COMPUTE LINE-COUNT = 66 - LINE-COUNT
            ADD 1 TO PAGE-COUNT
            WRITE OUT-RECORD FROM REPORT-FOOTER
                AFTER ADVANCING LINE-COUNT LINES
            MOVE 0 TO LINE-COUNT.

```

Initializing, Invoking, and Terminating HOT-DI

The following programs illustrate how to initialize, invoke, and terminate HOT-DI. When the HOT-DI session is initialized, storage is acquired to hold the CCB. For more information, see “COBOL CCB” on page C-2. This storage is acquired SHARED so that it persists across a number of subsequent HOT-DI tasks. In this particular example, the address of the CCB is kept in a temporary storage queue named EDICCB. When the HOT-DI session is terminated, the storage is freed.

COBOL HOT-DI Initialization Example

The following COBOL program illustrates how to initialize a HOT-DI session. For more information on the SNB definition, see “COBOL SNB” on page C-3, for the CCB definition, see “COBOL CCB” on page C-2, and for the FCB definition, see “COBOL FCB” on page C-3.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. EDIHOT1.

*****
*   Initialize one HOT-DI                                   *
*****

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

*****
*   Service Name Block  (SNB)                               *
*****

```

```

*****

01  SNB.
    05  FILLER          PIC X(12)  VALUE LOW-VALUES.
    05  ZSNBNAME        PIC X(08)  VALUE 'ENVSERV'.
    05  FILLER          PIC X(04)  VALUE LOW-VALUES.
    05  ZSNBPC          PIC S9(04) COMP-4 VALUE +4.
    05  FILLER          PIC X(06)  VALUE LOW-VALUES.

*****

*    Function Code Block  (FCB)                                *
*****

01  FCB.
    05  FILLER          PIC X(02)  VALUE LOW-VALUES.
    05  ZFCBFUNC        PIC S9(04) COMP-4 VALUE +4.

*****

*    Application ID                                           *
*****

01  APPLID              PIC X(08)  VALUE 'EDIFFS'.

*****

*    Message Area                                             *
*****

01  MSG.
    05  FILLER          PIC X(15)  VALUE 'HOT-DI INIT RC='.
    05  MSG-RC          PIC 9(06)  VALUE ZERO.
    05  FILLER          PIC X(05)  VALUE ' ERC='.
    05  MSG-ERC         PIC 9(06)  VALUE ZERO.

*****

*    Working Storage Variables                                *
*****

01  LEN                 PIC S9(04) COMP-4 VALUE +4.
01  ONE                 PIC S9(04) COMP-4 VALUE +1.

LINKAGE SECTION.

01  DFHCOMMAREA         PIC X(01).

01  LINKAGE-POINTERS.
    05  FILLER          PIC S9(08) COMP-4.
    05  CCB-PTR         PIC S9(08) COMP-4.

*****

*    Common Control Block  (CCB)                                *
*****

01  CCB.
    05  FILLER          PIC X(12).
    05  ZCCBRC          PIC S9(08) COMP-4.
    05  ZCCBERC         PIC S9(08) COMP-4.
    05  FILLER          PIC X(34).

```



```

05 ZCCBLPID      PIC X(06).
05 FILLER        PIC X(548).

```

PROCEDURE DIVISION.

```

EXEC CICS
  GETMAIN SET(CCB-PTR) LENGTH(608) SHARED
END-EXEC.

```

```

MOVE LOW-VALUES TO CCB.
MOVE 'ENU' TO ZCCBLPID.

```

```

CALL 'FXXZCBL' USING SNB CCB FCB APPLID.

```

```

MOVE ZCCBRC TO MSG-RC.
MOVE ZCCBERC TO MSG-ERC.

```

```

EXEC CICS
  DELETEQ TS QUEUE('EDICCB') NOHANDLE
END-EXEC.

```

```

EXEC CICS
  WRITEQ TS QUEUE('EDICCB')
        FROM(CCB-PTR) LENGTH(LEN) ITEM(ONE) MAIN
END-EXEC.

```

```

EXEC CICS
  SEND CONTROL ERASE FREEKB
END-EXEC.

```

```

EXEC CICS
  SEND FROM(MSG) LENGTH(32)
END-EXEC.

```

```

EXEC CICS
  RETURN
END-EXEC.

```

```

STOP RUN.

```

COBOL HOT-DI Invocation Example

The following COBOL program illustrates how to invoke the Utility through a HOT-DI session. In this particular example, the Utility command to be executed would be in a temporary storage queue named EDICMD. For more information on the SNB definition, see “COBOL SNB” on page C-3, for the CCB definition, see “COBOL CCB” on page C-2, for the FCB definition, see “COBOL FCB” on page C-3, and for the UCB definition, see “COBOL Utility Control Information Block” on page C-12.

IDENTIFICATION DIVISION.
PROGRAM-ID. EDIHOT2.

```
*****
*   Execute HOT-DI Utility                               *
*****
```

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

```
*****
*   Service Name Block  (SNB)                             *
*****
```

```
01  SNB.
    05  FILLER          PIC X(12)  VALUE LOW-VALUES.
    05  ZSNBNAME        PIC X(08)  VALUE 'UTILSRV'.
    05  FILLER          PIC X(04)  VALUE LOW-VALUES.
    05  ZSNBPC          PIC S9(04)  COMP-4 VALUE +4.
    05  FILLER          PIC X(06)  VALUE LOW-VALUES.
```

```
*****
*   Function Code Block  (FCB)                             *
*****
```

```
01  FCB.
    05  FILLER          PIC X(02)  VALUE LOW-VALUES.
    05  ZFCBFUNC        PIC S9(04)  COMP-4 VALUE +2.
```

```
*****
*   Utility Control Block  (UCB)                           *
*****
```

```
01  UCB.
    05  FILLER          PIC X(12)  VALUE LOW-VALUES.
    05  CMDNAME         PIC X(08)  VALUE 'EDICMD'.
    05  FILLER          PIC X(02)  VALUE LOW-VALUES.
    05  DELIMITER       PIC X(01)  VALUE SPACE.
    05  PRTNAME         PIC X(08)  VALUE 'PRTFILE'.
    05  FILLER          PIC X(101) VALUE LOW-VALUES.
    05  CCBRC           PIC S9(08)  COMP-4 VALUE ZERO.
    05  CCBERC          PIC S9(08)  COMP-4 VALUE ZERO.
    05  FILLER          PIC X(92)  VALUE LOW-VALUES.
    05  BATFLG          PIC X(01)  VALUE 'B'.
    05  FILLER          PIC X(15)  VALUE LOW-VALUES.
```

```
*****
*   Message Area                                           *
*****
```

```
01  MSG.
    05  FILLER          PIC X(14)  VALUE 'HOT-DI CCB RC='.
    05  MSG-RC1         PIC 9(06)  VALUE ZERO.
    05  FILLER          PIC X(05)  VALUE ' ERC='.
```

```

05 MSG-ERC1      PIC 9(06)  VALUE ZERO.
05 FILLER        PIC X(09)  VALUE ' UTIL RC=' .
05 MSG-RC2       PIC 9(06)  VALUE ZERO.
05 FILLER        PIC X(05)  VALUE ' ERC=' .
05 MSG-ERC2      PIC 9(06)  VALUE ZERO.

*****
*   Working Storage Variables                               *
*****

01 LEN           PIC S9(04) COMP-4 VALUE +4.
01 ONE           PIC S9(04) COMP-4 VALUE +1.

LINKAGE SECTION.

01 DFHCOMMAREA   PIC X(01) .

01 LINKAGE-POINTERS.
05 FILLER        PIC S9(08) COMP-4.
05 CCB-PTR       PIC S9(08) COMP-4.

*****
*   Common Control Block (CCB)                             *
*****

01 CCB.
05 FILLER        PIC X(12) .
05 ZCCBRC        PIC S9(08) COMP-4.
05 ZCCBERC       PIC S9(08) COMP-4.
05 FILLER        PIC X(588) .

PROCEDURE DIVISION.

EXEC CICS
  READQ TS QUEUE('EDICCB')
        INTO(CCB-PTR) LENGTH(LEN) ITEM(ONE)
END-EXEC.

CALL 'FXXZCBL' USING SNB CCB FCB UCB.

MOVE ZCCBRC TO MSG-RC1.
MOVE ZCCBERC TO MSG-ERC1.
MOVE CCBRC TO MSG-RC2.
MOVE CCBERC TO MSG-ERC2.

EXEC CICS
  SEND CONTROL ERASE FREEKB
END-EXEC.

EXEC CICS
  SEND FROM(MSG) LENGTH(57)
END-EXEC.

```

```
EXEC CICS
  RETURN
END-EXEC.
```

```
STOP RUN.
```

COBOL HOT-DI Termination Example

The following COBOL program illustrates how to terminate a HOT-DI session. For more information on the SNB definition, see “COBOL SNB” on page C-3, for the CCB definition, see “COBOL CCB” on page C-2, and for the FCB definition, see “COBOL FCB” on page C-3.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EDIHOT3.

*****
*   Terminate One HOT-DI                               *
*****

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

*****
*   Service Name Block (SNB)                             *
*****

01  SNB.
    05  FILLER          PIC X(12)  VALUE LOW-VALUES.
    05  ZSNBNAME        PIC X(08)  VALUE 'ENVSERV'.
    05  FILLER          PIC X(04)  VALUE LOW-VALUES.
    05  ZSNBPC          PIC S9(04)  COMP-4 VALUE +3.
    05  FILLER          PIC X(06)  VALUE LOW-VALUES.

*****
*   Function Code Block (FCB)                             *
*****

01  FCB.
    05  FILLER          PIC X(02)  VALUE LOW-VALUES.
    05  ZFCBFUNC        PIC S9(04)  COMP-4 VALUE +2.

*****
*   Message Area                                           *
*****

01  MSG.
    05  FILLER          PIC X(15)  VALUE 'HOT-DI TERM RC='.
    05  MSG-RC          PIC 9(06)  VALUE ZERO.
    05  FILLER          PIC X(05)  VALUE ' ERC='.
    05  MSG-ERC         PIC 9(06)  VALUE ZERO.

*****
*   Working Storage Variables                             *
*****
```

```

*****

01  LEN                PIC S9(04) COMP-4 VALUE +4.
01  ONE                PIC S9(04) COMP-4 VALUE +1.

LINKAGE SECTION.

01  DFHCOMMAREA        PIC X(01).

01  LINKAGE-POINTERS.
    05  FILLER          PIC S9(08) COMP-4.
    05  CCB-PTR         PIC S9(08) COMP-4.

*****
*   Common Control Block (CCB)                               *
*****

01  CCB.
    05  FILLER          PIC X(12).
    05  ZCCBRC          PIC S9(08) COMP-4.
    05  ZCCBERC         PIC S9(08) COMP-4.
    05  FILLER          PIC X(588).

PROCEDURE DIVISION.

    EXEC CICS
        READQ TS QUEUE('EDICCB')
            INTO(CCB-PTR) LENGTH(LEN) ITEM(ONE)
    END-EXEC.

    CALL 'FXXZCBL' USING SNB CCB FCB.

    MOVE ZCCBRC TO MSG-RC.
    MOVE ZCCBERC TO MSG-ERC.

    EXEC CICS
        FREEMAIN DATA(CCB)
    END-EXEC.

    EXEC CICS
        DELETEQ TS QUEUE('EDICCB')
    END-EXEC.

    EXEC CICS
        SEND CONTROL ERASE FREEKB
    END-EXEC.

    EXEC CICS
        SEND FROM(MSG) LENGTH(32)
    END-EXEC.

    EXEC CICS
        RETURN
    END-EXEC.

    STOP RUN.

```

Invoking Response Programs

Response programs are user applications that the DataInterchange for CICS Utility (or continuous receive facility) invokes. For more information about response programs, see “Response Applications” on page 5-27.

COBOL Response Program Example

The following COBOL program illustrates the skeleton structure of a response application. For more information on the DFHCOMMAREA definition, see “COBOL Utility Control Information Block” on page C-12.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EDIRESP.

*****
*   DataInterchange/CICS Response Program Sample   *
*****

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

LINKAGE SECTION.

01  DFHCOMMAREA.
    03  FFUS-SYNCVAL          PIC S9(09) COMP-4.
    03  FFUS-CMDP             PIC S9(09) COMP-4.
    03  FFUS-CMDLEN           PIC S9(09) COMP-4.
    03  FFUS-CMDNAME          PIC X(08).
    03  FFUS-CMDTYPE          PIC X(02).
    03  FFUS-DELIMITER        PIC X(01).
    03  FFUS-PRTNAME          PIC X(08).
    03  FFUS-PRTTYPE          PIC X(02).
    03  FFUS-RPTNAME          PIC X(08).
    03  FFUS-RPTTYPE          PIC X(02).
    03  FFUS-EXCPNAME         PIC X(08).
    03  FFUS-EXCPTYPE         PIC X(02).
    03  FFUS-TRAKNAME         PIC X(08).
    03  FFUS-TRAKTYPE         PIC X(02).
    03  FFUS-QRYNAME          PIC X(08).
    03  FFUS-QRYTYPE          PIC X(02).
    03  FFUS-APPLID           PIC X(08).
    03  FFUS-LANGID           PIC X(06).
    03  FFUS-RESPID           PIC X(08).
    03  FFUS-RESPTYP          PIC X(02).
    03  FFUS-RTERMID          PIC X(04).
    03  FFUS-USRFLD           PIC X(16).
    03  FFUS-SYSID            PIC X(08).
    03  FFUS-RESPFLAG         PIC X(01).
    03  FFUS-FILETYP          PIC X(02).
    03  FFUS-ECBP             PIC S9(09) COMP-4.
    03  FFUS-CCBRC            PIC S9(09) COMP-4.
    03  FFUS-CCBERC           PIC S9(09) COMP-4.
    03  FFUS-FILEID           PIC X(08).
    03  FFUS-APPFILE          PIC X(08).
```

```

03 FFUS-ABNDPCODE      PIC X(04).
03 FFUS-THANDLE        PIC X(20).
03 FFUS-FFIEHDR        PIC S9(09) COMP-4.
03 FFUS-FARC           PIC S9(09) COMP-4.
03 FFUS-FAERC          PIC S9(09) COMP-4.
03 FFUS-FABUILT        PIC X(01).
03 FFUS-FFMTFLG        PIC X(01).
03 FFUS-USERSYNC       PIC X(01).
03 FFUS-RES1           PIC X(01).
03 FFUS-USERCOND       PIC S9(09) COMP-4.
03 FFUS-RES2           PIC X(23).
03 FFUS-RESPACTV       PIC X(01).
03 FFUS-WORKNAME       PIC X(08).
03 FFUS-BATFLG         PIC X(01).
03 FFUS-NOEXCP         PIC X(01).
03 FFUS-INVPARM        PIC X(01).
03 FFUS-LOGACTV        PIC X(01).
03 FFUS-FFUSADDR       PIC S9(09) COMP-4.
03 FFUS-CCBP           PIC S9(09) COMP-4.
03 FFUS-FFMTCBP       PIC S9(08) COMP-4.

```

PROCEDURE DIVISION.

1000-MAIN-PROGRAM.

* Response program code here

```

EXEC CICS
  RETURN
END-EXEC.

```

GOBACK.

Field Exit Program

This section provides sample field exit programs. You can copy these samples and customize them for your application.

Sample 1

The following is the first sample field exit program and is written in C. For more information on the parameters to this routine, see "Send Parameters" on page 4-4 and "Receive Parameters" on page 4-6. For more information on the CCB definition, see "C CCB" on page C-25 and for the SNB definition, see "C SNB" on page C-25.

```

/*****
/* Module Name: EDITRX1 */
/*
/* Descriptive Name: Sample field exit routine */
/*
/* Function: */
/* Send: This is a sample program of a field user exit */
/* routine. This routine checks the value of a */
/* field and returns a single character indicating */
/* if the value checked was small, medium, large, */
/* huge, or unbelievable. It also keeps track of the */
/* number of such fields processed, which will be used */

```

```

/*          later by the EDITRX2 field exit.          */
/*          */
/*      Receive: This is a sample program of a field user exit */
/*          routine. This routine accepts a coded value */
/*          and returns a numerical value consistent with */
/*          that code (the reverse of the send process) */
/*          */
/* Language:      C */
/*          */
/* Attributes:    Reentrant, AMODE(31) RMODE(ANY) */
/*          INCLUDE OBJ(EDITRX1) */
/*          INCLUDE OBJ(FXXZCITF) */
/*          ENTRY FXXZCITF */
/*          NAME EDITRX1(R) */
/*          */
/* NOTES: Since this program is written in C, the entry point */
/*          must be FXXZCITF, which is provided by the */
/*          DataInterchange product FXXZCITF will establish the */
/*          necessary C environment and branch to the main entry */
/*          point of the program. */
/*          */
/*          Parameters passed from DataInterchange may be above the */
/*          line so this program has to be 31 bit addressable. */
/*          */
/*          The program uses its logical name to determine if send */
/*          or receive processing should be done. */
/*          */
/*          */
/* User Exit:      Two ADAMCTL entries are needed for this program */
/*          For send processing: */
/*          Logical name: EDITRX1 */
/*          Physical name: EDITRX1 */
/*          Language:      C */
/*          For receive processing: */
/*          Logical name: EDITRR1 */
/*          Physical name: EDITRX1 */
/*          Language:      C */
/*          */
/* PARAMETERS:    IN - Service Name Block (snb) */
/*          Common Control Block (ccb) */
/*          Address of application data */
/*          Address of offset of data within structure */
/*          Address of the length of the data */
/*          Address of a work buffer */
/*          Address for returning data */
/*          Address for length of returned data */
/*          */
/*          IN OUT - None */
/*          */
/*          OUT - COMMON CONTROL BLOCK */
/*          */
/*          RC      ERC      Meaning */
/*          01      00      Throw away the data */
/*          */
/*****
/* Structure definitions used by this program */
/*****/

```



```

#include "diccb.h"          /* CCB definition          */
#include "disnb.h"          /* SNB definition          */
typedef struct Workarea workarea;
struct Workarea {
    long count;             /* Number of fields processed */
    char tempbuf[50];       /* temporary work space      */
};
/* ----- */
/* Static data used by program */
/* ----- */
#define SMALL 'S'
#define MEDIUM 'M'
#define LARGE 'L'
#define HUGE 'H'
#define UNBELIEVEABLE 'U'
#define BREAD_BASKET 1000000000
#define HUGE_VAL 1000000
#define LARGE_VAL 10000
#define MEDIUM_VAL 1000
#define SMALL_VAL 1

main(snbptr,ccbptr,aptr,offset,length,workbuf,
    result,result_length)
    snb *snbptr; /* Service name block pointer set up by DI */
                /* to invoke this program */
    ccb *ccbptr; /* Common block pointer used by DI for all */
                /* function requests */
    char *aptr; /* Pointer to the application data */
    long *offset; /* Address of offset of field within structure */
    long *length; /* Address of length of data */
    workarea *workbuf; /* Address of a work area */
    char *result; /* Area where you can move result data if you */
                /* are so inclined. */
    long *result_length; /* Length of data moved to result area */
{
    long val; /* Converted application value */

    /* ----- */
    /* Initialize the temporary buffer space with zeros. */
    /* ----- */
    memset(workbuf->tempbuf,'\0',sizeof(workbuf->tempbuf));
    /* ----- */
    /* Move the input field to the temporary buffer area. */
    /* ----- */
    memcpy(workbuf->tempbuf,aptr,(int)*length);
    /* ----- */
    /* Determine send or receive processing by looking at your name. */
    /* ----- */
    if (!memcmp(snbptr->zsnbname,"EDITRX1 ",8)) {
        /* ----- */
        /* SEND processing, convert a numeric value to a size indicator.*/
        /* ----- */
        /* Convert the data to an integer value. */
        /* ----- */
        val = atol(workbuf->tempbuf);
        /* ----- */
        /* Continue according to the value. */
        /* ----- */
    }
}

```

```

/* ----- */
if (!val)
/* ----- */
/* Zero or invalid. Tell DataInterchange to throw it away. */
/* ----- */
ccbptr->zccbrc=1;
else {
/* ----- */
/* Increment the number of fields processed. */
/* Set the result length to 1 and set the result value based */
/* on the value of the input field. */
/* ----- */
workbuf->count += 1;
*result_length = 1;
if (val > BREAD_BASKET)
    *result = UNBELIEVEABLE;
else if (val > HUGE_VAL)
    *result = HUGE;
else if (val > LARGE_VAL)
    *result = LARGE;
else if (val > MEDIUM_VAL)
    *result = MEDIUM;
else
    *result = SMALL;
}
} else {
/* ----- */
/* RECEIVE processing. Convert size indicator to maximum */
/* value associated with that size. */
/* ----- */
switch (*workbuf->tempbuf) {
    case UNBELIEVEABLE:
        val=BREAD_BASKET+1;
        break;
    case HUGE:
        val=HUGE_VAL+1;
        break;
    case LARGE:
        val=LARGE_VAL+1;
        break;
    case MEDIUM:
        val=MEDIUM_VAL+1;
        break;
    case SMALL:
        val=SMALL_VAL;
        break;
    default:
        val=0;
}
*result_length=10;
xiti1toa(val,result,*result_length);
}
}

/*****
/* A function to convert a number to character form with leading */

```

```

/* zeros. */
/*****
    int
xitltoa (num,outs,nchar)
    long    num;          /* Number to convert */
    char    *outs;        /* Output string */
    int     nchar;        /* Number of output characters */

{
    /* ----- */
    /* Loop through and convert each character. */
    /* ----- */
    for (outs+=nchar-1; nchar; --nchar, num /= 10, --outs)
        *outs = num % 10 + '0';

    return 0;
}

```

Sample 2

The following is the second field exit routine and is written in C. It is used in combination with the first example EDITRX1 described above.

```

/*****
/* Module Name: EDITRX2 */
/*
/* Descriptive Name: Sample field exit routine */
/*
/* Function:      This is a sample program of a field user
/*                  exit routine. This routine takes the value that
/*                  has been accumulated by the EDITRX1 routine and
/*                  returns it for mapping by DataInterchange.
/*
/* Language:      C
/*
/* Attributes:    Reentrant, AMODE(31) RMODE(ANY)
/*                  INCLUDE OBJ(EDITRX2)
/*                  INCLUDE OBJ(FXXZCITF)
/*                  ENTRY FXXZCITF
/*                  NAME EDITRX2(R)
/*
/* NOTES:      Since this program is written in C, the entry point
/*                  must be FXXZCITF, which is provided by the
/*                  DataInterchange product.
/*                  FXXZCITF will establish the necessary C environment
/*                  and branch to the main entry point of the program.
/*
/*                  Parameters passed from DataInterchange may be above the
/*                  line so this program has to be 31 bit addressable
/*
/* User Exit:      An ADAMCTL entry is needed for this program
/*                  that specifies:
/*                      Logical name: EDITRX2
/*                      Physical name: EDITRX2
/*                      Language:      C
/*
/* PARAMETERS:    IN - Service Name Block (snb)

```

```

/*          Common Block          (ccb)          */
/*          Address of application data          */
/*          Address of offset of data within structure          */
/*          Address of the length of the data          */
/*          Address of a work buffer          */
/*          Address for returning data          */
/*          Address for length of returned data          */
/*          */
/*          IN OUT - None          */
/*          */
/*          OUT - COMMON CONTROL BLOCK          */
/*          */
/*          RC      ERC      Meaning          */
/*          01      00      Throw away the data          */
/*          */
/*****
/* Structure definitions used by this program          */
/*****
#include "diccb.h"          /* CCB definition          */
#include "disnb.h"          /* SNB definition          */
typedef struct Workarea workarea;
struct Workarea {
    long count;          /* Number of fields processed          */
    char tempbuf[50];          /* temporary work space          */
};

main(snbptr,ccbptra,aptr,offset,length,workbuf,
    result,result_length)
    snb      *snbptr;          /* Service name block pointer set up by DI          */
/*          to invoke this program          */
    ccb      *ccbptra;          /* Common block pointer used by DI for all          */
/*          function requests          */
    char      *aptr;          /* Pointer to the application data          */
    long      *offset;          /* Address of offset of field within structure          */
    long      *length;          /* Address of length of data          */
    workarea *workbuf;          /* Address of a work area          */
    char      *result;          /* Area where you can move result data if you          */
/*          are so inclined.          */
    long      *result_length; /* Length of data moved to result area          */
{
    /* -----          */
    /* Tell DataInterchange to throw away the value if the count is          */
    /* zero.          */
    /* -----          */
    if (!workbuf->count)
    /* -----          */
    /* Zero or invalid. Tell DataInterchange to throw it away.          */
    /* -----          */
        ccbptr->zccbrc=1;
    else {
        /* -----          */
        /* Move the value to the result area, set the result          */
        /* length, and clear the value for a possible next iteration.          */
        /* -----          */
        *result_length = 5;
        xitltoa(workbuf->count,result,5);
        workbuf->count = 0;
    }
}

```

```

    }
}
/*****
/* A function to convert a number to character form with leading
/* zeros.
*****/
int
xitltoa (num,outs,nchar)
    long    num;        /* Number to convert
    char    *outs;      /* Output string
    int     nchar;      /* Number of output characters

{
    /* -----
    /* Loop through and convert each character.
    /* -----
    for (outs+=nchar-1; nchar; --nchar, num /= 10, --outs)
        *outs = num % 10 + '0';

    return 0;
}

```

Sample 3

The following is the third field exit routine and is written in COBOL. For more information on the CCB definition, see "COBOL CCB" on page C-2 and for the SNB definition, see "COBOL SNB" on page C-3.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. EDICOB.

```

```

*****
* This is a sample COBOL field exit routine. Programs
* like this may be used to provide additional processing
* for an application field or data element during
* translation.
*
*
*****

```

```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.

```

```

*****
*                               SNB
*****

```

```

01  SNB-DATA.
    COPY DISNB.

```

```

*****
*                               CCB
*****

```

```

01  CCB-DATA.
    COPY DICCB.

```

```

*****
*                               STANDARD DATA                               *
*****

01  STD-DATA                      PIC X(30).

*****
*   OFFSET OF CURRENT DATA ELEMENT WITHIN SEGMENT   *
*****

01  SEG-OFFSET                    PIC 9(09) COMP.

*****
*   LENGTH OF THE FIELD BEING PASSED TO THE EXIT ROUTINE *
*****

01  FLD-LENGTH                    PIC 9(09) COMP.

*****
*   THE 4096K BYTE PERMANENT WORK AREA                *
*****

01  PERM-AREA.
    05  PERM-VAR                    PIC X(03).
    05  FILLER                      PIC X(4093).

*****
*   THE 1024K BYTE TEMPORARY WORK AREA                *
*****

01  TEMP-AREA                      PIC X(1024).

*****
*   LENGTH OF DATA IN THE TEMPORARY WORK AREA        *
*****

01  TEMP-LENGTH                    PIC 9(09) COMP.

PROCEDURE DIVISION USING SNB-DATA
                        CCB-DATA
                        STD-DATA
                        SEG-OFFSET
                        FLD-LENGTH
                        PERM-AREA
                        TEMP-AREA
                        TEMP-LENGTH.

1000-PROGRAM-START.

*****
* The physical name of this program is EDICOB. You can *
* define many logical names and associate them with this *
* physical program. This is done in the ADAMCTL profile. *
* In this example, HLEXIT and SIEXIT are logical names *
* associated with this physical module.                  *
*                                                         *

```

```

* When the map is defined, a selected data element can
* specify a logical user exit program in the SNB-NAME
* field. The values of the data element is passed in
* the STD-DATA field.
*
* The PERM-AREA is a permanent 4096-byte work area.
* DataInterchange initializes this area with binary zeros
* at the start of translation. After that, the format
* and content of the area are determined by the exit
* routine. The same work area is passed to every exit
* routine for the entire translation session.
*
* The TEMP-AREA is a temporary 1024-byte work area. This
* area provides a place for the exit routine to put a
* modified version of the input field data.
* DataInterchange initializes this area with blanks
* before each exit routine call.
*
* The TEMP-LENGTH field has a value of zero on entry to
* the exit routine. A nonzero value in this field when
* the exit routine returns to DataInterchange indicates
* that data in TEMP-AREA should be used as the data
* element value.
*
* In this example, the value of the data element
* associated with exit routine HLEXIT, is stored in
* PERM-VAR. When the data element associated with exit
* routine SIEXIT is translated, a conditional check
* occurs. If the data currently in PERM-VAR happens
* to be XXX, then the value of the data element
* associated with SIEXIT and currently being translated
* is overridden with XXX.
*
*****

```

```

      IF SNB-NAME = 'HLEXIT '
        MOVE STD-DATA TO PERM-VAR.

      IF SNB-NAME = 'SIEXIT ' AND PERM-VAR = 'XXX'
        MOVE PERM-VAR TO TEMP-AREA
        MOVE 3 TO TEMP-LENGTH.

      GOBACK.

```

Test for Filter Type

The following is the sample test for filter type exit routine. This sample illustrates the use of the call exit function (see “Call Exit Routine” on page 4-31). This function allows one exit program to transfer control to another exit program. In this particular case, this general function must look at the data being provided and determine which of the ANSI defined filters must be invoked to filter the data. For more information on the CCB definition, see “C CCB” on page C-25, for the SNB definition, see “C SNB” on page C-25, for the FCB definition, see “COBOL FCB” on page C-3, and for the security data block definition, see “C SPDB” on page C-34.

```

/*****
/* Module Name: EDITRF4 */
/*
/* Descriptive Name: Program to transfer control to the appropriate */
/* filtering routine based on input data */
/*
/* Function: This program checks the security profile data */
/* block to determine the type of filtering that */
/* should be done and invokes the appropriate routine. */
/*
/* Restrictions: None */
/*
/* Language: C */
/*
/* Attributes: Reentrant, AMODE(31) RMODE(ANY) */
/* INCLUDE OBJ(EDITRF4) */
/* INCLUDE OBJ(FXXZCITF) */
/* INCLUDE OBJ(FXXZC) */
/* ENTRY FXXZCITF */
/* NAME EDITRF4(R) */
/*
/* NOTES: Since this program is written in C, the entry point */
/* must be FXXZCITF which is provided by the DI product. */
/* FXXZCITF establishes the necessary C environment */
/* and branch to the main entry point of the program. */
/*
/* Parameters passed from DI may be above the line so this */
/* program has to be 31 bit addressable */
/*
/* User Exit: This program is distributed as part of */
/* DataInterchange and, therefore, an entry in the */
/* ADAMCTL profile is not required. This program */
/* has a logical name of IBMFILTR and a physical */
/* name of EDITRF4. */
/*
/* PARAMETERS: IN - Service Name Block (snb) */
/* Common Block (ccb) */
/* Function Block (fcb) */
/* Filter handle (fh) */
/* Security data block (spdb) */
/* Buffer size */
/* Buffer containing input data */
/* Buffer for building output data */
/* Length of data in input buffer */
/* Number of characters remaining that would */
/* not fit into the input buffer */
/*
/* IN OUT - None */
/*
/* OUT - CAS COMMON BLOCK */
/*
/* RC ERC Meaning */
/* 12 50 filter program not known */
/*
/*****
#define NULLPTR (void*)0
#include "disnb.h" /* SNB definition */

```



```

#include "diccb.h"           /* CCB definition */
#include "difcb.h"           /* FCB definition */
#include "dispdb.h"          /* Security data block definition */
/* ----- */
/* Structure definitions used by this program */
/* ----- */
/* ----- */
/* Static data used by program */
/* ----- */
static fcb callfcb={4,3};
static char hex[8]="EDIHEX ";
static char ascii[8]="EDIASCII";
static char baudot[8]="EDIBAUDO";

main(snbptr,ccbptr,fcdbptr,fh,spdbptr,
    bufsize,bufin,bufout,datalen,rbc)
    snb    *snbptr; /* Service name block pointer set up by DI */
            /* to invoke this program */
    ccb    *ccbptr; /* Common block pointer used by DI for all */
            /* function requests */
    fcb    *fcdbptr; /* Function block pointer which indicates */
            /* the direction of the filter */
            /* 1=SEND, 2=RECEIVE */
    long   *fh;      /* Filter handle which is used when GETTING */
            /* more data or PUTTING filtered data */
    spdb    *spdbptr; /* Security profile member that specified */
            /* hexadecimal filtering should occur */
    long   *bufsize; /* The size of the input/output buffers */
    char   *bufin;   /* Buffer that contains the source data */
    char   *bufout;  /* Buffer that contains the filtered data */
    long   *datalen; /* Length of data within the input buffer */
    long   *rbc;     /* The number of bytes remaining of the source*/
            /* that would not fit into the input buffer */
{
    char *callname; /* Name of routine to invoke */

    /* ----- */
    /* Determine the routine to invoke. */
    /* ----- */
    callname=NULLPTR;
    switch (spdbptr->fltrtype) {
        case '1': /* Hex filter */
            callname=hex;
            break;
        case '2': /* ASCII filter */
            callname=ascii;
            break;
        case '3': /* ASCII/BAUDOT filter */
            callname=baudot;
            break;
        default: /* USER defined filter */
            ccbptr->zccbrc = 12;
            ccbptr->zccberc = 50;
            break;
    }
}

```

```

    }
    if (callname)
        fxxzc(fh,ccbptr,&callfcb,callname,NULLPTR);
    return;
}

```

Filtration Exit Examples

The next sections show examples of the filtration programs that are provided by DataInterchange. The HEXADECIMAL (EDITRF1), ASCII (EDITRF2), and ASCII/BAUDOT (EDITRF3), programs implement filters as defined by the American National Standards Institute. They are shown here as examples and they illustrate the use of the get data ("Get Data Routine" on page 4-30) and put data ("Put Data Routine" on page 4-31) routines. For more information on the filtration routines and the parameter definitions required for various languages, see "Filtering Routine" on page 4-26.

Hexadecimal Filter Example

The following is the hexadecimal filter exit routine that is distributed with DataInterchange. For more information on the CCB definition, see "C CCB" on page C-25, for the SNB definition, see "C SNB" on page C-25, for the FCB definition, see "C FCB" on page C-26, and for the SPDB definition, see "C SPDB" on page C-34.

```

/* ----- */
/* Module Name: EDITRF1 */
/* */
/* Descriptive Name: HEXADECIMAL filter */
/* */
/* Function:      This program is invoked for HEXADECIMAL filtering */
/*                of standard transaction data before the data is */
/*                transmitted. */
/* */
/* Language:      C */
/* */
/* Attributes:    Reentrant, AMODE(31) RMODE(ANY) */
/*                INCLUDE OBJ(EDITRF1) */
/*                INCLUDE OBJ(FXXZCITF) */
/*                INCLUDE OBJ(FXXZC) */
/*                ENTRY FXXZCITF */
/*                NAME EDITRF1(R) */
/* */
/* NOTES:      Since this program is written in C, the entry point */
/*              must be FXXZCITF which is provided by the DI product. */
/*              FXXZCITF establishes the necessary C environment */
/*              and branch to the main entry point of the program. */
/* */
/*              Parameters passed from DI may be above the line so this */
/*              program has to be 31 bit addressable */
/* */
/* User Exit:    This program is distributed as part of */
/*              DataInterchange and therefore an entry in the */
/*              ADAMCTL profile is not required. This program */
/*              has a logical name of EDIHGX and a physical */
/*              name of EDITRF1. */
/* */
/* PARAMETERS:  IN - Service Name Block (snb) */

```

```

/*          Common Block      (ccb)          */
/*          Function Block    (fcb)          */
/*          Filter handle     (fh)           */
/*          Security data block (spdb)        */
/*          Buffer size        */
/*          Buffer containing input data       */
/*          Buffer for building output data    */
/*          Length of data in input buffer    */
/*          Number of characters remaining that would
/*          not fit into the input buffer     */
/*
/*          IN OUT - None
/*
/*          OUT - CAS COMMON BLOCK
/*
/*          RC      ERC      Meaning
/*          8       21      Invalid function code
/*          12      22      Filter failure occurred
/*          12      23      Input data in error
/*
/* -----
/* Structure definitions used by this program
/* -----
#include "disnb.h"          /* SNB definition
#include "diccb.h"          /* CCB definition
#include "difcb.h"          /* FCB definition
#include "dispdb.h"         /* Security data block definition
/* -----
/* Static data used by program
/* -----
static fcb getfcb={4,1};
static fcb putfcb={4,2};
static char Hexval[]={0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,
                      0xF6,0xF7,0xF8,0xF9,0xC1,0xC2,
                      0xC3,0xC4,0xC5,0xC6};

main(snbptr,ccbptra,fcbptr,fh,spdbptr,
     bufsize,bufin,bufout,datalen,rbcb)
snb    *snbptr;    /* Service name block pointer set up by DI
/* to invoke this program
ccb    *ccbptra;   /* Common block pointer used by DI for all
/* function requests
fcb    *fcbptr;    /* Function block pointer which indicates
/* the direction of the filter
/* 1=SEND, 2=RECEIVE
long   *fh;        /* Filter handle which is used when GETTING
/* more data or PUTTING filtered data
spdb    *spdbptr;  /* Security profile member that specified
/* hexadecimal filtering should occur
long   *bufsize;   /* The size of the input/output buffers
char   *bufin;     /* Buffer that contains the source data
char   *bufout;    /* Buffer that contains the filtered data
long   *datalen;   /* Length of data within the input buffer
long   *rbcb;      /* The number of bytes remaining of the source*
/* that would not fit into the input buffer
{
char   *ss;        /* Local pointer to source data

```

```

long   sl;           /* Length of data in source buffer      */
char   *ds;          /* Local pointer to destination data                        */
long   dl;           /* Length of data in destination buffer                    */
unsigned char c1,c2; /* Used in the conversions                                */

/* ----- */
/* Set up local pointers to data buffers and initialize the */
/* lengths in each buffer.                                  */
/* ----- */
ss = bufin;
sl = *datalen;
ds = bufout;
dl = 0;
/* ----- */
/* Determine the direction of the filter and process accordingly */
/* ----- */
switch (fcbptr->zfcfunc) {
    case 0: /* EXISTENCE check */
        break;
    case 2: /* RECEIVE filter operation */
        /* ----- */
        /* Return an error if an even number of bytes is not being */
        /* provided; otherwise, drop into the SEND code.            */
        /* ----- */
        if ((sl&0x01) (*rbc&0x01) (*bufsize&0x01)) {
            ccbptr->zccbrc = 12;
            ccbptr->zccberc = 23;
            break;
        }
    case 1: /* SEND filter operation */
        /* ----- */
        /* Begin processing every character of input, either in the */
        /* current input buffer or the residual that must be         */
        /* obtained by calling the GET/PUT service. This code        */
        /* assumes that bufsize and datalen are even values when     */
        /* a receive filter operation is being done.                 */
        /* ----- */
        for (; (sl>0) (*rbc>0); ++ss, --sl) {
            /* ----- */
            /* Obtain more source data if the current source has     */
            /* been exhausted.                                         */
            /* ----- */
            if (!sl) {
                /* ----- */
                /* Set -sl- to maximum size you can accept and call */
                /* service to get more data. The value in -sl- on    */
                /* return will be the number of bytes returned.      */
                /* ----- */
                sl = *bufsize;
                fxxzc(fh, ccbptr, &getfcb, bufin, &sl);
                ss = bufin;
            }
            /* ----- */
            /* Flush the output buffer if it will not accept 2 more */
            /* bytes.                                                 */
            /* ----- */
            if ((dl+2) > *bufsize) {

```

```

        fxxzc(fh,ccbptr,&putfcb,bufout,&d1);
        if (ccbptr->zccbrc) {
            ccbptr->zccbrc = 12;
            ccbptr->zccberc = 22;
            break;
        }
        ds = bufout;
        d1 = 0;
    }
    if (fcbptr->zfcfunc == 1) {
        /* ----- */
        /* Construct two characters of output for each */
        /* character of input. */
        /* ----- */
        c1 = ((*ss)>>4) & 0x0F;
        *ds++ = Hexval[c1];
        c1 = (*ss) & 0x0F;
        *ds++ = Hexval[c1];
        d1 += 2;
    } else {
        /* ----- */
        /* Construct one character of output for each two */
        /* characters of input. */
        /* ----- */
        c1 = *ss++;
        if ((c1 < 0xF0) (c1 > 0xF9))
            c1 = c1+9;
        c1 = (c1&0x0F)<<4;
        c2 = *ss;
        if ((c2 < 0xF0) (c2 > 0xF9))
            c2 = c2+9;
        c2 = (c2&0x0F);
        *ds++ = c1 c2;
        d1 += 1;
        sl -= 1;
    }
}
/* ----- */
/* Flush data remaining in the output buffer. */
/* ----- */
if (!ccbptr->zccbrc && d1) {
    fxxzc(fh,ccbptr,&putfcb,bufout,&d1);
    /* ----- */
    /* Set your return code if there were any errors. */
    /* ----- */
    if (ccbptr->zccbrc) {
        ccbptr->zccbrc = 12;
        ccbptr->zccberc = 22;
    }
}
break;

```

```

        default: /* UNKNOWN filter operation */
                /* ----- */
                /* Set the return codes and terminate. */
                /* ----- */
                ccbptr->zccbrc = 8;
                ccbptr->zccberc = 21;
                break;
    }
    return;
}

```

ASCII Filter Example

The following is the ASCII filter exit routine that is distributed with DataInterchange. For more information on the CCB definition, see “C CCB” on page C-25, for the SNB definition, see “C SNB” on page C-25, and for the FCB definition, see “C FCB” on page C-26.

```

/*****
/* Module Name: EDITRF2 */
/*
/* Descriptive Name: ASCII filter
/*
/* Function: This program is invoked for ASCII filtering of,
/*           standard transaction data before the data is
/*           transmitted or for the reversal process when
/*           data is received.
/*
/* Language: C
/*
/* Attributes: Reentrant, AMODE(31) RMODE(ANY)
/*             INCLUDE OBJ(EDITRF2)
/*             INCLUDE OBJ(FXXZCITF)
/*             INCLUDE OBJ(FXXZC)
/*             ENTRY FXXZCITF
/*             NAME EDITRF2(R)
/*
/* NOTES: Since this program is written in C, the entry point
/*         must be FXXZCITF, which is provided by the DI product.
/*         FXXZCITF establishes the necessary C environment
/*         and branches to the main entry point of the program.
/*
/*         Parameters passed from DI may be above the line so this
/*         program has to be 31 bit addressable
/*
/* User Exit: This program is distributed as part of
/*            DataInterchange and, therefore, an entry in the
/*            ADAMCTL profile is not required. This program
/*            has a logical name of EDIASCII and a physical
/*            name of EDITRF2.
/*
/* PARAMETERS: IN - Service Name Block (snb)
/*              Common Block (ccb)
/*              Function Block (fcb)
/*              Filter handle (fh)
/*              Security data block (spdb)
/*              Buffer size
/*              Buffer containing input data
*/

```

```

/*          Buffer for building output data          */
/*          Length of data in input buffer          */
/*          Number of characters remaining that would */
/*          not fit into the input buffer           */
/*          */
/*          IN OUT - None                          */
/*          */
/*          OUT - CAS COMMON BLOCK                  */
/*          */
/*          RC      ERC      meaning                */
/*          8       21      Invalid function code    */
/*          12      22      Filter failure occurred  */
/*          12      23      Input data in error      */
/*          */
/*****
/* Structure definitions used by this program
/* -----
#include "disnb.h"          /* SNB definition
#include "diccb.h"          /* CCB definition
#include "difcb.h"          /* FCB definition
#include "dispdb.h"         /* Security data block definition
/* -----
/* Static data used by program
/* -----
static fcb getfcb={4,1};
static fcb putfcb={4,2};
static char hflag[]={0x00,0x80,0xc0,0xe0,0xf0,0xf8,0xfc,0xfe};
static char lflag[]={0x00,0x01,0x03,0x07,0x0f,0x1f,0x3f,0x7f,0xff};
static char AsciiTab[]={
    0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,
    0x29,0x2a,0x2b,0x2c,0x2e,0x2f,
    0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
    0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f,
    0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,
    0x48,0x49,0x4a,0x4b,0x4c,0x4d,0x4e,0x4f,
    0x50,0x51,0x52,0x53,0x54,0x55,0x56,0x57,
    0x58,0x59,0x5a,0x5b,0x5c,0x5d,0x5e,0x5f,
    0x60,0x61,0x62,0x63,0x64,0x65,0x66,0x67,
    0x68,0x69,0x6a,0x6b,0x6c,0x6d,0x6e,0x6f,
    0x70,0x71,0x72,0x73,0x74,0x75,0x76,0x77,
    0x78,0x79,0x7a,0x7b,0x7c,0x7d,0x7e};

main(snbptr,ccbptr,fcdbptr,fh,spdbptr,
    bufsize,bufin,bufout,datalen,rbcb)
    snb      *snbptr;    /* Service name block pointer set up by DI */
/* to invoke this program */
    ccb      *ccbptr;    /* Common block pointer used by DI for all */
/* function requests */
    fcb      *fcdbptr;   /* Function block pointer which indicates */
/* the direction of the filter */
/* 1=SEND, 2=RECEIVE */
    long     *fh;        /* Filter handle which is used when GETTING */
/* more data or PUTTING filtered data */
    spdb     *spdbptr;   /* Security profile member that specified */
/* ASCII filtering should occur */
    long     *bufsize;   /* The size of the input/output buffers */
    char     *bufin;     /* Buffer that contains the source data */

```

```

char    *bufout;    /* Buffer that contains the filtered data    */
long    *datalen;   /* Length of data within the input buffer    */
long    *rbc;       /* The number of bytes remaining of the source*/
                        /* that would not fit into the input buffer */
{
    unsigned char *ss; /* Local pointer to source data    */
    long    sl;        /* Length of data in source buffer    */
    char    *ds;       /* Local pointer to destination data  */
    long    dl;        /* Length of data in destination buffer */
    int     bcr;       /* Number of bits left in current byte */
    int     bcw;       /* Number of bits wanted for source data */
    unsigned int iv,fv; /* Index into the ASCII table    */
    int     bi;        /* Indicates when 2 bytes have been obtained */
    char    tv[2];     /* Holds two bytes of input    */
    int     done;      /* Processed finished when =1    */

    /* ----- */
    /* Set up local pointers to data buffers and initialize the */
    /* lengths in each buffer. */
    /* ----- */
    ss = bufin;
    sl = *datalen;
    ds = bufout;
    dl = 0;
    bcr=8;
    done=0;
    fv=0;
    /* ----- */
    /* Determine the direction of the filter and process accordingly. */
    /* ----- */
    switch (fcbptr->zfcfunc) {
        case 0: /* EXISTENCE check */
            break;
        case 1: /* SEND filter operation */
            /* ----- */
            /* Begin processing every character of input, either in the */
            /* current input buffer or the residual that must be */
            /* obtained by calling the GET/PUT service. */
            /* ----- */
            for (;!done;) {
                /* ----- */
                /* Flush the output buffer if it will not accept 2 more */
                /* bytes */
                /* ----- */
                if ((dl+2) > *bufsize) {
                    fxxzc(fh,ccbptr,&putfcb,bufout,&dl);
                    if (ccbptr->zccbrc) {
                        ccbptr->zccbrc = 12;
                        ccbptr->zccberc = 22;
                        break;
                    }
                    ds = bufout;
                    dl = 0;
                }
            }
            /* ----- */
            /* If you are picking up the final value (fv), then set */
            /* the terminate flag; otherwise, get the next 13 bits */

```



```

/* from the input stream. */
/* ----- */
if (iv=fv)
    done = 1;
else for (bcw=13;bcw && !fv;) {
    if (bcw < bcr) {
        /* ----- */
        /* The number of bits required can be satisfied */
        /* from the current byte. */
        /* 1. Shift the value to accept the next bits. */
        /* 2. Decrement the number of bits remaining */
        /* in byte. */
        /* 3. Extract, align the bits wanted, */
        /* and incorporate */
        /* 4. Set the number of bits wanted to zero. */
        /* ----- */
        iv = iv<<bcw;
        bcr = 8-bcw;
        iv = ((*ss&hflag[bcw])>>bcr);
        bcw = 0;
    } else {
        /* ----- */
        /* Make room the new bits, extract the remaining */
        /* bits from the current bytes and incorporate into */
        /* the result. */
        /* ----- */
        iv = (iv<<bcr) + (*ss&lflag[bcr]);
        bcw -= bcr;
        bcr = 8;
        ss += 1;
        sl -= 1;
        /* ----- */
        /* Obtain more source data if the current source */
        /* has been exhausted. */
        /* ----- */
        if (!sl && *rbc) {
            /* ----- */
            /* Set -sl- to maximum size you can accept and */
            /* call service to get more data. On return -sl- */
            /* will be the number of bytes returned. */
            /* ----- */
            sl = *bufsize;
            fxxzc(fh,ccbptr,&getfcb,bufin,&sl);
            ss=bufin;
        }
        /* ----- */
        /* If no data remains, then pad data to a multiple */
        /* of 13 bits. */
        /* ----- */
        if (!sl && !*rbc) {
            if (bcw >= 4) {
                /* ----- */
                /* Pad the current value and set terminate */
                /* flag. */
                /* ----- */
                iv = (iv<<bcw)+(bcw-1);
                done = 1;
            }
        }
    }
}

```

```

        } else {
            /* ----- */
            /* Pad current value with zeros and set the */
            /* value for the final 13 bits. */
            /* ----- */
            iv = iv << (bcw);
            fv = 12+bcw;
        }
        break;
    }
}
}
}
*ds++ = Asciitab[iv/sizeof(Asciitab)];
*ds++ = Asciitab[iv%sizeof(Asciitab)];
dl += 2;
}
break;
case 2: /* RECEIVE filter operation */
/* ----- */
/* Return an error if an even number of bytes is not being */
/* provided. */
/* ----- */
if ((sl + *rbc)&0x01) {
    ccbptr->zccbrc = 12;
    ccbptr->zccberc = 23;
    break;
}
/* ----- */
/* Continue process as long as there is data in the current */
/* buffer or residual data. */
/* ----- */
bi = 0;
bcr = 8;
*ds = '\0';
for (; sl *rbc;) {
    if (!sl) {
        /* ----- */
        /* Obtain more data. Set -sl- to maximum amount you */
        /* can accept. On return -sl- will contain the number */
        /* of bytes received. */
        /* ----- */
        sl = *bufsize;
        fxxzc(fh,ccbptr,&getfcb,bufin,&sl);
        ss=bufin;
    }
    /* ----- */
    /* Get 2 bytes from the input source. */
    /* ----- */
    tv[bi]=*ss++;
    sl -= 1;
    if (++bi != 2) continue;
    bi = 0;
    /* ----- */
    /* Determine the 13-bit value associated with 2 bytes. */
    /* ----- */
    iv = getidx(*tv);
    fv = getidx(*(tv+1));

```

```

    iv = (iv*sizeof(Asciitab))+fv;
    /* ----- */
    /* Remove any padding bits that were added. */
    /* ----- */
    if ((!sl) && (!*rbc)) {
        bcw = (iv&0x0f)+1;
        if (bcw>=13)
            break;
        else
            bcw=13-bcw;
    } else
        bcw=13;
    /* ----- */
    /* Put the 13 bits into the output buffer. */
    /* ----- */
    for (; bcw;) {
        if (bcr<=bcw) {
            *ds++ = (iv>>(13-bcr));
            dl += 1;
            bcw -= bcr;
            iv = iv<<bcr;
            bcr = 8;
            *ds = '\0';
        } else {
            bcr = (8-bcw);
            *ds = iv>>5;
            bcw = 0;
        }
    }
    /* ----- */
    /* Flush the output buffer if you have a complete byte */
    /* and the buffer will not hold 13 more bytes of data. */
    /* ----- */
    if ((bcr==8) && ((dl+13)>*bufsize)) {
        fxxzc(fh,ccbptr,&putfcb,bufout,&dl);
        if (ccbptr->zccbrc) {
            ccbptr->zccbrc = 12;
            ccbptr->zccberc = 22;
            break;
        }
        ds = bufout;
        dl = 0;
        *ds = '\0';
    }
    break;
default: /* UNKNOWN filter operation */
    /* ----- */
    /* Set the return codes and terminate. */
    /* ----- */
    ccbptr->zccbrc = 8;
    ccbptr->zccberc = 21;
    break;
}
/* ----- */
/* Flush data remaining in the output buffer. */
/* ----- */

```

```

    if (!ccbptr->zccbrc && dl) {
        fxxzc(fh,ccbptr,&putfcb,bufout,&dl);
        /* ----- */
        /* Set the return code if there were any errors. */
        /* ----- */
        if (ccbptr->zccbrc) {
            ccbptr->zccbrc = 12;
            ccbptr->zccberc = 22;
        }
    }
    return;
}
/* ----- */
/* $MINOR */
/* ----- */
/* Routine Name: getidx      (Return index value into AsciiTab) */
/* ----- */
/* Function:      Determines the index value of a char into AsciiTab */
/* ----- */
/* Arguments:     char to search for */
/* ----- */
/* Return Values: index value */
/* ----- */
/* ----- */
int
getidx(sc)
    register int sc; /* Character to search for */
{
    register char *ts;
    register int ii;

    ts=AsciiTab;
    for (ii=0;(sc != *ts)&&(ii<sizeof(AsciiTab));++ts,++ii);
    return (ii==sizeof(AsciiTab))? 0:ii;
}

```

ASCII/BAUDOT Filter Example

The following is the ASCII/BAUDOT filter exit routine that is distributed with DataInterchange. For more information on the CCB definition, see "C CCB" on page C-25, for the SNB definition, see "C SNB" on page C-25, for the FCB definition, see "C FCB" on page C-26, and for the SPDB definition, see "C SPDB" on page C-34.

```

/* ----- */
/* Module Name: EDITRF3 */
/* ----- */
/* Descriptive Name: ASCII/BAUDOT filter */
/* ----- */
/* Function:      This program is invoked for ASCII/BAUDOT filtering, */
/*                of standard transaction data before the data is */
/*                transmitted or for the reversal process when */
/*                data is received. */
/* ----- */
/* Language:      C */
/* ----- */
/* Attributes:     Reentrant, AMODE(31) RMODE(ANY) */
/*                INCLUDE OBJ(EDITRF3) */
/* ----- */

```

```

/*      INCLUDE OBJ(FXXZCITF)      */
/*      INCLUDE OBJ(FXXZC)        */
/*      ENTRY FXXZCITF            */
/*      NAME EDITRF3(R)           */
/*                                */
/* NOTES:  Since this program is written in C, the entry point      */
/*          must be FXXZCITF, which is provided by the DI product.  */
/*          FXXZCITF establishes the necessary C environment        */
/*          and branches to the main entry point of the program.    */
/*                                */
/*          Parameters passed from DI may be above the line so this */
/*          program has to be 31 bit addressable                    */
/*                                */
/* User Exit:  This program is distributed as part of              */
/*             DataInterchange and, therefore, an entry in the     */
/*             ADAMCTL profile is not required. This program       */
/*             has a logical name of EDIBAUDO and a physical        */
/*             name of EDITRF3.                                     */
/*                                */
/* PARAMETERS:  IN - Service Name Block (snb)                      */
/*              Common Block (ccb)                                  */
/*              Function Block (fcb)                                */
/*              Filter handle (fh)                                  */
/*              Security data block (spdb)                          */
/*              Buffer size                                          */
/*              Buffer containing input data                         */
/*              Buffer for building output data                    */
/*              Length of data in input buffer                    */
/*              Number of characters remaining that would          */
/*              not fit into the input buffer                     */
/*                                */
/*              IN OUT - None                                       */
/*                                */
/*              OUT - CAS COMMON BLOCK                             */
/*                                */
/*              RC      ERC      Meaning                          */
/*              8       21      Invalid function code             */
/*              12      22      Filter failure occurred           */
/*              12      23      Input data in error               */
/*                                */
/* ----- */
/* Structure definitions used by this program                      */
/* ----- */
#include "disnb.h"          /* SNB definition */
#include "diccb.h"          /* CCB definition */
#include "difcb.h"          /* FCB definition */
#include "dispdb.h"         /* Security data block definition */
/* ----- */
/* Static data used by program                                     */
/* ----- */
static fcb getfcb={4,1};
static fcb putfcb={4,2};
static char hflag[]={0x00,0x80,0xc0,0xe0,0xf0,0xf8,0xfc,0xfe};
static char lflag[]={0x00,0x01,0x03,0x07,0x0f,0x1f,0x3f,0x7f,0xff};
static char ABaudtab[]={
    0x41,0x44,0x45,0x47,0x48,0x49,0x4A,0x4B,0x4C,
    0x4D,0x4F,0x50,0x51,0x52,0x53,

```

```

    0x55,0x56,0x57,0x58,0x59};
#define ABSIZE (sizeof(ABaudtab))

main(snbptr,ccbptr,fcptr,fh,spdbptr,
    bufsize,bufin,bufout,datalen,rbc)
    snb      *snbptr;    /* Service name block pointer set up by DI    */
                        /* to invoke this program                                */
    ccb      *ccbptr;    /* Common block pointer used by DI for all                          */
                        /* function requests                                                */
    fcptr    *fcptr;     /* Function block pointer which indicates                            */
                        /* the direction of the filter                                       */
                        /* 1=SEND, 2=RECEIVE                                                */
    long      *fh;        /* Filter handle which is used when GETTING                          */
                        /* more data or PUTTING filtered data                                */
    spdb      *spdbptr;   /* Security profile member that specified                            */
                        /* ASCII/BAUDOT filtering should occur                              */
    long      *bufsize;   /* The size of the input/output buffers                              */
    char      *bufin;     /* Buffer that contains the source data                              */
    char      *bufout;    /* Buffer that contains the filtered data                            */
    long      *datalen;   /* Length of data within the input buffer                            */
    long      *rbc;       /* The number of bytes remaining of the source*                      */
                        /* that would not fit into the input buffer                          */
{
    unsigned char *ss;    /* Local pointer to source data                                      */
    long    sl;          /* Length of data in source buffer                                  */
    char    *ds;         /* Local pointer to destination data                                */
    long    dl;          /* Length of data in destination buffer                             */
    int     bcr;         /* Number of bits left in current byte                              */
    int     bcw;         /* Number of bits wanted for source data                            */
    unsigned int iv,fv;   /* Index into the ABaudtab                                          */
    unsigned int i1,i2,i3,i4; /* Index values into ABaudtab                                       */
    int     bi;          /* Used to tell when 3/4 bytes are available                        */
    char    tv[4];       /* Holds four bytes of input                                        */
    int     done;        /* Processed finished when =1                                       */

    /* ----- */
    /* Set up local pointers to data buffers and initialize the      */
    /* lengths in each buffer.                                        */
    /* ----- */
    ss = bufin;
    sl = *datalen;
    ds = bufout;
    dl = 0;
    bcr=8;
    done=0;
    fv=0;
    /* ----- */
    /* Determine the direction of the filter and process accordingly. */
    /* ----- */
    switch (fcptr->zfcfunc) {
        case 0: /* EXISTENCE check */
            break;
        case 1: /* SEND filter operation */
            /* ----- */
            /* Begin processing every character of input, either in the */
            /* current input buffer or the residual that must be         */
            /* obtained by calling the GET/PUT service.                  */
            /* ----- */

```

```

/* ----- */
for (;!done;) {
/* ----- */
/* Flush the output buffer if it will not accept 4 more */
/* bytes. */
/* ----- */
if ((dl+4) > *bufsize) {
    fxxzc(fh,ccbpntr,&putfcb,bufout,&dl);
    if (ccbpntr->zccbrn) {
        ccbpntr->zccbrn = 12;
        ccbpntr->zccbern = 22;
        break;
    }
    ds = bufout;
    dl = 0;
}
/* ----- */
/* If you are picking up the final value (fv), then set */
/* the terminate flag; otherwise, get the next 13 bits */
/* from the input stream. */
/* ----- */
if (iv=fv)
    done = 1;
else for (bcw=13;bcw && !fv;) {
    if (bcw < bcr) {
/* ----- */
/* The number of bits required can be satisfied */
/* from the current byte. */
/* 1. Shift the value to accept the next bits. */
/* 2. Decrement number of bits remaining in byte. */
/* 3. Extract, align bits wanted, and incorporate. */
/* 4. Set the number of bits wanted to zero. */
/* ----- */
        iv = iv<<bcw;
        bcr = 8-bcw;
        iv = ((*ss&hflag[bcw])>>bcr);
        bcw = 0;
    } else {
/* ----- */
/* Make room the new bits, extract the remaining */
/* bits from the current byte, and incorporate */
/* into the result. */
/* ----- */
        iv = (iv<<bcr) + (*ss&lflag[bcr]);
        bcw -= bcr;
        bcr = 8;
        ss += 1;
        sl -= 1;
/* ----- */
/* Obtain more source data if the current source */
/* has been exhausted. */
/* ----- */
        if (!sl && *rbc) {
/* ----- */
/* Set -sl- to maximum size you can accept and */
/* call service to get more data. Return */
/* -sl- on will be the number of bytes returned.*/

```

```

/* -----*/
sl = *bufsize;
fxxzc(fh,ccbptr,&getfcb,bufin,&sl);
ss=bufin;
}
/* -----*/
/* If no data remains, then pad data to a multiple */
/* of 13 bits. */
/* -----*/
if (!sl && !*rbc) {
    if (bcw >= 4) {
        /* -----*/
        /* Pad the current value and set terminate */
        /* flag. */
        /* -----*/
        iv = (iv<<bcw)+(bcw-1);
        done = 1;
    } else {
        /* -----*/
        /* Pad current value with zeros and set the */
        /* value for the final 13 bits. */
        /* -----*/
        iv = iv << bcw;
        fv = 12+bcw;
    }
    break;
}
}
}
dl += 3;
if (iv < 380) {
    *ds++ = *ABaudtab;
    *ds++ = ABaudtab[(iv/ABSIZE)+1];
    *ds++ = ABaudtab[iv%ABSIZE];
} else if (iv < 7980) {
    iv -= 380;
    *ds++ = ABaudtab[(iv/(ABSIZE*ABSIZE))+1];
    *ds++ = ABaudtab[(iv/ABSIZE)%ABSIZE];
    *ds++ = ABaudtab[iv%ABSIZE];
} else {
    dl += 1;
    *ds++ = *ABaudtab;
    *ds++ = *ABaudtab;
    iv -= 7980;
    *ds++ = ABaudtab[iv/20];
    *ds++ = ABaudtab[iv%20];
}
}
break;
case 2: /* RECEIVE filter operation */
/* -----*/
/* Continue process as long as there is data in the current */
/* buffer or residual data. */
/* -----*/
bi = 0;
bcr = 8;
*ds = '\0';

```



```

for (; sl *rbc;) {
    if (!sl) {
        /* ----- */
        /* Obtain more data. Set -sl- to maximum amount you */
        /* can accept. On return, -sl- will contain the */
        /* number of bytes received. */
        /* ----- */
        sl = *bufsize;
        fxxzc(fh,ccbptr,&getfcb,bufin,&sl);
        ss=bufin;
    }
    /* ----- */
    /* Get 2 bytes from the input source. */
    /* ----- */
    tv[bi++]=*ss++;
    sl -= 1;
    if (bi < 3)
        continue;
    if ((bi==3)&&(*tv==*ABaudtab)&&*(tv+1)==*ABaudtab))
        continue;
    /* ----- */
    /* Determine the 13-bit value associated with 3/4 bytes. */
    /* ----- */
    i1 = getidx1(*tv);
    i2 = getidx1(*(tv+1));
    i3 = getidx1(*(tv+2));
    if (bi == 4) {
        i4 = getidx1(*(tv+3));
        iv = 7980+(i3*ABSIZE)+i4;
    } else if (*tv == *ABaudtab) {
        iv = ((i2-1)*ABSIZE)+i3;
    } else {
        iv = ((i1-1)*ABSIZE*ABSIZE)+(i2*ABSIZE)+i3+380;
    }
    bi = 0;
    /* ----- */
    /* Put the 13 bits into the output buffer. */
    /* ----- */
    if ((!sl) && (!*rbc)) {
        bcw = (iv&0x0f)+1;
        if (bcw>=13)
            break;
        else
            bcw=13-bcw;
    } else
        bcw=13;
    for (; bcw;) {
        if (bcr<=bcw) {
            *ds++ = (iv>>(13-bcr));
            dl += 1;
            bcw -= bcr;
            iv = iv<<bcr;
            bcr = 8;
            *ds = '\\0';
        } else {
            bcr = (8-bcw);
            *ds = iv>>5;

```

```

        bcw = 0;
    }
}
/* ----- */
/* Flush the output buffer if you have a complete byte */
/* and the buffer will not hold 13 more bytes of data. */
/* ----- */
if ((bcr==8) && ((dl+13)>*bufsize)) {
    fxxzc(fh,ccbptr,&putfcb,bufout,&dl);
    if (ccbptr->zccbrc) {
        ccbptr->zccbrc = 12;
        ccbptr->zccberc = 22;
        break;
    }
    ds = bufout;
    dl = 0;
    *ds = '\0';
}
}
/* ----- */
/* Set an error code if you terminated wanting data. */
/* ----- */
if (bi) {
    ccbptr->zccbrc = 12;
    ccbptr->zccberc = 23;
}
break;
default: /* UNKNOWN filter operation */
/* ----- */
/* Set the return codes and terminate. */
/* ----- */
ccbptr->zccbrc = 8;
ccbptr->zccberc = 21;
break;
}
/* ----- */
/* Flush data remaining in the output buffer. */
/* ----- */
if (!ccbptr->zccbrc && dl) {
    fxxzc(fh,ccbptr,&putfcb,bufout,&dl);
    /* ----- */
    /* Set the return code if there were any errors. */
    /* ----- */
    if (ccbptr->zccbrc) {
        ccbptr->zccbrc = 12;
        ccbptr->zccberc = 22;
    }
}
return;
}
/* ----- */
/* $MINOR */
/* ----- */
/* Routine Name: getidx1 (Return index value into AsciiTab) */
/* ----- */
/* Function: Determines the index value of a char into AsciiTab */
/* ----- */

```

```

/* Arguments:   char to search for          */
/*                                                    */
/* Return Values: index value                */
/*                                                    */
/* ----- */
int
getidx1(sc)
register int sc; /* Character to search for */
{
    register char *ts;
    register int ii;

    ts=ABaudtab;
    for (ii=0;(sc != *ts)&&(ii<ABSIZE);++ts,++ii);
    return (ii==ABSIZE)? 0:ii;
}

```

Authentication Examples

The following is the authentication exit routine that is distributed with DataInterchange. For more information on the authentication routines and the parameter definition required for various languages, see "Authentication Routine" on page 4-22. For more information on the CCB definition, see "C CCB" on page C-25, for the SNB definition, see "C SNB" on page C-25, for the FCB definition, see "C FCB" on page C-26, and for the SPDB definition, see "C SPDB" on page C-34.

Sample 1

```

/* ----- */
/* $MAJOR                                     */
/*                                                    */
/* Module Name: EDITRAA                       */
/*                                                    */
/* Descriptive Name: Authentication routine    */
/*                                                    */
/* STATUS:                                     */
/*                                                    */
/* Function:  This program is invoked by DI for the authentication */
/*            of data by calling the IBM 4753 Network Security      */
/*            Processor.                                           */
/*                                                    */
/* Dependencies: none                                             */
/*                                                    */
/* Restrictions: None                                             */
/*                                                    */
/* Language:   C                                                  */
/*                                                    */
/* Attributes:  Reentrant, AMODE(31) RMODE(ANY)                   */
/*              INCLUDE OBJ(EDITRAA)                             */
/*              INCLUDE OBJ(FXXZCITF)                             */
/*              INCLUDE OBJ(FXXZC)                               */
/*              ENTRY FXXZCITF                                    */
/*              NAME EDITRAA(R)                                   */
/*                                                    */
/* NOTES:  Since this program is written in C, the entry point    */
/*         must be FXXZCITF, which is provided by the DI product. */
/*         FXXZCITF establishes the necessary C environment       */

```

```

/*      and branches to the main entry point of the program.      */
/*      */
/*      Parameters passed from DI may be above the line so this    */
/*      program has to be 31 bit addressable                        */
/*      */
/* User Exit:      This program is distributed as part of          */
/*                  DataInterchange and, therefore, an entry in the */
/*                  ADAMCTL profile is not required. This program   */
/*                  has a logical name of IBMNSPA and a physical    */
/*                  name of EDITRAA.                                */
/*      */
/* PARAMETERS:      IN - Service Name Block (snb)                  */
/*                  Common Block (ccb)                             */
/*                  Function Block (fcb)                             */
/*                  Authentication handle (fh)                       */
/*                  Authentication key (ak)                           */
/*                  Security data block (spdb)                       */
/*                  Buffer size                                       */
/*                  Buffer containing input data                     */
/*                  Length of data in input buffer                 */
/*                  Number of characters remaining that would      */
/*                  not fit into the input buffer                  */
/*                  Address for return of the MAC value            */
/*      */
/*      IN OUT - None                                             */
/*      */
/*      OUT - CAS COMMON BLOCK                                     */
/*      */
/*      RC      ERC      Meaning                                   */
/*      8      11      Key not found                               */
/*      8      21      Invalid function code                       */
/*      8      24      Error getting data                          */
/*      */
/*      ----- */
/*      */
#include <stdefs.h>
#pragma linkage (CSNBMGN,OS)      /* NSP MAC Generate Routine */
#include "disnb.h"                 /* SNB definition */
#include "diccb.h"                 /* CCB definition */
#include "difcb.h"                 /* FCB definition */
#include "dispdb.h"               /* Security data block definition */

/* ----- */
/* Constant definitions used by this program */
/* ----- */
#define NULLPTR      (void *) 0      /* Null pointer */
#define EXISTS      0                /* Existence check */
#define MAC_GEN      1                /* Generate a MAC */
#define MAC_VER      2                /* Verify a MAC */
#define KEY_SIZE     16              /* Key size */
#define MAC_SIZE     4                /* MAC length */

/* ----- */
/* Static data used by program. */
/* ----- */

```

```

static fcb getfcb={4,1};          /* Used to get more data */
static char *MAC_METHOD = "X9.9-1 "; /* MAC X9.9-1 method */
static char *ONLY = "ONLY "; /* No segmenting */
static char *FIRST = "FIRST "; /* First segment of data */
static char *MIDDLE = "MIDDLE "; /* Middle segment of data */
static char *LAST = "LAST "; /* Last segment of data */
static char *MAC_LENGTH = "MACLEN4 "; /* MAC length of 4 bytes */

main(snbptr,ccbptra,fcbptra,fh,ak,spdbptr,
     bufsize,bufin,datalen,rbc,macval)
    snb *snbptr; /* Service name block pointer set up by DI */
               /* to invoke this program. */
    ccb *ccbptra; /* Common block pointer used by DI for all */
               /* function requests. */
    fcb *fcbptra; /* Function block pointer which indicates */
               /* the direction of the authentication. */
               /* 1=SEND(MAC_GEN), 2=RECEIVE(MAC_VER) */
    long *fh; /* Auth. handle which is used when GETTING */
               /* more data. */
    char *ak; /* 16 byte key name for the process. */
    spdb *spdbptr; /* Security profile member that specified */
               /* this program should be called. */
    long *bufsize; /* The size of the input buffer. */
    char *bufin; /* Buffer that contains the source data. */
    long *datalen; /* Length of data within the input buffer. */
    long *rbc; /* The number of source bytes remaining */
               /* that would not fit into the input buffer. */
    long *macval; /* Pointer where MAC value should return. */
{
    long rule_count; /* Rule array count for NSP */
    char rule_array[5][8]; /* Rule array for NSP */
    char min_in[8]; /* Minimum size buffer to work with */
    char *local_in; /* Local input buffer pointer */
    long local_dl; /* Length of data in local buffer */
    char key_token[64]; /* Key token for authentication */
    long local_long; /* Filler variable for NSP */
    char chain_vect[18]; /* Chaining vector */
    char local_macval[8]; /* Must pass NSP routine 8 byte area */
    char *rule; /* Current rule to follow */

    /* ----- */
    /* Determine what you are asked to do and process. */
    /* ----- */
    switch(fcbptr->zfcfunc)
    {
        case EXISTS:
            /* ----- */
            /* Existence check just returns with success. */
            /* ----- */
            break;
        case MAC_GEN:
        case MAC_VER:
            /* ----- */
            /* Just generate a MAC value both times and let */
            /* DI handle the verification. */
            /* ----- */

```

```

/* ----- */
/* Set up some initial values. */
/* ----- */
rule_count = 3; /* Method, control, length*/
memset(min_in, ' ', sizeof(min_in));
memset(chain_vect, 0x00, sizeof(chain_vect));
memset(rule_array, ' ', sizeof(rule_array));
memcpy(rule_array[0], MAC_METHOD, sizeof(rule_array[0]));
memcpy(rule_array[2], MAC_LENGTH, sizeof(rule_array[2]));
memset(key_token, ' ', sizeof(key_token));
memcpy(key_token, ak, KEY_SIZE);

/* ----- */
/* Set up local pointers and sizes. You want to */
/* ensure that you have a buffer of at least 8 */
/* bytes. */
/* ----- */
local_dl = *datalen;
if (*bufsize < 8)
{
    memcpy(min_in, bufin, *bufsize);
    local_in = min_in;
}
else
    local_in = bufin;

/* ----- */
/* In order to chain the data, you have to have */
/* at least 8-bytes for the -FIRST- segment. */
/* ----- */
if (*rbc)
{
    if (local_dl < 8)
    {
        /* ----- */
        /* Attempt to fill the buffer up. */
        /* ----- */
        local_long = 8 - local_dl;
        fxxzc(fh, ccbptr, &getfcb,
            &local_in[local_dl], &local_long);
        if (ccbptr->zccbrc)
        {
            ccbptr->zccbrc = 8; /* Error ... */
            ccbptr->zccberc = 24; /* Error getting data */
            return;
        }
        local_dl += local_long;
    }
    /* ----- */
    /* If you have more bytes left, you should */
    /* have filled the buffer up to eight for */
    /* the -FIRST- segment. */
    /* ----- */
    rule = (*rbc) ? FIRST : ONLY;
}
else
    rule = ONLY;

```

```

memcpy(rule_array[1], rule, sizeof(rule_array[1]));

/* ----- */
/* Call the MAC generate verb. */
/* ----- */

CSNBMGN(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
        NULLPTR, key_token, &local_dl, local_in, &rule_count,
        rule_array, chain_vect, local_macval);

if (ccbptr->zccbrc) /* Error? ... */
{
    if (ccbptr->zccberc == 30) /* If key not found ... */
        ccbptr->zccberc = 11; /* DI wants this */
    return;
}

while (*rbc)
{
    /* ----- */
    /* Get some more data. */
    /* ----- */
    fxxzc(fh, ccbptr, &getfcb, local_in, &local_dl);
    if (ccbptr->zccbrc)
    {
        ccbptr->zccbrc = 8; /* Error ... */
        ccbptr->zccberc = 24; /* Error getting data */
        return;
    } /* end if */

    /* ----- */
    /* Reset the segmenting control. */
    /* ----- */
    memcpy(rule_array[1], ((*rbc) ? MIDDLE : LAST),
           sizeof(rule_array[1]));

    /* ----- */
    /* Call the MAC generate verb. */
    /* ----- */

    CSNBMGN(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, key_token, &local_dl, local_in, &rule_count,
            rule_array, chain_vect, local_macval);

    if (ccbptr->zccbrc) /* Error? ... */
    {
        if (ccbptr->zccberc == 30) /* If key not found ... */
            ccbptr->zccberc = 11; /* DI wants this */
        return;
    }

    } /* end while */
    memcpy(macval, local_macval, MAC_SIZE); /* Copy 4-byte MAC */
    break;
default:
    /* ----- */
    /* Wrong function code. Set error and return. */
    /* ----- */

```

```

        /* ----- */
        ccbptr->zccbrc = 8;          /* Error ... */
        ccbptr->zccberc = 21;       /* Invalid function code */
        break;
    } /* end switch */
return;
} /* end main */

```

Sample 2

```

/* ----- */
/* $MAJOR */
/*
/* Module Name: CCAA */
/*
/* Descriptive Name: Authentication routine */
/*
/* STATUS: */
/*
/* Function: This program is an example of an authentication
/*           routine which uses the IBM Common Cryptographic
/*           Architecture Cryptographic Application Programming
/*           Interface as defined in reference SC40-1675.
/*
/*           This program is invoked by DataInterchange (DI)
/*           during the enveloping or de-enveloping process
/*
/* Dependencies: none */
/*
/* Restrictions: None */
/*
/* Language:      C */
/*
/* Attributes:   Reentrant, AMODE(31) RMODE(ANY)
/*               INCLUDE OBJ(CCAA)
/*               INCLUDE OBJ(FXXZCITF)
/*               INCLUDE OBJ(FXXZC)
/*               ENTRY FXXZCITF
/*               NAME CCAA(R)
/*
/* NOTES:  Since this program is written in C, the entry point
/*          must be FXXZCITF, which is provided by the DI product.
/*          FXXZCITF establishes the necessary C environment
/*          and branches to the main entry point of the program.
/*
/*          Parameters passed from DI may be above the line so this
/*          program has to be 31 bit addressable
/*
/*
/* PARAMETERS:  IN - Service Name Block (snb)
/*               Common Block (ccb)
/*               Function Block (fcb)
/*               Authentication handle (fh)
/*               Authentication key (ak)
/*               Security data block (spdb)
/*               Buffer size
/*               Buffer containing input data

```



```

/*          Length of data in input buffer          */
/*          Number of characters remaining that would */
/*          not fit into the input buffer            */
/*          Address for return of the MAC value      */
/*          */
/*          IN OUT - None                            */
/*          */
/*          OUT - CAS COMMON BLOCK                   */
/*          */
/*          RC      ERC      Meaning                */
/*          8       21      Invalid function code    */
/*          8       24      Error getting data       */
/*          */
/*          * As defined by the processor implementing the CCA */
/*          */
/* ----- */

#include <stdefs.h>
#pragma linkage (CSNBMGN,OS)      /* MAC Generate Routine */
#include "disnb.h"                 /* SNB definition        */
#include "diccb.h"                 /* CCB definition        */
#include "difcb.h"                 /* FCB definition        */
#include "dispdb.h"                /* Security data block definition */

/* ----- */
/* Constant definitions used by this program          */
/* ----- */
#define NULLPTR      (void *) 0      /* Null pointer          */
#define EXISTS       0               /* Existence check       */
#define MAC_GEN       1              /* Generate a MAC        */
#define MAC_VER       2              /* Verify a MAC          */
#define KEY_SIZE      16             /* Key size              */
#define MAC_SIZE      4              /* MAC length            */

/* ----- */
/* Static data used by program                        */
/* ----- */
static fcb getfcb={4,1};           /* Used to get more data */
static char *MAC_METHOD = "X9.9-1 "; /* MAC X9.9-1 method    */
static char *ONLY       = "ONLY ";   /* No segmenting         */
static char *FIRST      = "FIRST ";  /* First segment of data */
static char *MIDDLE     = "MIDDLE "; /* Middle segment of data */
static char *LAST       = "LAST ";   /* Last segment of data  */
static char *MAC_LENGTH = "MACLEN4 "; /* MAC length of 4 bytes */

main(snbptr,ccbpntr,fcbptr,fh,ak,spdbptr,
     bufsize,bufin,datalen,rcb,macval)
     snb      *snbptr;    /* Service name block pointer set up by DI */
     ccb      *ccbpntr;   /* Common block pointer used by DI for all */
     fcb      *fcbptr;    /* Function block pointer which indicates */
     long     *fh;        /* Auth. handle which is used when GETTING */

```

```

char    *ak;           /* 16 byte key name for the process      */
spdb    *spdbptr;      /* Security profile member that specified */
                        /* this program should be called         */
long    *bufsize;      /* The size of the input buffer          */
char    *bufin;        /* Buffer that contains the source data   */
long    *datalen;      /* Length of data within the input buffer */
long    *rbc;          /* The number of source bytes remaining  */
                        /* that would not fit into the input buffer */
long    *macval;       /* Pointer where MAC value should be returned*/

{
    long    rule_count; /* Rule array count                      */
    char    rule_array[5][8]; /* Rule array                          */
    char    min_in[8];   /* Minimum size buffer to work with     */
    char    *local_in;   /* Local input buffer pointer           */
    long    local_dl;    /* Length of data in local buffer       */
    char    key_token[64]; /* Key token for authentication         */
    long    local_long;   /* Filler variable                     */
    char    chain_vect[18]; /* Chaining vector                     */
    char    local_macval[8]; /* Must pass routine 8 byte area       */
    char    *rule;       /* Current rule to follow               */

/* ----- */
/* Do some special processing if this is an GEN or VER request. */
/* ----- */
if ((fcbptr->zfcbfnc == MAC_GEN)
    (fcbptr->zfcbfnc == MAC_VER)) {
    /* ----- */
    /* The key passed to this routine by DI (ak) is a 16-byte key */
    /* name that is taken from either the DI mapping records for */
    /* generation or the S1S or S2S security segments for        */
    /* validation. The CCA MACgen and MACver calls require that   */
    /* a CCA MAC key be provided as a key token value. It is your */
    /* responsibility to write code to handle the storage         */
    /* of internal key tokens in a key storage data set so that an */
    /* association can be made between the name of a key (key label) */
    /* and the value of that key (CCA MAC key value). You        */
    /* must also provide a key management routine that can be called */
    /* from this routine. The function of this key management routine*/
    /* would be to accept a 16-byte key label and type as input and */
    /* return as output a 64-byte key identifier. For more       */
    /* information on key labels, key types, and key identifiers, see */
    /* "IBM Common Cryptographic Architecture Cryptographic      */
    /* Application Programming Interface Reference (SC40-1675)"   */
    /* ----- */
    /* If your security subsystem is the Integrated Cryptographic */
    /* Feature (ICRF) with the Integrated Cryptographic Service   */
    /* Facility / MVS (ICSF/MVS), you should replace "key_transform" */
    /* by a routine that calls whatever program your installation */
    /* uses to manage DATA keys and MAC keys. For more information */
    /* on ICSF/MVS, see the following ICSF/MVS publications:      */
    /* "General Information (GC23-0093)"                          */
    /* "Administrator's Guide (SC23-0097)"                      */
    /* "Application Programmer's Guide (SC23-0098)"             */
    /* "System Programmer's Guide (SC23-0096)"                  */
    /* ----- */
    memset(key_token, ' ', sizeof(key_token));

```

```

    key_transform(ak,"MAC      ",key_token);
}
/* ----- */
/* Determine what you are asked to do and process. */
/* ----- */
switch(fcbptr->zfcfunc)
{
    case EXISTS:
        /* ----- */
        /* Existence check returns with success. */
        /* ----- */
        break;
    case MAC_GEN:
    case MAC_VER:
        /* ----- */
        /* Generate a MAC value both times and let */
        /* DI handle the verification. */
        /* ----- */
        /* ----- */
        /* Set up some initial values. */
        /* ----- */
        rule_count = 3; /* Method, control, length */
        memset(min_in, ' ', sizeof(min_in));
        memset(chain_vect, 0x00, sizeof(chain_vect));
        memset(rule_array, ' ', sizeof(rule_array));
        memcpy(rule_array[0], MAC_METHOD, sizeof(rule_array[0]));
        memcpy(rule_array[2], MAC_LENGTH, sizeof(rule_array[2]));
        memcpy(key_token, ak, KEY_SIZE);

        /* ----- */
        /* Set up local pointers and sizes. You want to */
        /* ensure that you have a buffer of at least 8 */
        /* bytes. */
        /* ----- */
        local_dl = *datalen;
        if (*bufsize < 8)
        {
            memcpy(min_in, bufin, *bufsize);
            local_in = min_in;
        }
        else
            local_in = bufin;

        /* ----- */
        /* In order to chain the data, you have to have */
        /* at least 8 bytes for the -FIRST- segment. */
        /* ----- */
        if (*rbc)
        {
            if (local_dl < 8)
            {
                /* ----- */
                /* Attempt to fill the buffer up. */
                /* ----- */
                local_long = 8 - local_dl;
                fxxzc(fh, ccbptr, &getfcb,
                    &local_in[local_dl], &local_long);
            }
        }
    }
}

```

```

        if (ccbptr->zccbrc)
        {
            ccbptr->zccbrc = 8;          /* Error ...          */
            ccbptr->zccberc = 24;       /* Error getting data */
            return;
        }
        local_dl += local_long;
    }
    /* ----- */
    /* If you have more bytes left, you should          */
    /* have filled the buffer up to 8 for              */
    /* the -FIRST- segment.                            */
    /* ----- */
    rule = (*rbc) ? FIRST : ONLY;
}
else
    rule = ONLY;
memcpy(rule_array[1], rule, sizeof(rule_array[1]));

/* ----- */
/* Call the MAC generate verb.                        */
/* ----- */

CSNBMGN(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
        NULLPTR, key_token, &local_dl, local_in, &rule_count,
        rule_array, chain_vect, local_macval);

if (ccbptr->zccbrc)          /* Error? ...          */
    return;

while (*rbc)
{
    /* ----- */
    /* Get some more data.                            */
    /* ----- */
    fxxzc(fh, ccbptr, &getfcb, local_in, &local_dl);
    if (ccbptr->zccbrc)
    {
        ccbptr->zccbrc = 8;          /* Error ...          */
        ccbptr->zccberc = 24;       /* Error getting data */
        return;
    } /* end if */

    /* ----- */
    /* Reset the segmenting control.                  */
    /* ----- */
    memcpy(rule_array[1], ((*rbc) ? MIDDLE : LAST),
           sizeof(rule_array[1]));

    /* ----- */
    /* Call the MAC generate verb.                    */
    /* ----- */

    CSNBMGN(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, key_token, &local_dl, local_in, &rule_count,
            rule_array, chain_vect, local_macval);

```

```

        if (ccbptr->zccbrc)                /* Error? ...      */
            return;

        } /* end while */
        memcpy(macval,local_macval,MAC_SIZE); /* Copy 4-byte MAC  */
        break;
    default:
        /* ----- */
        /* Wrong function code. Set error and return.      */
        /* ----- */
        ccbptr->zccbrc = 8;                /* Error ...      */
        ccbptr->zccberc = 21;             /* Invalid function code */
        break;
    } /* end switch */
return;
} /* end main */

```

Encryption Examples

The following is the encryption exit routine that is distributed with DataInterchange. For more information on the encryption routines and the parameter definition required for various languages, see "Encryption Routine" on page 4-18. For more information on the CCB definition, see "C CCB" on page C-25, for the SNB definition, see "C SNB" on page C-25, for the FCB definition, see "C FCB" on page C-26, and for the SPDB definition, see "C SPDB" on page C-34.

Sample 1

```

/* ----- */
/* $MAJOR                                     */
/*                                           */
/* Module Name: EDITREE                     */
/*                                           */
/* Descriptive Name: Encryption/Decryption routine */
/*                                           */
/* Status:                                   */
/*                                           */
/* Function: This program is invoked for the encryption/decryption */
/*           of data by calling the IBM 4753 Network Security      */
/*           Processor.                                           */
/*                                           */
/* Dependencies: none                                             */
/*                                           */
/* Restrictions: none                                             */
/*                                           */
/* Language:      C                                              */
/*                                           */
/* Attributes:    Reentrant, AMODE(31) RMODE(ANY)                 */
/*                INCLUDE OBJ(EDITREE)                           */
/*                INCLUDE OBJ(FXXZCITF)                           */
/*                INCLUDE OBJ(FXXZC)                             */
/*                ENTRY FXXZCITF                                  */
/*                NAME EDITREE(R)                                 */
/*                                           */
/* Notes:         Since this program is written in C, the entry   */
/*                point must be FXXZCITF, which is provided by the */

```

```

/*      DI product. FXXZCITF establishes the necessary      */
/*      C environment and branches to the main entry point */
/*      of the program.                                     */
/*                                                         */
/*      Parameters passed from DI may be above the line so */
/*      this program has to be 31 bit addressable.         */
/*                                                         */
/* User Exit:      This program is distributed as part of   */
/*                 DataInterchange and, therefore, an entry in the */
/*                 ADAMCTL profile is not required. This program */
/*                 has a logical name of IBMNSPE and a physical */
/*                 name of EDITREE.                           */
/*                                                         */
/* Parameters:  IN - Service Name Block   (snb)             */
/*               Common Block             (ccb)             */
/*               Function Block            (fcb)             */
/*               Encryption Handle         (fh)              */
/*               Encryption key value      (ek)              */
/*               Security Data Block       (spdb)            */
/*               Buffer size                */
/*               Buffer containing input data */
/*               Buffer containing output data */
/*               Length of data in input buffer */
/*               Number of characters remaining that would */
/*               not fit into the input buffer */
/*               Initialization vector return address */
/*                                                         */
/*      IN OUT - None                                     */
/*                                                         */
/*      OUT - CAS Common Block                           */
/*                                                         */
/*      RC      ERC      Meaning                       */
/*      0       0       Success                         */
/*      8       21      Invalid function code            */
/*      8       11      Keyname not known                */
/*      *       *       *                               */
/*                                                         */
/*      * See "Transaction Security System Programming    */
/*      Guide and Reference" (SC31-2934) for others.     */
/*                                                         */
/* ----- */

#include "stdefs.h"
#include "disnb.h"          /* SNB definition      */
#include "diccb.h"          /* CCB definition      */
#include "difcb.h"          /* FCB definition      */
#include "dispdb.h"         /* Security data block definition */
#pragma linkage (CSNBRNG,OS) /* NSP Random Number Generate */
#pragma linkage (CSNBENC,OS) /* NSP Encipher routine */
#pragma linkage (CSNBDEC,OS) /* NSP Decipher routine */

/* ----- */
/* Constant definitions used by this program */
/* ----- */
#define EXISTS    0          /* Existence check */
#define ENCRYPT    1          /* Encrypt data */

```

```

#define DECRYPT 2          /* Decrypt data */
#define GETIV 3           /* Get initialization vector */
#define CBC_TYPE '1'      /* CBC ciphering type */
#define IV_SIZE 8         /* Initialization vector size */
#define KEY_SIZE 16       /* Input key size */
#define NULLPTR (void *) 0 /* NULL pointer */
#define True 1
#define False 0

/* ----- */
/* Static data used by this program */
/* ----- */
static fcb getfcb = {4,1}; /* Used to get more data */
static fcb putfcb = {4,2}; /* Used to put data */
static char *RANDOM = "RANDOM "; /* Used for random number */
static char *CBC_METH = "CBC "; /* CBC ciphering method */

/* ----- */
/* Local function prototypes */
/* ----- */
void pad_buff(char *buff, long data_len, long *crypt_len);

/* ----- */
/* Main entry point begins here. */
/* ----- */
main(snbptr, ccbptr, fcbptr, fh, ek, spdbptr,
     bufsize, bufin, bufout, datalen, rbc, iv)

    snb *snbptr; /* Service name block pointer set up by DI */
    ccb *ccbptr; /* Common block pointer used by DI */
    fcb *fcbptr; /* Function block pointer which indicates
                 /* what to do:
                 /* 0=EXISTS, 1=ENCRYPT, 2=DECRYPT, 3=GETIV
    long *fh; /* Encryption handle - used with GET and PUT */
    char *ek; /* Encryption key name */
    spdb *spdbptr; /* Security profile member */
    long *bufsize; /* The size of the input/output buffers */
    char *bufin; /* Buffer that contains the source data */
    char *bufout; /* Buffer for the output data */
    long *datalen; /* Length of data in the input buffer */
    long *rbc; /* The number of bytes remaining of the source*
              /* that would not fit into the input buffer
    char *iv; /* Initialization vector return area

{
    long rule_count; /* Rule array count for NSP */
    char rule_array[8]; /* Rule array for NSP */
    char form[8]; /* Form of random number */
    char min_in[16]; /* Minimum size buffer to work with */
    char min_out[16]; /* Minimum output buffer to work out */
    char *local_in; /* Local input buffer pointer */
    char *local_out; /* Local output buffer pointer */
    long local_bs; /* Size of local buffer */
    long local_dl; /* Length of data in local buffer */
    char key_token[64]; /* Key token for encrypt/decrypt */
    char siv[IV_SIZE]; /* Save original IV value */
    long filler; /* Filler variable for NSP */
    long bytes_left; /* Number bytes remaining after input */

```

```

/* buffer adjusted for encryption */
long crypt_in; /* Encryption/decryption length */
long crypt_len; /* Encryption/decryption length */
char chain_vect[18]; /* Chaining vector */
int padded = False; /* Loop exit variable */

/* ----- */
/* Set up some initial values. */
/* ----- */
rule_count = 1;
memset(min_in, ' ', sizeof(min_in));
memset(min_out, ' ', sizeof(min_out));
memset(rule_array, ' ', sizeof(rule_array));
memcpy(rule_array, CBC_METH, sizeof(rule_array));
memset(key_token, ' ', sizeof(key_token));
memcpy(key_token, ek, KEY_SIZE);

/* ----- */
/* Determine what you are asked to do and process. */
/* ----- */
switch(fcbptr->zfcfunc)
{
case EXISTS:
    / ----- */
    /* Existence check returns with success. */
    / ----- */
    break;
case GETIV:
    / ----- */
    /* Call the Random number generate verb to get */
    /* a random number. Note: The random number is */
    /* not encrypted at this time. Since you are */
    /* passing the return and extended return codes, */
    /* you are not checking for errors here. */
    / ----- */
    memcpy(form, RANDOM, sizeof(form));
    CSNBRNG(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, form, iv);
    break;
case ENCRYPT:
    / ----- */
    /* Check for the correct ciphering method. You */
    /* only do CBC here. */
    / ----- */
    if (spdbptr->encrtype != CBC_TYPE)
    {
        ccbptr->zccbrc = 8; /* Error ... */
        ccbptr->zccberc = 22; /* Invalid encrypt type */
        return;
    }

    / ----- */
    /* Set up local pointers and sizes. */
    /* Minimum size buffer to work with is 16. */
    / ----- */
    if (*bufsize < 16)

```



```

    {
        memcpy(min_in, bufin, *bufsize);
        local_in = min_in;
        local_out = min_out;
        local_bs = sizeof(min_in);
    }
else
    {
        local_in = bufin;
        local_out = bufout;
        local_bs = *bufsize;
    }
local_dl = *datalen;
memcpy(siv,iv,IV_SIZE);
/* ----- */
/* Now that your working buffer size is right, */
/* you may have gotten data in that is less then */
/* 8 bytes. First attempt to fill the buffer up */
/* to 8 bytes. If that doesn't work, pad it. */
/* ----- */
if (local_dl < 8)
{
    if (*rbc) /* Any bytes not passed? */
    {
        filler = 8 - local_dl;
        fxxzc(fh,ccbptra,&getfcb, &local_in[local_dl],&filler);
        if (ccbptra->zccbrca)
        {
            ccbptr->zccbrca = 8; /* Error ... */
            ccbptr->zccberca = 24; /* Error getting data */
            return;
        }
        if ((filler+local_dl) < 8)
        {
            pad_buff(local_in,filler+local_dl,&local_bs);
            padded = True;
        }
        local_dl = 8; /* Either padded or filled */
    }
else
    {
        pad_buff(local_in, local_dl, &local_bs);
        local_dl = 8;
        padded = True;
    } /* end if(rbc) */
} /* end if(local_dl) */

/* ----- */
/* Adjust the buffer length to a multiple of 8 */
/* and set the number of bytes that will not be */
/* processed the first pass. */
/* You have to have at least 8 bytes at the end */
/* of the buffer because of the way the NSP */
/* routine treats the buffer. */
/* ----- */
bytes_left = local_dl % 8;
crypt_len = local_dl - (bytes_left);

```

```

if (crypt_len > 8)
{
    crypt_len -= 8;
    bytes_left += 8;
}
crypt_in = crypt_len;

/* ----- */
/* Start looping until all data is processed. */
/* You know you are finished when some type of */
/* padding has been done. */
/* ----- */
for(;;)
{
    /* ----- */
    /* Call the encipher verb. */
    /* ----- */
    filler = 0; /* Set the Pad character*/
    crypt_len = crypt_in;
    CSNBENC(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, key_token, &crypt_len, local_in, iv,
            &rule_count, rule_array, &filler, chain_vect,
            local_out);

    if (ccbptr->zccbrc) /* Error? ... */
    {
        if (ccbptr->zccberc == 30) /* If key not found ... */
            ccbptr->zccberc = 11; /* DI wants this */
        return;
    }

    /* ----- */
    /* Put the data out. */
    /* ----- */
    fxxzc(fh, ccbptr, &putfcb, local_out, &crypt_len);
    if (ccbptr->zccbrc)
    {
        ccbptr->zccbrc = 8; /* Error ... */
        ccbptr->zccberc = 23; /* Error putting data */
        return;
    }

    /* ----- */
    /* Check to see if any padding has been done. */
    /* If so, break out of the loop. */
    /* ----- */
    if (padded)
        break;

    /* ----- */
    /* Ensure that any bytes not processed in the */
    /* first pass are accounted for. */
    /* ----- */
    if (bytes_left)
        memcpy(local_in, local_in+crypt_in, bytes_left);

    /* ----- */

```

```

        /* Check for any data that was not passed to          */
        /* you in the original input buffer.                  */
        /* ----- */
if (bytes_left >= crypt_in)
{
    bytes_left -= crypt_in;
}
else
{
    if (*rbc)
    {
        filler = crypt_in - bytes_left;
        fxxzc(fh,ccbptra,&getfcb,
            &local_in[bytes_left],&filler);
        if (ccbptra->zccbrc)
        {
            ccbptra->zccbrc = 8; /* Error ...          */
            ccbptra->zccberc = 24; /* Error getting data */
            return;
        }
    }
    else
        filler = 0;
    /* ----- */
    /* Check to see if any padding needs to be done. */
    /* ----- */
    if ((filler+bytes_left) < crypt_in)
    {
        pad_buff(local_in, filler+bytes_left, &crypt_in);
        padded = True;
    }

    bytes_left = 0;          /* Set back to zero    */
}

/* ----- */
/* Change the Initialization vector to continue.          */
/* ----- */
memcpy(iv,chain_vect,IV_SIZE);
} /* end for(;;) */

/* ----- */
/* Encrypt the IV using ECB. The CBC method with an IV of */
/* 0 and 8 bytes of data is the same.                      */
/* ----- */
crypt_len = IV_SIZE;
memset(iv,0x00,IV_SIZE);
CSNBENC(&(ccbptra->zccbrc), &(ccbptra->zccberc), NULLPTR,
    NULLPTR, key_token, &crypt_len, siv, iv,
    &rule_count, rule_array, &filler, chain_vect, min_out);
memcpy(iv,min_out,IV_SIZE);

if (ccbptra->zccbrc)          /* Error? ...          */
{
    if (ccbptra->zccberc == 30) /* If key not found ... */
        ccbptra->zccberc = 11; /* DI wants this      */
}

```

```

        return;
    }

    break;
case DECRYPT:
    /* ----- */
    /* Check for the correct ciphering method, You */
    /* only do CBC here. */
    /* ----- */
    if (spdbptr->encrtype != CBC_TYPE)
    {
        ccbptr->zccbrc = 8; /* Error ... */
        ccbptr->zccberc = 22; /* Invalid encrypt type*/
        return;
    }

    /* ----- */
    /* Encrypted data MUST be a multiple of 8. */
    /* ----- */
    if (((*rbc + *datalen)%8 != 0) ((*rbc + *datalen) <= 0))
    {
        ccbptr->zccbrc = 8; /* Error ... */
        ccbptr->zccberc = 26; /* Input data error */
        return;
    }

    /* ----- */
    /* Call the decipher verb to decipher the IV. */
    /* You are using the CBC method with an IV of */
    /* 0 and data length of 8, which is the same as */
    /* the ECB method. */
    /* ----- */
    crypt_len = IV_SIZE; /* Length of the IV */
    memcpy(min_in, iv, IV_SIZE); /* Copy IV over for input*/
    memset(form, 0x00, sizeof(form)); /* Used as the IV */

    CSNBDEC(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, key_token, &crypt_len, min_in, form,
            &rule_count, rule_array, chain_vect, iv);

    if (ccbptr->zccbrc) /* Error? ... */
    {
        if (ccbptr->zccberc == 30) /* If key not found ... */
            ccbptr->zccberc = 11; /* DI wants this */
        return;
    }

    /* ----- */
    /* Set up local pointers and sizes. */
    /* Minimum size buffer to work with is 8. */
    /* ----- */
    local_dl = *datalen;
    if (*bufsize < 8)
    {
        memcpy(min_in, bufin, *bufsize);
        local_in = min_in;
        local_out = min_out;
    }

```

```

        local_bs = sizeof(min_in);
    }
else
{
    local_in = bufin;
    local_out = bufout;
    local_bs = *bufsize;
} /* end if */

/* ----- */
/* Now that your working buffer size is right, */
/* you may have gotten data in that is less than */
/* 8 bytes. Encrypted data MUST be a multiple of */
/* of 8 bytes. So, if you didn't get 8, */
/* there must be more out there to get. (You */
/* checked for a multiple of 8 in the beginning.) */
/* ----- */
if (local_d1 < 8)
{
    filler = 8 - local_d1; /* Get only up to 8 */
    fxxzc(fh,ccbptr,&getfcb, &local_in[local_d1],&filler);
    if (ccbptr->zccbrc)
    {
        ccbptr->zccbrc = 8; /* Error ... */
        ccbptr->zccberc = 24; /* Error getting data */
        return;
    }
    local_d1 = 8;
}

/* ----- */
/* Adjust the buffer length to a multiple of 8 */
/* and set the number of bytes that will not be */
/* processed the first pass. If you were passed */
/* a buffer length that was not a multiple of */
/* 8, you will process only the highest multiple */
/* you can get on the first call to decipher, */
/* then any bytes left with the residual not */
/* passed to us. */
/* ----- */
bytes_left = local_d1 % 8;
crypt_len = local_d1 - bytes_left;

/* ----- */
/* Start looping until all data is processed. */
/* You break out of this loop when you determine */
/* that all characters have been processed and */
/* subtracted the number of pad characters. */
/* ----- */
for(;;)
{
    /* ----- */
    /* Call the decipher verb. */
    /* ----- */
    crypt_in = crypt_len;
    CSNBDEC(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,

```

```

        NULLPTR, key_token, &crypt_len, local_in, iv,
        &rule_count, rule_array, chain_vect, local_out);

if (ccbptr->zccbrc)                /* Error? ... */
{
    if (ccbptr->zccberc == 30)      /* If key not found ... */
        ccbptr->zccberc = 11;      /* DI wants this */
    return;
}

/* ----- */
/* Put the data out. If no data is left to pro- */
/* cess, the last byte will contain the number */
/* of pad characters in ASCII. When this is */
/* encountered, you exit the loop. */
/* ----- */
if (*rbc)
{
    fxxzc(fh, ccbptr, &putfcb, local_out, &crypt_len);
    if (ccbptr->zccbrc)
    {
        ccbptr->zccbrc = 8;        /* Error ... */
        ccbptr->zccberc = 23;      /* Error putting data */
        return;
    } /* end if */
}
else
{
    /* ----- */
    /* The last position must be the number */
    /* of pad characters in ASCII. If this */
    /* is not true, exit in error. */
    /* ----- */
    filler = local_out[crypt_len-1] & 0x0F;
    if ((filler < 1) (filler > 8))
    {
        ccbptr->zccbrc = 8;        /* Error ... */
        ccbptr->zccberc = 26;      /* Input data error */
        return;
    } /* end if */

    /* ----- */
    /* Flush output minus pad characters. */
    /* ----- */
    crypt_len = crypt_len - filler;
    fxxzc(fh, ccbptr, &putfcb, local_out, &crypt_len);
    if (ccbptr->zccbrc)
    {
        ccbptr->zccbrc = 8;        /* Error ... */
        ccbptr->zccberc = 23;      /* Error putting data */
        return;
    } /* end if */
    break;                        /* Exit the loop */
} /* end if */

```

```

/* ----- */
/* Ensure that any bytes not processed in the */
/* first pass are accounted for. This only */
/* needs to be done for the first pass. */
/* ----- */
if (bytes_left)
    memcpy(local_in, local_in+crypt_in, bytes_left);

/* ----- */
/* Get any data that was not passed to you in */
/* the original input buffer. You wouldn't have */
/* gotten this far if there wasn't anything */
/* there. */
/* ----- */
filler = crypt_in - bytes_left;
fxxzc(fh,ccbptr,&getfcb, &local_in[bytes_left],&filler);
if (ccbptr->zccbrc)
{
    ccbptr->zccbrc = 8;          /* Error ... */
    ccbptr->zccberc = 24;       /* Error getting data */
    return;
}
crypt_len = filler + bytes_left;
bytes_left = 0;               /* Set back to zero */

/* ----- */
/* Change the initialization vector for the next call. */
/* ----- */
memcpy(iv,chain_vect,IV_SIZE);
} /* end for(;;) */
break;
default:
/* ----- */
/* Wrong function code. Set error and return. */
/* ----- */
ccbptr->zccbrc = 8;          /* Error ... */
ccbptr->zccberc = 21;       /* Invalid function code */
break;
} /* end switch */
return;
} /* end main */

/* ----- */
/* Routine: pad_buff */
/* ----- */
/* Purpose: Pad a buffer with ASCII 0s and put the number of places */
/* padded in the last position. */
/* ----- */
/* In      : buff      - The buffer to pad */
/*           data_len - The length of data in the buffer */
/*           crypt_len - Current maximum buff size */
/* ----- */
/* Out     : Modified buffer (buff) and data length (crypt_len) */
/* ----- */
void pad_buff(buff, data_len, crypt_len)
    char *buff;               /* Input buffer to pad */

```

```

    long  data_len;           /* Current data length in buffer */
    long  *crypt_len;        /* Current cipher length of buffer*/

{
    long num_pad = 0;        /* Number of characters padded */

    /* ----- */
    /* If the buffer length is on a 8-byte boundary (Including */
    /* zero), force the first pad. 'crypt_len' is always on an */
    /* 8-byte boundary and 'data_len' is always less than it, so */
    /* you can never overflow your buffer. */
    /* ----- */
    if ((data_len%8) == 0)
    {
        buff[data_len++] = 0x30;    /* Force the first pad */
        num_pad = 1;
    } /* end if */

    /* ----- */
    /* Pad with ASCII 0s. */
    /* ----- */
    for (;data_len%8;num_pad++, ++data_len)
        buff[data_len] = 0x30;

    /* ----- */
    /* Put number of pad characters in ASCII in last position */
    /* and change the crypt length to new buffer size. */
    /* ----- */
    buff[data_len-1] = (num_pad & 0x0F) 0x30;
    *crypt_len = data_len;
} /* end pad_buff() */

```

Sample 2

```

/* ----- */
/* $MAJOR */
/*
/* Module Name: CCAEE */
/*
/* Descriptive Name: Encryption/Decryption routine */
/*
/* Status: */
/*
/* Function: This program is an example of a encryption/decryption */
/*           routine which uses the IBM Common Cryptographic */
/*           Architecture Cryptographic Application Programming */
/*           Interface as defined in reference SC40-1675. */
/*
/*           This program is invoked by DataInterchange (DI) */
/*           during the enveloping or de-enveloping process */
/*
/* Dependencies: none */
/*
/* Restrictions: none */
/*
/* Language:    C */
/*

```



```

/* Attributes:  Reentrant, AMODE(31) RMODE(ANY)          */
/*              INCLUDE OBJ(CCAEE)                      */
/*              INCLUDE OBJ(FXXZCITF)                   */
/*              INCLUDE OBJ(FXXZC)                     */
/*              ENTRY FXXZCITF                          */
/*              NAME CCAEE(R)                           */
/*                                                      */
/* Notes:        Since this program is written in C, the entry */
/*              point must be FXXZCITF, which is provided by the */
/*              DI product. FXXZCITF establishes the necessary */
/*              C environment and branches to the main entry point */
/*              of the program.                            */
/*                                                      */
/*              Parameters passed from DI may be above the line so */
/*              this program has to be 31 bit addressable.      */
/*                                                      */
/* Parameters:  IN - Service Name Block   (snb)          */
/*              Common Block              (ccb)          */
/*              Function Block             (fcb)          */
/*              Encryption Handle          (fh)           */
/*              Encryption key value (ek)                */
/*              Security Data Block (spdb)               */
/*              Buffer size                 */
/*              Buffer containing input data              */
/*              Buffer containing output data             */
/*              Length of data in input buffer           */
/*              Number of characters remaining that would */
/*              not fit into the input buffer            */
/*              Initialization vector return address     */
/*                                                      */
/*              IN OUT - None                          */
/*                                                      */
/*              OUT - CAS Common Block                  */
/*                                                      */
/*              RC      ERC      Meaning               */
/*              0       0       Success                */
/*              8       21      Invalid function code   */
/*              8       22      CFB method not supported */
/*              8       23      Error returning data to DI */
/*              8       24      Error getting data from DI */
/*              8       26      Invalid input data      */
/*              8       11      Keyname not known       */
/*              *       *       *                      */
/*              * As defined by processor implementing the */
/*              CCA                                         */
/*              ----- */
/*
#include <stdefs.h>
#include "disnb.h"          /* SNB definition          */
#include "diccb.h"          /* CCB definition          */
#include "difcb.h"          /* FCB definition          */
#include "dispdb.h"         /* Security data block definition */
#pragma linkage (CSNBRNG,OS) /* Random Number Generate  */
#pragma linkage (CSNBENC,OS) /* Encipher routine        */
#pragma linkage (CSNBDEC,OS) /* Decipher routine        */

```

```

/* ----- */
/* Constant definitions used by this program */
/* ----- */
#define EXISTS    0          /* Existence check */
#define ENCRYPT    1          /* Encrypt data */
#define DECRYPT    2          /* Decrypt data */
#define GETIV     3          /* Get initialization vector */
#define CBC_TYPE  '1'        /* CBC ciphering type */
#define IV_SIZE   8          /* Initialization vector size */
#define KEY_SIZE  16         /* Input key size */
#define NULLPTR   (void *) 0 /* NULL pointer */
#define True      1
#define False     0

/* ----- */
/* Static data used by this program */
/* ----- */
static fcb getfcb = {4,1};    /* Used to get more data */
static fcb putfcb = {4,2};    /* Used to put data */
static char *RANDOM = "RANDOM "; /* Used for random number */
static char *CBC_METH = "CBC "; /* CBC ciphering method */
static char *ICV_INIT = "INITIAL "; /* Starting ICV value */

/* ----- */
/* Local function prototypes */
/* ----- */
void pad_buff(char *buff, long data_len, long *crypt_len);

/* ----- */
/* Main entry point begins here. */
/* ----- */
main(snbptr, ccbptr, fcbptr, fh, ek, spdbptr,
     bufsize, bufin, bufout, datalen, rbc, iv)

    snb    *snbptr;    /* Service name block pointer set up by DI */
    ccb    *ccbptr;    /* Common block pointer used by DI */
    fcb    *fcbptr;    /* Function block pointer which indicates
                        /* what to do:
                        /* 0=EXISTS, 1=ENCRYPT, 2=DECRYPT, 3=GETIV
    long    *fh;        /* Encryption handle - used with GET and PUT
    char    *ek;        /* Encryption key name
    spdb    *spdbptr;   /* Security profile member
    long    *bufsize;   /* The size of the input/output buffers
    char    *bufin;     /* Buffer that contains the source data
    char    *bufout;    /* Buffer for the output data
    long    *datalen;   /* Length of data in the input buffer
    long    *rbc;       /* The number of bytes remaining of the source*
                        /* that would not fit into the input buffer
    char    *iv;        /* Initialization vector return area

{
    long    rule_count;    /* Rule array count
    char    rule_array[8]; /* Rule array
    char    form[8];       /* Form of random number
    char    min_in[16];    /* Minimum size buffer to work with
    char    min_out[16];   /* Minimum output buffer to work out

```

```

char    *local_in;           /* Local input buffer pointer      */
char    *local_out;          /* Local output buffer pointer     */
long    local_bs;            /* Size of local buffer            */
long    local_dl;            /* Length of data in local buffer  */
char    key_token[64];        /* CCA DATA key token             */
char    siv[IV_SIZE];        /* Save original IV value          */
long    filler;              /* Filler variable                 */
long    bytes_left;          /* Number bytes remaining after input
                               /* buffer adjusted for encryption */

long    crypt_in;            /* Encryption/decryption length   */
long    crypt_len;           /* Encryption/decryption length   */
char    chain_vect[18];      /* Chaining vector                 */
int     padded = False;      /* Loop exit variable              */

/* ----- */
/* Set up some initial values. */
/* ----- */
rule_count = 1;
memset(min_in, ' ', sizeof(min_in));
memset(min_out, ' ', sizeof(min_out));
memset(rule_array, ' ', sizeof(rule_array));
memcpy(rule_array, CBC_METH, sizeof(rule_array));
memset(key_token, ' ', sizeof(key_token));

/* ----- */
/* Do some special processing if this is an encrypt or decrypt */
/* request. */
/* ----- */
if ((fcbptr->zfcfunc == ENCRYPT)
    (fcbptr->zfcfunc == DECRYPT)) {
    /* ----- */
    /* Check for the correct ciphering method. CCA only defines the */
    /* CBC method, so reject a request for the Cipher Feedback method */
    /* ----- */
    if (spdbptr->encrtype != CBC_TYPE) {
        ccbptr->zccbrc = 8;           /* Error ... */
        ccbptr->zccberc = 22;        /* Invalid cipher method */
        return;
    }
    /* ----- */
    /* The key passed to this routine by DI (ek) is a 16-byte key */
    /* name that is taken from either the DI mapping records for */
    /* encryption or the S1S or S2s security segments for */
    /* decryption. The CCA Encipher and Decipher calls require that */
    /* a CCA DATA key be provided as a key token value. It is */
    /* your responsibility to write code to handle the storage */
    /* of internal key tokens in a key storage data set so that an */
    /* association can be made between the name of a key (key label) */
    /* and the value of that key (CCA DATA key value). You */
    /* must also provide a key management routine that can be called */
    /* from this routine. The function of this key management routine */
    /* would be to accept a 16-byte key label and type as input and */
    /* return as output a 64-byte key identifier. For more */
    /* information on key labels, key types, and key identifiers, see */
    /* "IBM Common Cryptographic Architecture Cryptographic */
    /* Application Programming Interface Reference (SC40-1675)" */

```

```

/*
/* If your security subsystem is the Integrated Cryptographic
/* Feature (ICRF) with the Integrated Cryptographic Service
/* Facility / MVS (ICSF/MVS), you should replace "key_transform"
/* by a routine that calls whatever program your installation
/* uses to manage DATA keys and MAC keys. For more information
/* on ICSF/MVS, see the following ICSF/MVS publications:
/* "General Information (GC23-0093)"
/* "Administrator's Guide (SC23-0097)"
/* "Application Programmer's Guide (SC23-0098)"
/* "System Programmer's Guide (SC23-0096)"
/* -----
key_transform(ek,"DATA      ",key_token);
}

/* -----
/* Determine what you are asked to do and process accordingly.
/* -----
switch(fcbptr->zfcfunc)
{
    case EXISTS:
        /* -----
        /* Existence check returns with success.
        /* -----
        break;
    case GETIV:
        /* -----
        /* Call the Random Number generate verb to get a random
        /* number. Return codes from the CSNBRNG routine will be
        /* returned to your caller.
        /* -----
        memcpy(form, RANDOM, sizeof(form));
        CSNBRNG(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, form, iv);
        break;
    case ENCRYPT:
        /* -----
        /* Tuck away the original IV value for later encryption.
        /* -----
        memcpy(siv,iv,IV_SIZE);

        /* -----
        /* The size of the buffers passed by DataInterchange is
        /* specified in the Security Profile. This program needs a
        /* buffer size of at least 16 bytes. If you specified
        /* a size less than 16, then use a buffer internal to this
        /* routine rather than the buffers passed by DI.
        /* -----
        if (*bufsize < 16) {
            memcpy(min_in, bufin, *bufsize);
            local_in = min_in;
            local_out = min_out;
            local_bs = sizeof(min_in);
        } else {
            local_in = bufin;
            local_out = bufout;
            local_bs = *bufsize;
        }
}

```

```

local_dl = *datalen;

/* ----- */
/* Now that your working buffer size is right, */
/* you may have gotten data in that is less than */
/* 8 bytes. First, attempt to fill the buffer */
/* up to 8 bytes. If that doesn't work, pad it. */
/* ----- */
if (local_dl < 8) {
    if (*rbc) { /* Any bytes not passed? */
        filler = 8 - local_dl;
        fxxzc(fh,ccbptr,&getfcb, &local_in[local_dl],&filler);
        if (ccbptr->zccbrc) {
            ccbptr->zccbrc = 8; /* Error ... */
            ccbptr->zccberc = 24; /* Error getting data */
            return;
        }
        if ((filler+local_dl) < 8) {
            pad_buff(local_in,filler+local_dl,&local_bs);
            padded = True;
        }
        local_dl = 8; /* Either padded or filled */
    } else {
        pad_buff(local_in, local_dl, &local_bs);
        local_dl = 8;
        padded = True;
    } /* end if(rbc) */
} /* end if(local_dl) */

/* ----- */
/* Adjust the buffer length to a multiple of 8 */
/* and set the number of bytes that will not be */
/* processed the first pass. */
/* ----- */
bytes_left = local_dl % 8;
crypt_len = local_dl - (bytes_left);
if (crypt_len > 8) {
    crypt_len -= 8;
    bytes_left += 8;
}
crypt_in = crypt_len;

/* ----- */
/* Start looping until all data is processed. */
/* You know you are finished when some type of */
/* padding has been done. */
/* ----- */
for(;;) {
    /* ----- */
    /* Call the encipher verb and check for errors. */
    /* ----- */
    filler = 0;
    crypt_len = crypt_in;
    CSNBENC(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, key_token, &crypt_len, local_in, iv,
            &rule_count, rule_array, &filler, chain_vect,
            local_out);

```

```

if (ccbptr->zccbrc)
    return;

/* ----- */
/* Use the DI provided interface to return the encrypted */
/* data to DI. */
/* ----- */
fxxzc(fh, ccbptr, &putfcb, local_out, &crypt_len);
if (ccbptr->zccbrc) {
    ccbptr->zccbrc = 8;          /* Error ... */
    ccbptr->zccberc = 23;       /* Error putting data */
    return;
}

/* ----- */
/* If padding was done, then you are finished. */
/* Else, continue with next set of data. */
/* ----- */
if (padded)
    break;

/* ----- */
/* Ensure that any bytes not processed in the */
/* first pass are accounted for. */
/* ----- */
if (bytes_left)
    memcpy(local_in, local_in+crypt_in, bytes_left);

/* ----- */
/* Check for any data that was not passed to */
/* you in the original input buffer. */
/* ----- */
if (bytes_left >= crypt_in) {
    bytes_left -= crypt_in;
} else {
    if (*rbc) {
        filler = crypt_in - bytes_left;
        fxxzc(fh, ccbptr, &getfcb,
            &local_in[bytes_left], &filler);
        if (ccbptr->zccbrc) {
            ccbptr->zccbrc = 8; /* Error ... */
            ccbptr->zccberc = 24; /* Error getting data */
            return;
        }
    } else
        filler = 0;
    /* ----- */
    /* Check to see if any padding needs to be done. */
    /* ----- */
    if ((filler+bytes_left) < crypt_in) {
        pad_buff(local_in, filler+bytes_left, &crypt_in);
        padded = True;
    }

    bytes_left = 0;          /* Set back to zero */
}

```

```

/* ----- */
/* Continuation is accomplished by copying the first 8 */
/* bytes of the chaining vector into the initialization */
/* vector for the next call. */
/* ----- */
memcpy(iv,chain_vect,IV_SIZE);
} /* end for(;;) */

/* ----- */
/* Encrypt the IV using ECB. The CBC method with an IV of */
/* 0 and 8 bytes of data is the same. */
/* ----- */
crypt_len = IV_SIZE;
memset(iv,0x00,IV_SIZE);
CSNBENC(&(ccbpntr->zccbrn), &(ccbpntr->zccbrn), NULLPTR,
        NULLPTR, key_token, &crypt_len, siv, iv,
        &rule_count, rule_array, &filler, chain_vect, min_out);

/* ----- */
/* Check for error encrypting the IV. */
/* ----- */
if (ccbpntr->zccbrn)
    return;

/* ----- */
/* Return the encrypted IV to the caller. */
/* ----- */
memcpy(iv,min_out,IV_SIZE);
break;
case DECRYPT:
/* ----- */
/* Encrypted data MUST be a multiple of 8. */
/* ----- */
if (((*rbc + *datalen)%8 != 0) ((*rbc + *datalen) <= 0)) {
    ccbpntr->zccbrn = 8; /* Error ... */
    ccbpntr->zccbrn = 26; /* Input data error */
    return;
}

/* ----- */
/* Call the decipher verb to decipher the IV. You are using */
/* the CBC method with an IV of 0 and data length of 8 which */
/* is the same as the ECB method. */
/* ----- */
crypt_len = IV_SIZE;
memcpy(min_in, iv, IV_SIZE);
memset(form, 0x00, sizeof(form));
CSNBDEC(&(ccbpntr->zccbrn), &(ccbpntr->zccbrn), NULLPTR,
        NULLPTR, key_token, &crypt_len, min_in, form,
        &rule_count, rule_array, chain_vect, iv);

/* ----- */
/* Check for errors deciphering the IV. */
/* ----- */
if (ccbpntr->zccbrn)
    return;

```

```

/* ----- */
/* The size of the buffers passed by DataInterchange is      */
/* specified in the Security Profile.  This program needs a   */
/* buffer size of at least 8 bytes.  If you specified a size  */
/* less than 8, then use a buffer internal to this           */
/* routine rather than the buffers passed by DI.             */
/* ----- */
local_dl = *datalen;
if (*bufsize < 8) {
    memcpy(min_in, bufin, *bufsize);
    local_in = min_in;
    local_out = min_out;
    local_bs = sizeof(min_in);
} else {
    local_in = bufin;
    local_out = bufout;
    local_bs = *bufsize;
} /* end if */

/* ----- */
/* Now that your working buffer size is right, you may have  */
/* received data that is less than 8 bytes.  Encrypted data  */
/* MUST be a multiple of 8 bytes.  If you did not receive    */
/* 8, there must be more out there to get.                   */
/* ----- */
if (local_dl < 8) {
    filler = 8 - local_dl;          /* Get only up to 8      */
    fxxzc(fh,ccbptr,&getfcb, &local_in[local_dl],&filler);
    if (ccbptr->zccbrc) {
        ccbptr->zccbrc = 8;          /* Error ...          */
        ccbptr->zccberc = 24;        /* Error getting data */
        return;
    }
    local_dl = 8;
}

/* ----- */
/* Adjust the buffer length to a multiple of 8               */
/* and set the number of bytes that will not be              */
/* processed the first pass.  If you were passed             */
/* a buffer length that was not a multiple of                */
/* 8, you will process only the highest multiple             */
/* you can get on the first call to decipher,                */
/* then any bytes left with the residual not                 */
/* passed to us.                                             */
/* ----- */
bytes_left = local_dl % 8;
crypt_len = local_dl - bytes_left;

/* ----- */
/* Start looping until all data is processed.                */
/* You break out of this loop when you determine             */
/* that all characters have been processed and                */
/* subtracted the number of pad characters.                   */
/* ----- */
for(;;) {
    /* ----- */

```



```

/* Call the decipher verb and check for errors.          */
/* ----- */
crypt_in = crypt_len;
CSNBDEC(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
        NULLPTR, key_token, &crypt_len, local_in, iv,
        &rule_count, rule_array, chain_vect, local_out);

if (ccbptr->zccbrc)
    return;

/* ----- */
/* Put the data out. If no data is left to pro-        */
/* cess, the last byte will contain the number         */
/* of pad characters in ASCII. When this is            */
/* encountered, you exit the loop.                     */
/* ----- */
if (*rbc) {
    fxxzc(fh, ccbptr, &putfcb, local_out, &crypt_len);
    if (ccbptr->zccbrc) {
        ccbptr->zccbrc = 8;      /* Error ...          */
        ccbptr->zccberc = 23;    /* Error putting data */
        return;
    } /* end if */
} else {
    /* ----- */
    /* The last position must be the number             */
    /* of pad characters in ASCII. If this              */
    /* is not true, exit in error.                      */
    /* ----- */
    filler = local_out[crypt_len-1] & 0x0F;
    if ((filler < 1) (filler > 8)) {
        ccbptr->zccbrc = 8;      /* Error ...          */
        ccbptr->zccberc = 26;    /* Input data error   */
        return;
    } /* end if */

    /* ----- */
    /* Flush output minus pad characters.               */
    /* ----- */
    crypt_len = crypt_len - filler;
    fxxzc(fh, ccbptr, &putfcb, local_out, &crypt_len);
    if (ccbptr->zccbrc) {
        ccbptr->zccbrc = 8;      /* Error ...          */
        ccbptr->zccberc = 23;    /* Error putting data */
        return;
    } /* end if */
    break; /* Exit the loop */
} /* end if */

/* ----- */
/* Ensure that any bytes not processed in the         */
/* first pass are accounted for. This only             */
/* needs to be done for the first pass.               */
/* ----- */
if (bytes_left)
    memcpy(local_in, local_in+crypt_in, bytes_left);

```

```

/* ----- */
/* Get any data that was not passed to you in */
/* the original input buffer. You wouldn't have */
/* gotten this far if there wasn't anything */
/* there. */
/* ----- */
filler = crypt_in - bytes_left;
fxxzc(fh,ccbptr,&getfcb, &local_in[bytes_left],&filler);
if (ccbptr->zccbrc) {
    ccbptr->zccbrc = 8;          /* Error ... */
    ccbptr->zccberc = 24;        /* Error getting data */
    return;
}
crypt_len = filler + bytes_left;
bytes_left = 0;                /* Set back to zero */

/* ----- */
/* Change the initialization vector for the next call. */
/* ----- */
memcpy(iv,chain_vect,IV_SIZE);
} /* end for(;;) */
break;
default:
/* ----- */
/* Wrong function code. Set error and return. */
/* ----- */
ccbptr->zccbrc = 8;              /* Error ... */
ccbptr->zccberc = 21;            /* Invalid function code */
break;
} /* end switch */
return;
} /* end main */

/* ----- */
/* Routine: pad_buff */
/* ----- */
/* Purpose: Pad a buffer with ASCII 0s and put the number of places */
/* padded in the last position. */
/* ----- */
/* In      : buff      - The buffer to pad */
/*          : data_len  - The length of data in the buffer */
/*          : crypt_len - Current maximum buff size */
/* ----- */
/* Out     : Modified buffer (buff) and data length (crypt_len) */
/* ----- */
/* ----- */
void pad_buff(buff, data_len, crypt_len)
    char *buff;          /* Input buffer to pad */
    long data_len;        /* Current data length in buffer */
    long *crypt_len;      /* Current cipher length of buffer*/

{
    long num_pad = 0;     /* Number of characters padded */

    /* ----- */
    /* If the buffer length is on a 8-byte boundary (Including */

```

```

/* zero), force the first pad. 'crypt_len' is always on an      */
/* 8-byte boundary and 'data_len' is always less than it, so    */
/* you can never overflow your buffer.                          */
/* ----- */
if ((data_len%8) == 0) {
    buff[data_len++] = 0x30;      /* Force the first pad      */
    num_pad = 1;
} /* end if */

/* ----- */
/* Pad with ASCII 0s.                                           */
/* ----- */
for (;data_len%8;num_pad++, ++data_len)
    buff[data_len] = 0x30;

/* ----- */
/* Put number of pad characters in ASCII in last position      */
/* and change the crypt length to new buffer size.            */
/* ----- */
buff[data_len-1] = (num_pad & 0x0F) 0x30;
*crypt_len = data_len;
} /* end pad_buff() */

```

Get Envelope Service Example

The following CICS COBOL program is an example of an IEXIT outbound user exit that uses the Get Envelope Service to “get” an envelope directly from the translator and route it to a transient data queue. In this fashion, the temporary storage queue restriction in DataInterchange for CICS can be circumvented.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. EDIGETE.

```

```

*****
*           OutBound Envelope CICS User Exit           *
*                                                         *
* On PERFORM ENVELOPE type commands specify,          *
*   FILEID(filename) IEXIT(EDIGETE) IACCESS(M) ITYPE(UE) *
*                                                         *
* This program gets an envelope from a virtual array the *
* translator builds (80 bytes at a time) via the Get    *
* Envelope Service and writes the data to the file      *
* specified in the FILEID keyword (in this case, the    *
* file is known to be a TD queue). The name of this    *
* user exit must exist in the ADAMCTL profile. When     *
* this program is link/edited, include OBJ(FXXZASM).    *
* Object deck FXXZASM (unlike object deck FXXZASMC or   *
* FXXZCBLC) keeps the CICS link-level at its current   *
* position, rather than drops the link-level down as   *
* it invokes the Service Director with the Get Envelope *
* Service call.                                         *
*****

```

```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```

```

01  WS-VARS.
    05 WS-BUFF          PIC X(80).
    05 WS-FLEN          PIC S9(09) COMP-4.

LINKAGE SECTION.

01  SNB                  PIC X(32).
01  CCB.
    05 FILLER            PIC X(12).
    05 CCB-RC            PIC S9(09) COMP-4.
    05 CCB-ERC            PIC S9(09) COMP-4.
    05 FILLER            PIC X(588).
01  FCB                  PIC X(04).
01  GPCB                 PIC X(52).
01  UCB                  PIC X(248).
01  TRCB.
    05 FILLER            PIC X(866).
    05 FILEID            PIC X(08).
    05 FILLER            PIC X(662).
01  TPDB                 PIC X(1532).

PROCEDURE DIVISION USING SNB CCB FCB GPCB UCB TRCB TPDB.

1000-PROGRAM-START.

```

```

    MOVE ZEROES TO CCB-RC.
    PERFORM 2000-PROCESS-DATA UNTIL CCB-RC NOT = ZERO.
    IF (CCB-RC = 8 AND CCB-ERC = 3)
        MOVE ZEROES TO CCB-RC CCB-ERC.
    EXEC CICS
        RETURN
    END-EXEC.
    GOBACK.

```

```

2000-PROCESS-DATA.

    MOVE SPACES TO WS-BUFF.
    MOVE 80 TO WS-FLEN.
    CALL 'FXXZASM' USING GPCB CCB FCB WS-BUFF WS-FLEN.
    IF (CCB-RC = ZERO)
        EXEC CICS
            WRITEQ TD QUEUE(FILEID) FROM(WS-BUFF) LENGTH(80)
        END-EXEC.

```

Put Envelope Service Example

The following CICS COBOL program is an example of an IEXIT inbound user exit that uses the Put Envelope Service to directly provide envelope data to the translator. Here, data is read from a transient data queue and passed directly to the translator for de-enveloping. In this fashion, the temporary storage queue restriction in DataInterchange for CICS can be circumvented.

IDENTIFICATION DIVISION.
PROGRAM-ID. EDIPUTE.

```
*****
*           Inbound Envelope CICS User Exit           *
*                                                     *
* On PERFORM DEENVELOPE type commands specify,      *
* FILEID(filename) IEXIT(EDIPUTE) IACCESS(M) ITYPE(UE) *
*                                                     *
* This program reads the envelope file specified in the *
* FILEID keyword (here the file is known to be a TD   *
* queue) and writes the data to a virtual array using *
* the Put Envelope Service. The translator then      *
* processes the virtual array. The name of this      *
* user exit must exist in the ADAMCTL profile. When  *
* this program is link/edited, include OBJ(FXXZASM). *
* Object deck FXXZASM (unlike object deck FXXZASMC or *
* FXXZCBLC) keeps the CICS link-level at its current *
* position, rather than drops the link-level down as *
* it invokes the Service Director with the Put Envelope *
* Service call.                                     *
*****
```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

```
01 WS-VARS.
   05 WS-PTR          POINTER.
   05 WS-RESP         PIC S9(09) COMP-4.
   05 WS-FLEN         PIC S9(09) COMP-4.
   05 WS-HLEN         PIC S9(04) COMP-4.
```

LINKAGE SECTION.

```
01 SNB                PIC X(32).
01 CCB.
   05 FILLER          PIC X(12).
   05 CCB-RC          PIC S9(09) COMP-4.
   05 CCB-ERC         PIC S9(09) COMP-4.
   05 FILLER          PIC X(588).
01 FCB                PIC X(04).
01 GPCB               PIC X(52).
01 UCB                PIC X(248).
01 TRCB.
   05 FILLER          PIC X(866).
   05 FILEID          PIC X(08).
   05 FILLER          PIC X(662).
01 TPDB               PIC X(1532).

01 BUFFER             PIC X(01).
```

PROCEDURE DIVISION USING SNB CCB FCB GPCB UCB TRCB TPDB.

1000-PROGRAM-START.

MOVE ZEROES TO WS-RESP CCB-RC.

```

PERFORM 2000-PROCESS-DATA UNTIL
    WS-RESP NOT = ZERO OR CCB-RC NOT = ZERO.
IF (WS-RESP NOT = ZERO AND WS-RESP NOT = 23)
    MOVE 8 TO CCB-RC
    MOVE WS-RESP TO CCB-ERC.
EXEC CICS
    RETURN
END-EXEC.
GOBACK.

```

2000-PROCESS-DATA.

```

MOVE ZERO TO WS-HLEN.
EXEC CICS
    READQ TD QUEUE(FILEID)
        SET(WS-PTR) LENGTH(WS-HLEN) NOHANDLE
END-EXEC.
MOVE EIBRESP TO WS-RESP.
IF (WS-RESP = ZERO)
    SET ADDRESS OF BUFFER TO WS-PTR
    MOVE WS-HLEN TO WS-FLEN
    CALL 'FXXZASM' USING GPCB CCB FCB BUFFER WS-FLEN.

```

Inbound Envelope Program Example

IDENTIFICATION DIVISION.
PROGRAM-ID.EDIENV.

```

*****
*           Inbound Envelope CICS Program           *
*                                                                 *
* On PERFORM DEENVELOPE type commands specify,          *
* IEXIT(EDIENV) ITYPE(PG)                                *
*                                                                 *
* This COBOL II program gets invoked directly via an      *
* EXEC CICS LINK command after the envelope has been     *
* deenveloped. The DFHCOMMAREA contains pointers to     *
* the following blocks: Utility Control Block,          *
* Translator Control Block, Trading Partner Data Block, *
* and the Common Control Block.                          *
*                                                                 *
*****

```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

LINKAGE SECTION.

```

01 DFHCOMMAREA.
   05 UCB      POINTER.
   05 TRCBP    POINTER.
   05 TPDBP    POINTER.
   05 CCBP     POINTER.

   01 UCB      PIC X(0248).

```

```

01 TRCB      PIC X(1536).
01 TPDB      PIC X(1532).
01 CCB       PIC X(0608).

```

PROCEDURE DIVISION.

1000-PROGRAM-START.

```

SET ADDRESS OF UCB TO UCBP.
SET ADDRESS OF TRCB TO TRCBP.
SET ADDRESS OF TPDB TO TPDBP.
SET ADDRESS OF CCB TO CCBP.

```

```

EXEC CICS
  WRITEQ TS QUEUE('EDIENV') FROM(UCB) LENGTH(0248)
END-EXEC.
EXEC CICS
  WRITEQ TS QUEUE('EDIENV') FROM(TRCB) LENGTH(1536)
END-EXEC.
EXEC CICS
  WRITEQ TS QUEUE('EDIENV') FROM(CCB) LENGTH(0608)
END-EXEC.

```

```

EXEC CICS
  RETURN
END-EXEC.

```

GOBACK.

Outbound Envelope Program Example

IDENTIFICATION DIVISION.
PROGRAM-ID.EDIOENV.

```

*****
*           Outbound Envelope CICS Program           *
*                                                     *
* On PERFORM DEENVELOPE type commands specify,      *
*   IEXIT(EDIENV) ITYPE(PG)                         *
*                                                     *
* This COBOL II program gets invoked directly via an *
* EXEC CICS LINK command after the envelope has been *
* created. The DFHCOMMAREA contains pointers to the  *
* following blocks: Utility Control Block, Translator *
* Control Block, Trading Partner Data Block, and the  *
* Common Control Block.                             *
*                                                     *
*****

```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

LINKAGE SECTION.

01 DFHCOMMAREA.

```

05 UCB      POINTER.
05 TRCB     POINTER.
05 TPDBP    POINTER.
05 CCB      POINTER.

01 UCB      PIC X(0248).
01 TRCB     PIC X(1536).
01 TPDB     PIC X(1532).
01 CCB      PIC X(0608).

```

PROCEDURE DIVISION.

1000-PROGRAM-START.

```

SET ADDRESS OF UCB TO UCBP.
SET ADDRESS OF TRCB TO TRCBP.
SET ADDRESS OF TPDB TO TPDBP.
SET ADDRESS OF CCB TO CCBP.

```

```

EXEC CICS
  WRITEQ TS QUEUE('EDIOENV') FROM(UCB) LENGTH(0248)
END-EXEC.
EXEC CICS
  WRITEQ TS QUEUE('EDIOENV') FROM(TRCB) LENGTH(1536)
END-EXEC.
EXEC CICS
  WRITEQ TS QUEUE('EDIOENV') FROM(CCB) LENGTH(0608)
END-EXEC.

```

```

EXEC CICS
  RETURN
END-EXEC.

```

GOBACK.

VANICICS Network Program Example

IDENTIFICATION DIVISION.
PROGRAM-ID.EDINETP.

```

*-----*
* THIS CICS COBOL II PROGRAM IS A VANICICS NETWORK PROGRAM *
*-----*

```

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

LINKAGE SECTION.

```

01 DFHCOMMAREA.
05 SNBP      POINTER.
05 CCBP      POINTER.
05 FCBP      POINTER.
05 CMBP      POINTER.
05 TDBP      POINTER.

```



```

05 NDBP    POINTER.
05 RDBP    POINTER.

01    RC      REDEFINES DFHCOMMAREA PIC X(0028)

01 SNB      PIC X(0032).
01 CCB      PIC X(0608).
01 FCB      PIC X(0004).
01 CMB      PIC X(0254).
01 TDB      PIC X(1532)
01 NDB      PIC X(0300)
01 RDB      PIC X(0264)

```

PROCEDURE DIVISION.

1000-PROGRAM-START.

```

    SET ADDRESS OF SNB TO SNBP.
    SET ADDRESS OF CCB TO CCBP.
    SET ADDRESS OF FCB TO FCBP.
    SET ADDRESS OF CMB TO CMBP.
    SET ADDRESS OF TDB TO TDBP.
    SET ADDRESS OF NDB TO NDBP.
    SET ADDRESS OF RDB TO RDBP.

```

```

EXEC CICS
    WRITEQ TS QUEUE('EDINETP') FROM(SNB) LENGTH(0032)
END-EXEC.
EXEC CICS
    WRITEQ TS QUEUE('EDINETP') FROM(CCB) LENGTH(0608)
END-EXEC.
EXEC CICS
    WRITEQ TS QUEUE('EDINETP') FROM(FCB) LENGTH(0004)
END-EXEC.
EXEC CICS
    WRITEQ TS QUEUE('EDINETP') FROM(CMB) LENGTH(0254)
END-EXEC.
EXEC CICS
    WRITEQ TS QUEUE('EDINETP') FROM(TDB) LENGTH(1532)
END-EXEC.
EXEC CICS
    WRITEQ TS QUEUE('EDINETP') FROM(NDB) LENGTH(0300)
END-EXEC.
EXEC CICS
    WRITEQ TS QUEUE('EDINETP') FROM(RDB) LENGTH(0264)
END-EXEC.

```

```

MOVE 'HI00000' TO RC.

```

```

EXEC CICS
    RETURN
END-EXEC.

```

```

GOBACK.

```

Appendix E. Using Sample JCL

DataInterchange Utility (EDIUTILV) JCL

This section describes the DataInterchange Utility JCL. Sample Utility JCLs are distributed in the JCL distribution library (EDI.V3R1M0.SEDIINS1). The VSAM version is distributed as member EDIUTILV and the DB2 version is distributed as member EDIUTILD. This section contains the VSAM version with all the datasets required for all the DataInterchange Utility functions. The JCL will be divided into sub-sections based primarily on function. Although this JCL is VSAM oriented, DB2 implications will be noted as necessary.

Section 1

The beginning of the JCL states that certain elements of the JCL must be modified to run in your environment. This is usual and customary. The minimum region size to execute the DataInterchange Utility is 4096K.

```
//EDIUTILV JOB (INSTALLATION DEPENDENCIES, REGION=4096K)
//*
//*****
//* This sample JCL will invoke the DataInterchange Utility.      *
//*****
//* 1. Change the JOB statement as necessary.                    *
//* 2. Revise ddnames and dataset names to meet your requirements. *
//* 3. If necessary, revise STEPLIB to match your libraries.      *
//* 4. If a language other than English is installed, change all  *
//*    "EDIENU." to "EDI###.", where "###" represents the proper  *
//*    language identifier value shown in the program directory.  *
//*****
```

Section 2

When invoking the DataInterchange Utility, certain parameters are passed in. These parameters are keyword-oriented. The keywords should be separated by at least one blank character. Valid keywords are:

Keyword	Description
APPLID=value	The application ID. EDIFFS is the default value. This keyword also identifies the log file as specified in the ACTLOGS profile.
LANGID=value	The language ID. ENU is the default value. The value supplied must match an entry in the LANGPROF profile. The language ID is used to establish values such as date formats and decimal notations.
SYSID=value	The installation-defined DataInterchange system ID. DIENU is the default value. This keyword is provided to help control access to various components of DataInterchange. The SYSID value is part of the resource name defined using RACF or some other resource control product.
DLM=value	The delimiter used in place of left and right parentheses to enclose DataInterchange Utility command values.
PLAN=value	The DB2 plan name. Providing this parameter is required in DB2 installations, if there is no EDITSIN dataset counterpart. If the PLAN is specified here and in EDITSIN, the value in EDITSIN overrides. There is no default.

| SYSTEM=value The DB2 subsystem ID. Providing this parameter is required in DB2 installations, if
| there is no EDITSIN dataset counterpart. If the SYSTEM is specified here and in
| EDITSIN, the value in EDITSIN overrides. There is no default.

| In VSAM, these parameters are specified with the PARM keyword in the EXEC PGM=EDIFFUT statement.

| In DB2, the parameters can be specified in multiple places. If the EXEC statement specifies
| PGM=IKJEFT01, the parameters are passed in with the PARM keyword in the RUN statement in the
| SYSTSIN dataset. If the EXEC statement specifies PGM=EDIFFUT, the parameters are passed in with
| the PARM keyword in the EXEC statement (however, the DB2 plan and subsystem ID may come from the
| EDITSIN dataset). See "DataInterchange for MVS and DB2 Attachment" on page 3-2 for more
| information.

| In DB2, change the parm "PLAN=EDIENU31" to match your DB2 plan name and change the parm
| "SYSTEM=DSN" to match your DB2 subsystem ID.

```
| /******  
| /* Change the PARMS as follows: *  
| /* 1. If necessary, change the "SYSID=DIENU" parm to match *  
| /* your DI/MVS RACF installation. *  
| /* 2. If necessary, change the "APPLID=EDIFFS" parm to match *  
| /* the desired DI/MVS application ID. *  
| /* 3. If necessary, change the "LANGID=ENU" parm to the *  
| /* proper language identifier. *  
| /******  
| /*  
| //XDIUTIL EXEC PGM=EDIFFUT,PARM='SYSID=DIENU APPLID=EDIFFS LANGID=ENU'  
| /*
```

| Section 3

| STEPLIB accesses the DataInterchange load modules. If you make communication requests, STEPLIB
| must include the library where the communication routines reside. STEPLIB is not required if you have
| provided access to the necessary programs in other ways (such as, adding the load libraries to the
| LNKLST or loading all necessary programs into the link pack area).

| In DB2, STEPLIB must also include the DB2 load library.

```
| //STEPLIB DD DSN=EDI.V3R1M0.SEDILMD1,DISP=SHR    <--EDI LOAD LIBRARY  
| //        DD DSN=SYS1.SNA.LOADLIB,DISP=SHR       <--COMM. LOAD LIBRARY
```

| Section 4

| This section describes the DataInterchange VSAM work files. By and large, all of these files are required
| by all DataInterchange Utility functions. Exceptions to this rule are DOCMAP and LOGEDI. DOCMAP is
| used when processing business documents and LOGEDI is used by the DataInterchange facility.
| EDIMSG contains the text for all DataInterchange messages. EDIPROF and EDITPXRF are affiliated with
| the trading partner profile. LOGFFS is the default DataInterchange Utility event log file. PROFDAT and
| PROFDEF contain all the DataInterchange profiles and their members. TABLDAT and TABLDEF contain
| all the validation and translation tables and their entries. If you are using your own APPLID, you must
| supply a DD statement for the name of the log file associated with your APPLID, and both LOGEDI and
| LOGFFS then become unnecessary.

| In DB2, EDIMSG, EDIPROF, EDITPXRF, LOGEDI, LOGFFS, PROFDAT, PROFDEF, TABLDAT, and
| TABLDEF are not required and should be omitted. Also, log files associated with user-defined APPLIDs,
| are not required.

```

| /*******
| /** Specify EDI required files *
| /** The physical file names are installation dependent. *
| /*******
| /**
| //DOCMAP DD DSN=EDIENU.V3R1M0.DOCMAP,DISP=SHR
| //EDIMSG DD DSN=EDIENU.V3R1M0.MSGTEXT,DISP=SHR
| //EDIPROF DD DSN=EDIENU.V3R1M0.EDIVPROF,DISP=SHR
| //EDITPXRF DD DSN=EDIENU.V3R1M0.TPPXREF,DISP=SHR
| //LOGEDI DD DSN=EDIENU.V3R1M0.LOGEDI,DISP=SHR
| //LOGFFS DD DSN=EDIENU.V3R1M0.LOGFFS,DISP=SHR
| //PROFDAT DD DSN=EDIENU.V3R1M0.PROFDAT,DISP=SHR
| //PROFDEF DD DSN=EDIENU.V3R1M0.PROFDEF,DISP=SHR
| //TABLDAT DD DSN=EDIENU.V3R1M0.TABLDAT,DISP=SHR
| //TABLDEF DD DSN=EDIENU.V3R1M0.TABLDEF,DISP=SHR
| //SYSTSPRT DD SYSOUT=*
| //SYSPRINT DD SYSOUT=*
| /**

```

Section 5

This section identifies all the repository files used by the VSAM version of DataInterchange.

In DB2, none of these DD statements are necessary.

```

| /*******
| /** Allocate VSAM files for repository *
| /*******
| /**

```

Section 5a

EDIVBDxx files are used by the Interactive Entry Facility (IEF). EDIBDxxn files are alternate indexes for files EDIVBDxx.

```

| //EDIVBDDE DD DSN=EDIENU.V3R1M0.EDIVBDDE,DISP=SHR
| //EDIBDDE1 DD DSN=EDIENU.V3R1M0.EDIBDDE1,DISP=SHR
| //EDIBDDE2 DD DSN=EDIENU.V3R1M0.EDIBDDE2,DISP=SHR
| //EDIVBDDF DD DSN=EDIENU.V3R1M0.EDIVBDDF,DISP=SHR
| //EDIVBDDT DD DSN=EDIENU.V3R1M0.EDIVBDDT,DISP=SHR
| //EDIVBDFR DD DSN=EDIENU.V3R1M0.EDIVBDFR,DISP=SHR
| //EDIBDFR1 DD DSN=EDIENU.V3R1M0.EDIBDFR1,DISP=SHR
| //EDIVBDMS DD DSN=EDIENU.V3R1M0.EDIVBDMS,DISP=SHR
| //EDIBDMS1 DD DSN=EDIENU.V3R1M0.EDIBDMS1,DISP=SHR

```

Section 5b

EDIVCSTX stores the control strings used to translate data between application formats and standard formats.

```

| //EDIVCSTX DD DSN=EDIENU.V3R1M0.EDIVCSTX,DISP=SHR

```

Section 5c

EDIVMRxx files are used to maintain trading partner management reporting statistics.

```
| //EDIVMRCM DD DSN=EDIENU.V3R1M0.EDIVMRCM,DISP=SHR
| //EDIVMRPC DD DSN=EDIENU.V3R1M0.EDIVMRPC,DISP=SHR
| //EDIVMRPR DD DSN=EDIENU.V3R1M0.EDIVMRPR,DISP=SHR
| //EDIVMRPS DD DSN=EDIENU.V3R1M0.EDIVMRPS,DISP=SHR
| //EDIVMRRT DD DSN=EDIENU.V3R1M0.EDIVMRRT,DISP=SHR
| //EDIVMRST DD DSN=EDIENU.V3R1M0.EDIVMRST,DISP=SHR
```

| Section 5d

| EDIVSCxx files store standards and envelope definitions. EDISCxxn files are alternate indexes for files EDIVSCxx.

```
| //EDIVSCDE DD DSN=EDIENU.V3R1M0.EDIVSCDE,DISP=SHR
| //EDIVSCDU DD DSN=EDIENU.V3R1M0.EDIVSCDU,DISP=SHR
| //EDISCDU1 DD DSN=EDIENU.V3R1M0.EDISCDU1,DISP=SHR
| //EDIVSCSG DD DSN=EDIENU.V3R1M0.EDIVSCSG,DISP=SHR
| //EDIVSCST DD DSN=EDIENU.V3R1M0.EDIVSCST,DISP=SHR
| //EDIVSCSU DD DSN=EDIENU.V3R1M0.EDIVSCSU,DISP=SHR
| //EDISCSU1 DD DSN=EDIENU.V3R1M0.EDISCSU1,DISP=SHR
| //EDIVSCTX DD DSN=EDIENU.V3R1M0.EDIVSCTX,DISP=SHR
```

| Section 5e

| EDIVTDxx files store the application data format definitions and profile support files.

```
| //EDIVTDID DD DSN=EDIENU.V3R1M0.EDIVTDID,DISP=SHR
| //EDIVTDST DD DSN=EDIENU.V3R1M0.EDIVTDST,DISP=SHR
| //EDIVTPCM DD DSN=EDIENU.V3R1M0.EDIVTPCM,DISP=SHR
| //EDIVTPCN DD DSN=EDIENU.V3R1M0.EDIVTPCN,DISP=SHR
| //EDIVTPCT DD DSN=EDIENU.V3R1M0.EDIVTPCT,DISP=SHR
| //EDITPCN1 DD DSN=EDIENU.V3R1M0.EDITPCN1,DISP=SHR
```

| Section 5f

| EDIVTPxx files are used to maintain mapping and map usage information. EDITPxxn files are alternate indexes for files EDIVTPxx.

```
| //EDIVTPDD DD DSN=EDIENU.V3R1M0.EDIVTPDD,DISP=SHR
| //EDITPDD1 DD DSN=EDIENU.V3R1M0.EDITPDD1,DISP=SHR
| //EDIVTPRT DD DSN=EDIENU.V3R1M0.EDIVTPRT,DISP=SHR
| //EDITPRT1 DD DSN=EDIENU.V3R1M0.EDITPRT1,DISP=SHR
| //EDITPRT2 DD DSN=EDIENU.V3R1M0.EDITPRT2,DISP=SHR
| //EDIVTPRU DD DSN=EDIENU.V3R1M0.EDIVTPRU,DISP=SHR
| //EDIVTPSG DD DSN=EDIENU.V3R1M0.EDIVTPSG,DISP=SHR
| //EDITPSG1 DD DSN=EDIENU.V3R1M0.EDITPSG1,DISP=SHR
| //EDITPSG2 DD DSN=EDIENU.V3R1M0.EDITPSG2,DISP=SHR
| //EDIVTPST DD DSN=EDIENU.V3R1M0.EDIVTPST,DISP=SHR
| //EDITPST1 DD DSN=EDIENU.V3R1M0.EDITPST1,DISP=SHR
| //EDITPST2 DD DSN=EDIENU.V3R1M0.EDITPST2,DISP=SHR
| //EDITPST3 DD DSN=EDIENU.V3R1M0.EDITPST3,DISP=SHR
| //EDIVPTPX DD DSN=EDIENU.V3R1M0.EDIVPTPX,DISP=SHR
| //EDITPTX1 DD DSN=EDIENU.V3R1M0.EDITPTX1,DISP=SHR
| //EDITPTX2 DD DSN=EDIENU.V3R1M0.EDITPTX2,DISP=SHR
| //EDITPTX3 DD DSN=EDIENU.V3R1M0.EDITPTX3,DISP=SHR
```

Section 5g

EDIVTSxx files are used to maintain the Transaction Store. EDITSxxn files are alternate indexes for files EDIVTSxx.

```
//EDIVTSAU DD DSN=EDIENU.V3R1M0.EDIVTSAU,DISP=SHR
//EDITSAU1 DD DSN=EDIENU.V3R1M0.EDITSAU1,DISP=SHR
//EDITSAU2 DD DSN=EDIENU.V3R1M0.EDITSAU2,DISP=SHR
//EDIVTSEV DD DSN=EDIENU.V3R1M0.EDIVTSEV,DISP=SHR
//EDITSEV1 DD DSN=EDIENU.V3R1M0.EDITSEV1,DISP=SHR
//EDITSEV2 DD DSN=EDIENU.V3R1M0.EDITSEV2,DISP=SHR
//EDIVTSGP DD DSN=EDIENU.V3R1M0.EDIVTSGP,DISP=SHR
//EDIVTSTH DD DSN=EDIENU.V3R1M0.EDIVTSTH,DISP=SHR
//EDITSTH1 DD DSN=EDIENU.V3R1M0.EDITSTH1,DISP=SHR
//EDITSTH2 DD DSN=EDIENU.V3R1M0.EDITSTH2,DISP=SHR
//EDIVTSTI DD DSN=EDIENU.V3R1M0.EDIVTSTI,DISP=SHR
//EDIVTSTO DD DSN=EDIENU.V3R1M0.EDIVTSTO,DISP=SHR
//EDIVTSTU DD DSN=EDIENU.V3R1M0.EDIVTSTU,DISP=SHR
//EDITSTU1 DD DSN=EDIENU.V3R1M0.EDITSTU1,DISP=SHR
```

Section 5h

EDIVSSTK is used to store SAP status tracking information. EDISSTK1 is an alternate index for file EDIVSSTK.

```
//EDIVSSTK DD DSN=EDIENU.V3R1M0.EDIVSSTK,DISP=SHR
//EDISSTK1 DD DSN=EDIENU.V3R1M0.EDISSTK1,DISP=SHR
//*
```

Section 6

PRTFILE is required for all DataInterchange Utility functions. Error messages generated during function processing and a summary report will be written to PRTFILE. This dataset contains fixed block addressing (FBA) or variable block addressing (VBA) records with a logical record length of 132.

```
/******
/* Specify the Audit print file. In this case the summary report      *
/* generated by the utility will be returned to JES. You may want      *
/* to allocate an external file, and print it for future reference.    *
/******
/*
//PRTFILE DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=132)
/*
```

Section 7

The APDATA01 and APDATA02 ddnames are examples of files containing application data that is to be translated into standard format. The files can contain C and D records, or they can contain data in raw data format. The actual ddnames must match the names specified with the APPFILE keyword in the PERFORM command. A file is expected to contain C and D records unless the RAWFMTID keyword is provided. These datasets can contain fixed or variable length records.

```

|  /*******
|  /*** If translating to standard formats:
|  /***
|  /*** Allocate all input datasets that will be translated in this step.*
|  /*** These ddnames should be referenced by the APPFILE keyword in
|  /*** the EDISYSIN input command language statements.
|  /*******
|  /***
|  /**APDATA01 DD DSN=PHYSICAL.FILE1.NAME,DISP=OLD
|  /**APDATA02 DD DSN=PHYSICAL.FILE2.NAME,DISP=OLD
|  /***

```

Section 8

When translating to standard format, the DataInterchange Utility writes transactions that were not translated successfully to FFSEXCP. When translating to application format, the Utility writes translated transactions to FFSEXCP if it could not write them to the intended file (this is usually because the intended file could not be opened or is full). Optional records (IEGTQ) are also written to FFSEXCP if the tracking file (FFSTRAK) does not exist and your application data is in C and D format. FFSEXCP must be large enough to contain the largest data record you are translating, and the largest information record the translator might return. Use the JCL DISP option to control whether FFSEXCP is cleared or appended to during the first use. After the first use, DataInterchange automatically appends to the file. FFSEXCP can contain fixed or variable length records.

```

|  /*******
|  /*** If translating to standard or to application formats:
|  /***
|  /*** Specify the exception file to hold transactions in error
|  /*** DCB= Allocated the same as application data files, since
|  /*** this file holds a copy of the untranslated transaction.
|  /*******
|  /***
|  /**FFSEXCP DD DSN=BAD.TRANSACTION.FILE,DISP=MOD
|  /***

```

Section 9

FFSTRAK holds the optional information records (IEGTQ) if they are requested. If you are using C and D records and if FFSTRAK is not provided, the optional records are written to the exception file (FFSEXCP). If your application data is in raw data format and FFSTRAK is not provided, no optional records are written even if they are requested. FFSTRAK must be large enough to accommodate the largest information record. FFSTRAK can contain fixed or variable length records.

```

|  /*******
|  /*** If translating to standard formats:
|  /***
|  /*** Specify the tracking file to hold IEGTQ type records if
|  /*** they are to be separated from the exception file. This
|  /*** is recommended if your application input data is in a raw data
|  /*** format. Allocate the same as FFSEXCP.
|  /*******
|  /***
|  /**FFSTRAK DD DSN=HOLD.TRACING.FILE,DISP=MOD
|  /***

```


Section 10

FFSWORK is required only when translating from application format to standard format. FFSWORK holds the current transaction in case of translation errors. If an error occurs, the current transaction is copied from the FFSWORK file to the exception file (FFSEXCP). This dataset can contain variable record format with a logical record length of 32756. If your system guidelines allow, specifying UNIT=VIO on this DD statement will drastically reduce I/O EXCPs on the dataset. Allocation using a record format of variable blocked (VB) will also reduce the number of I/O EXCPs on the dataset.

```
/******  
/* If translating to standard or to application formats:      *  
/**                                                           *  
/* Specify the work file (temporary hold file during translation) *  
/******  
/**                                                           *  
//FFSWORK DD DSN=&&FFSWORK,DISP=(NEW,DELETE),UNIT=SYSALLDA,  
//          DCB=(RECFM=V,BLKSIZE=32760),SPACE=(TRK,(1,1))  
/**
```

Section 11

If your Utility request involves enveloping and/or sending of transaction data, this is the section of JCL that defines the standard data file to envelope into and/or send from. The actual ddname should match one of the following:

- The "Trans data queue" field value as specified in the network profile (NETPROF). If a name has not been supplied in the NETPROF member, the default is QDATA. The name specified holds transactions that have either ISA, ICS, or GS enveloping. The name specified with an E appended (for example, QDATAE) holds transactions that have either UNB or STX enveloping. The name specified with a U appended (for example, QDATAU) holds transactions that have BG enveloping. There must be a ddname here for each network that can be processed during the run.
- The value specified with the FILEID keyword in the PERFORM command. If a FILEID override is provided, that file is used to hold all envelope types for all networks.

This dataset can contain fixed-length or variable-length records with a logical record length of 80 or greater.

```
/******  
/* If this job step includes enveloping and/or sending data:  *  
/**                                                           *  
/* Specify the network transaction files to hold queued transactions *  
/* The ddnames MUST match the transaction data queue field    *  
/* in the network profile unless you will be using the FILEID  *  
/* keyword override in the EDISYSIN input command language    *  
/* statements.                                                 *  
/**                                                           *  
/* Use DISP=MOD to append data to the file                    *  
/*     DISP=OLD to replace the file                            *  
/**                                                           *  
/* Suggested: DCB=(RECFM=V,LRECL=80,BLKSIZE=23440). However, this *  
/*     may not satisfy YOUR network requirements, so it      *  
/*     is suggested that you verify this allocation.          *  
/******  
/**                                                           *  
//QDATA DD DSN=NETWORK.HOLDFILE.NAME,DISP=MOD  
/**
```

Section 12

This group of JCL statements is used when sending to or receiving from a network. The JCL shown here is required for the IBM Global Network. Each network has its own JCL requirements.

Section 12a

This section contains JCL statements for Expedite/MVS network program TPMAIN. The DataInterchange supplied network profile is IINR3. INFILE contains commands written by DataInterchange for processing by Expedite/MVS. The ddname of INFILE must match the value supplied in the "Network input file" field of the network profile member. The logical record length of INFILE must match the value supplied in the "Input rec length" field of the network profile member. OUTFILE contains responses from Expedite/MVS after its processing. The ddname of OUTFILE must match the value supplied in the "Net output file" field of the network profile member. OUTFILE contains fixed-length records with a logical record length of 80. ISCMSGS and ISCTRACE are ddnames required by Expedite/MVS for its own processing and are not used by DataInterchange.

```
/******  
/* If this job step includes sending/receiving data or the      *  
/* UPDATE STATUS command:                                     *  
/*                                                             *  
/* Specify the IGN Expedite/MVS network program (TPMAIN)      *  
/* required work files. The network profile name is IINR3.     *  
/* INFILE is used to hold the network commands                *  
/* OUTFILE contains the completion status of the commands      *  
/* Suggested: DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)            *  
/******  
/*  
//INFILE DD DSN=&&INFILE,DISP=(NEW,DELETE),UNIT=SYSALLDA,  
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(1,1))  
//OUTFILE DD DSN=NETWORK.OUTFILE,DISP=OLD  
//ISCMSGS DD SYSOUT=*  
//ISCTRACE DD UNIT=SYSALLDA,SPACE=(CYL,(1,5)),DISP=(,PASS),  
//          DCB=(LRECL=6000,BLKSIZE=12288,BUFNO=2,RECFM=VB)  
/*
```

Section 12b

This section contains JCL statements for Expedite Base/MVS network program IEBASE. The DataInterchange supplied network profiles are IINB1 (IEBASE Release 1.1), IINB41 (IEBASE Release 4.1), and IINB42 (IEBASE Release 4.2 and higher). The files described here are required by Expedite Base/MVS. INMSG is the network input file and contains commands written by DataInterchange for processing by Expedite Base/MVS. OUTMSG is the network output file and contains responses from Expedite Base/MVS after its processing. INB1STAT is used during UPDATE STATUS to hold network acknowledgments. For more information about these Expedite Base/MVS files, see the *Expedite Base/MVS Programming Guide*.

```

| /*******
| /** Specify the IGN Expedite Base/MVS network program (IEBASE)      *
| /** required work files. The network profile name is IINB42.         *
| /**                                                                    *
| /** DSNNAME      DESCRIPTION                      RECFM      LRECL      *
| /**                                                                    *
| /** ERRORMSG     ERROR DESCRIPTIONS FILE          FB          80        *
| /** ERRORTXT     EXTENDED ERROR DESCRIPTIONS FILE FB          80        *
| /** INB1STAT     STATUS INFORMATION FILE           V            255        *
| /** INMSG        MESSAGE COMMAND FILE             FB          80        *
| /** INPRO        PROFILE COMMAND FILE             FB          80        *
| /** OUTMSG       MESSAGE RESPONSE FILE            FB          80        *
| /** OUTPRO       PROFILE RESPONSE FILE            FB          80        *
| /**                                                                    *
| /** NOTE:  INPRO in sample JCL needs to be customized for your      *
| /**        system.                                                  *
| /**                                                                    *
| /*******
| /**
| /**ERRORMSG DD DSN=SYS1.EXPBASE,EXPMSGs(ERRORMSG),DISP=SHR
| /**ERRORTXT DD DSN=SYS1.EXPBASE,EXPMSGs(ERRORTXT),DISP=SHR
| /**INB1STAT DD DSN=NETWORK.INB1STAT,DISP=OLD
| /**INMSG    DD DSN=&&INMSG,DISP=(NEW,DELETE),UNIT=SYSALLDA,
| /**          DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(1,1))
| /**INPRO    DD DSN=EDI.V1R3M0.SEDISAM1(EDIINPRO),DISP=SHR
| /**OUTMSG   DD DSN=NETWORK.OUTMSG,DISP=OLD
| /**OUTPRO   DD DSN=NETWORK.OUTPRO,DISP=OLD

```

If destination tables are needed to resolve Information Exchange (IE) mailboxes, then a qualifier table (QUTTABLE) and one or more destination tables (TTABLExx) may be necessary. The ddname of the destination table(s) may be specified in the data of the qualifier table and can be any valid ddname. If the qualifier table (QUTTABLE) is not used, then the following default ddnames are used for the destination table(s):

Envelope Type	Ddname
X12	TTABLExx - where xx is the two-character qualifier (ISA07 element of the interchange envelope)
UCS	TTABLE01
UN/TDI	TTABLE
EDIFACT	TTABLExx - where xx is the first two characters of the qualifier (UNTO:2 element of the interchange envelope)

```

| //QUTTABLE DD DSN=USERFILE.IN.ANYNAME.SEQ.FILE,DISP=SHR
| //TTABLExx DD DSN=USERFILE.IN.ANYNAME.SEQ.FILExx,DISP=SHR
| /**

```

Section 12c

This section contains JCL statements for Expedite Base/MVS network program IEBase with Comm-Press. In addition to the files mentioned in Section 12b, these files become required if you are using Expedite Base/MVS with compression. For more information about these Expedite Base/MVS files, see the *Expedite Base/MVS Programming Guide*.

```

| /*******
| /** If you are using Expedite Base/MVS with Comm-Press, then specify *
| /** the following work files: *
| /** *
| /** DSNNAME      DESCRIPTION      RECFM      LRECL      *
| /** *
| /** COMPPDS      COMPRESSION PDS FILE      FB      80      *
| /** COMPWRK      COMPRESSION WORK FILE      FB      80      *
| /** INMSGC      MESSAGE COMMAND FILE      FB      80      *
| /** INMSGR      MESSAGE RESPONSE FILE      FB      80      *
| /** OUTMSGC      MESSAGE COMMAND FILE      FB      80      *
| /** OUTMSGR      MESSAGE RESPONSE FILE      FB      80      *
| /** SYSUT1      TEMPORARY WORK FILE FOR COMPRESSION FB      80      *
| /** *
| /** COMPTRC      COMPRESSION TRACE FILE *
| /** CPLOOKUP      COMPRESSION LOOKUP TABLE      FB      80      *
| /** *
| /** NOTE:  COMPTRC is required when BASE(Y) is used on the trace *
| /**        command. CPLOOKUP is required when using COMPRESS(T). *
| /** *
| /*******
| /**
| /**COMPPDS  DD DSN=NETWORK.COMPPDS,DISP=(NEW,CATLG),
| /**          DCB=(RECFM=FB,LRECL=80,BLKSIZE=0,DSORG=PO),
| /**          UNIT=SYSALLDA,SPACE=(CYL,(10,2,10))
| /**COMPWRK  DD DSN=NETWORK.COMPWRK,DISP=(NEW,CATLG),
| /**          DCB=(RECFM=V,LRECL=84,BLKSIZE=23440,DSORG=PS),
| /**          UNIT=SYSALLDA,SPACE=(TRK,(10,1))
| /**INMSGC   DD DSN=&&INMSGC,DISP=(NEW,DELETE),UNIT=SYSALLDA,
| /**          DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(1,1))
| /**INMSGR   DD DSN=&&INMSGR,DISP=(NEW,DELETE),UNIT=SYSALLDA,
| /**          DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(1,1))
| /**OUTMSGC  DD SYSOUT=*
| /**OUTMSGR  DD DSN=NETWORK.OUTMSGR,DISP=(NEW,CATLG),
| /**          DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
| /**          UNIT=SYSALLDA,SPACE=(TRK,(10,1))
| /**SYSUT1   DD DSN=&&SYSUT1,DISP=(NEW,DELETE,DELETE),
| /**          UNIT=SYSDA,DCB=BLKSIZE=23476,SPACE=(CYL,(1,1))
| /**COMPTRC  DD SYSOUT=*
| /**CPLOOKUP DD DSN=NETWORK.CPLOOKUP,DISP=(NEW,CATLG),
| /**          DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
| /**          UNIT=SYSALLDA,SPACE=(TRK,(10,1))
| /**

```

Section 13

If your Utility request involves receiving and/or deenveloping of transaction data, this is the section of JCL that defines the standard data file to receive into and/or deenvelope from. The actual ddname should match one of the following:

- The "Receive file name" field value as specified in the requestor profile (REQPROF) associated with the REQID keyword in the PERFORM command. There must be a ddname here for each requestor that can be processed during the run.
- The value specified with the FILEID keyword in the PERFORM command. The FILEID value will override any "Receive file name" values from requestor profiles.

| This dataset can contain fixed-length or variable-length records with a logical record length of 80 or greater.

```
| /******  
| /* If this job step includes receiving and/or deenveloping of      *  
| /* transaction data:                                           *  
| /*                                                                *  
| /* Specify the Network Transaction files to receive into and/or  *  
| /* deenvelope from.                                           *  
| /*                                                                *  
| /* The ddname MUST match the receive file name in the Requestor *  
| /* profile or the FILEID override keyword in the EDISYSIN input *  
| /* command language statements.                               *  
| /* A ddname is required for each unique ddname specified by a  *  
| /* Requestor ID (receive ddname obtained from Requestor profile). *  
| /* Use DISP=MOD to append data to the file                     *  
| /* DISP=OLD to replace the file                                *  
| /******  
| /*  
| //REQ1DD DD DSN=REQ1.TRANS,DISP=MOD  
| //REQ2DD DD DSN=REQ2.TRANS,DISP=MOD  
| /*
```

| Section 14

| INVOICE identifies a file that holds data in application format when translating from standard format. The actual ddname is specified in the "Application file name" field in the application data format. A ddname has to specified here for each application data format that can be processed during the run. The name specified in the application data format can be overridden with a value in the "Application file name" field in the receive usage. If there is no applicable DD statement, the application data is written to the exception file (FFSEXCP). This dataset can contain fixed or variable length records. The logical record length must be large enough to hold the largest application record.

```
| /******  
| /* If translating to application formats:                        *  
| /*                                                                *  
| /* Specify the sequential application (output) files           *  
| /*                                                                *  
| /* ddnames MUST be supplied for every Application Data Format  *  
| /* that can be received in this job and they must match the  *  
| /* ddname specified in defining the Application Data Format.   *  
| /* Multiple job steps can be used if multiple network transaction *  
| /* files must be separated into unique application files.     *  
| /* Transactions will either replace the file or are appended to *  
| /* the file based on the DISP parameter.                       *  
| /* Use DISP=MOD to append data to the file                     *  
| /* DISP=OLD to replace the file                                *  
| /* Allocation parameters should handle your largest data record *  
| /* plus any informational records you may request.             *  
| /******  
| /*  
| //INVOICE DD DSN=INVOICE.DATA.FILE,DISP=MOD  
| //PURCORD DD DSN=PURCORD.DATA.FILE,DISP=MOD  
| /*
```

Section 15

RPTFILE identifies a file that contains reports generated by the PERFORM PRINT command and by various other PERFORM commands. This dataset can contain fixed block addressing (FBA) or variable block addressing (VBA) records with a logical record length of 132.

```
/******  
/*  
/*          PLEASE READ!!!!  
/*  
/* The following DD assignments (up to the EDISYSIN DD assignment)  
/* are used for specific commands which most likely will not be  
/* used too often. Please read the comments carefully to determine  
/* if the ddnames should be allocated or removed.  
/******  
/*  
/******  
/* If using any of the PRINT commands:  
/*  
/* Specify the output file to write the reports out to.  
/*  
/* Use DISP=MOD to append to the existing file  
/*     DISP=OLD to replace the file  
/* Allocation parameters should indicate a record length of 133.  
/******  
/*  
//RPTFILE DD DSN=REPORT.FILE,DISP=MOD  
/*
```

Section 16

EDIQUERY identifies a file that contains output from PERFORM QUERY commands and from various other PERFORM commands. This dataset should contain variable length records with a block size of 32760.

```
/******  
/* If using the following commands:  
/*     TRADING PARTNER PROFILE DATA EXTRACT  
/*     TRADING PARTNER CAPABILITY DATA EXTRACT  
/*     TRANSACTION ACTIVITY DATA EXTRACT  
/*     NETWORK ACTIVITY DATA EXTRACT  
/*     TRANSACTION DATA EXTRACT  
/*     ENVELOPE DATA EXTRACT  
/*     QUERY  
/*  
/* Specify the output file to write the extracted data or matching  
/* handles to.  
/*  
/* Use DISP=MOD to append to the existing file  
/*     DISP=OLD to replace the file  
/*  
/* Allocation parameters should indicate a variable record format  
/* with a block size of 32760. This will allow any type of data  
/* to be written to the query file without the possibility of  
/* records being truncated.  
/******  
/*
```

```
| //EDIQUERY DD DSN=QUERY.FILE,DISP=MOD
| /**
```

| Section 17

| This section of JCL is required if you use the MAPPING MIGRATION command. These ddnames should be referenced by the INCNTL and OUTCNTL keywords in PERFORM MAPPING MIGRATION command. You should run the migration once with an OUTCNTL file and no INCNTL file. If changes to the mapping are needed, edit the OUTCNTL file and submit only the changed records as the INCNTL file on the next run of mapping migration. You can use the same file and data definition for INCNTL and OUTCNTL.

```
| /*******
| /** If using the MAPPING MIGRATION command: *
| /** * *
| /** You may want to specify an INCNTL and/or an OUTCNTL file. *
| /** These ddnames should be referenced by the INCNTL and OUTCNTL *
| /** keywords in the EDISYSIN input command language statements. *
| /** * *
| /** Allocation parameters should indicate record length of at least *
| /** 80 bytes. *
| /** * *
| /** You should run the migration once with an OUTCNTL file *
| /** and no INCNTL file. If changes to the mapping are needed, EDIt *
| /** the OUTCNTL file and submit only the changed records as the *
| /** INCNTL file the next run of mapping migration. You can use the *
| /** same file and data definition name for INCNTL and OUTCNTL. *
| /** * *
| /*******
| /**
| /**INCNTL DD DSN=MMCNTL.FILE,DISP=OLD
| /**OUTCNTL DD DSN=MMCNTL.FILE,DISP=OLD
```

| Section 18

| This section of JCL is required if you request an export (PERFORM EXPORT) or import (PERFORM IMPORT).

```
| /*******
| /** If using the EXPORT or IMPORT commands: *
| /** * *
| /** 1) Allocate the export/import DI VSAM control file. *
| /** * *
| /** 2) Allocate the export/import user supplied control file. *
| /** This control file describes what data is to be exported or *
| /** imported. An allocation of fixed format and record length 84 *
| /** should be used. This file can also be allocated inline. *
| /** Make sure to use the same ddname allocated with the *
| /** CTLFILE keyword in the EDISYSIN input command language *
| /** statements. In this sample CTLFILE(EXIMCTL) would be used. *
| /** * *
| /** 3) Allocate the DD statements for the export/import files *
| /** themselves. All the ddnames can point to a single file *
| /** or they can each point to a separate file. This sample *
| /** JCL shows the ddnames pointing to separate files. *
| /** Allocation of the dataset(s) should have a record format *
| /** of variable, a record length of 4089, and a block size of 4093.*
| /** Use DISP=MOD to append to the existing file(s) *
```

```
|  /*          DISP=OLD to replace the file(s)                      *
|  /******
|  /*
```

| Section 18a

| EXIMCTL contains the control statements describing what data should be exported or imported. This ddname must match the CTLFILE keyword value that you specify on the PERFORM command. The export and import control statements are described in “Export/Import Control File (CTLFILE)” on page 2-12. The record format should be fixed with a logical record length of 84.

```
|  /******
|  /* Step 1 - Allocate user supplied control file
|  /******
|  /*
|  //EXIMCTL DD DSN=EXPORT.IMPORT.CTLFILE,DISP=OLD
```

| Section 18b

| The remaining EDIEIxxx DD statements define files to which data is exported, or from which data is imported. It is possible to assign all DD statements to a single physical file, or to separate them as shown in this example. These datasets contain variable length records with a logical record length of 4089.

```
|  /******
|  /* Step 2 - Allocate export/import files
|  /******
|  /*
|  //EDIEIADF DD DSN=EXPORT.IMPORT.ADF,DISP=MOD
|  //EDIEICST DD DSN=EXPORT.IMPORT.CST,DISP=MOD
|  //EDIEIDDF DD DSN=EXPORT.IMPORT.DDF,DISP=MOD
|  //EDIEIMAP DD DSN=EXPORT.IMPORT.MAP,DISP=MOD
|  //EDIEIPRF DD DSN=EXPORT.IMPORT.PRF,DISP=MOD
|  //EDIEISTD DD DSN=EXPORT.IMPORT.STD,DISP=MOD
|  //EDIEITBL DD DSN=EXPORT.IMPORT.TBL,DISP=MOD
|  //EDIEITPT DD DSN=EXPORT.IMPORT.TPT,DISP=MOD
|  /*
```

| Section 19

| FAENV contains override data that the translator uses to generate functional acknowledgments. This is an optional file processed during deenvelope functions. The format of the data in this file can be found in “Enveloping Options File for Functional Acknowledgments (FAENV)” on page 2-9. This dataset can contain fixed or variable length records. The logical record length must be as large as the largest record in the file.

```
|  /******
|  /* If deenveloping data:
|  /*
|  /* Specify the sequential file which contains overrides
|  /* that you want used when functional acknowledgments are
|  /* generated for the data that is being deenveloped.
|  /*
|  /* This file is optional and if not specified the envelope data
|  /* for functional acknowledgments will be taken from the standard
|  /* profile member specified for the transactions.
|  /*
|  /* Suggested: DCB=(RECFM=V,LRECL=255)
```



```

| //*****
| //*
| //FAENV      DD DSN=FA.OVERRIDES.FILE,DISP=SHR
| //*

```

Section 20

EDISYSIN is the file from which the Utility PERFORM commands are read. If EDISYSIN is not defined, the Utility will check SYSIN for the PERFORM commands. For more information on these commands see Chapter 1, "Using the DataInterchange Utility" on page 1-1. This dataset can contain fixed or variable length records with any logical record length.

```

| //*****
| //*          VERY IMPORTANT: EDISYSIN DD ASSIGNMENT          *
| //*          *                                               *
| //* The EDISYSIN device contains all command language statements *
| //* to be processed in this job step. In this sample JCL it    *
| //* has been allocated inline, but EDISYSIN can be allocated to any *
| //* dataset containing command language input. If it is allocated *
| //* to a dataset, any record length and format can be used.    *
| //* Make sure sequence numbers do not appear in cols 73-80; this will *
| //* cause errors.                                             *
| //*          *                                               *
| //* Note: The DataInterchange Utility will first look for PERFORM *
| //*       commands in EDISYSIN. If there is no EDISYSIN defined, *
| //*       the Utility will look in SYSIN.                     *
| //*****
| //*
| //EDISYSIN DD *
| * An asterisk in column one denotes a comment line..
| * ----- *
| * Here is a sample of command language input to translate and
| * envelope transactions from application file APDATA01 and generate
| * optional records.
| * ----- *
| PERFORM TRANSLATE AND ENVELOPE
| WHERE APPFILE(APDATA01) OPTRECS(IEGTQ)
|
| * ----- *
| * Here is a sample of command language input to send queued
| * transactions on behalf of requestor ID ROBOX
| * ----- *
| PERFORM SEND WHERE REQID(ROBOX)
|
| * ----- *
| * Here is a sample of command language input to receive transactions
| * from requestor ID ROBOX
| * ----- *
| PERFORM RECEIVE WHERE REQID(ROBOX)
|
| * ----- *
| * Here is a sample of command language input to deenvelope and
| * translate received transactions associated with requestor ID ROBOX
| * ----- *
| PERFORM DEENVELOPE AND TRANSLATE
| WHERE REQID(ROBOX)
| /*

```

Section 21

This section identifies the Pageable Translation work file, EDIVAX. EDIVAX should be defined as a temporary dataset. Space allocation is dependent on the amount of data that will be paged. Pageable Translation is specified by using the PAGE keyword available on all translate type commands and on all envelope/deenvelope type commands. For information regarding EDIVAX space allocation and Pageable Translation in general, see the PAGE keyword description on page 1-99.

```
/******  
/* Specify the Pageable Translation work file. Uncomment the      *  
/* following DD statement if you use this feature. The space    *  
/* allocation is dependent on the amount of data to be paged.   *  
/******  
/*  
/*EDIVAX DD DISP=(NEW,DELETE,DELETE),UNIT=SYSALLDA,SPACE=(CYL,25)  
/*
```

Section 22

This section is for DB2 only. It is typical in a DB2 environment that batch application programs (such as the DataInterchange Utility) run under the TSO Terminal Monitor Program (TMP) in background mode. This is specified by naming IKJEFT01 in the EXEC JCL statement. For example, the following EXEC JCL statement would replace the one shown in "Section 2" on page E-1.

```
/*XDIUTIL EXEC PGM=IKJEFT01,DYNAMNBR=50
```

The parameters passed to IKJEFT01 are specified in the SYSTSIN dataset. These parameters include the DB2 subsystem ID, the DB2 plan, the name of the application program, and the application program parameter string. IKJEFT01 connects DB2 and opens the plan, runs the application program (EDIFFUT), and then closes the plan and disconnects DB2. When EDIFFUT executes via IKJEFT01 the DB2 processing mode is called DSN. This processing mode assumes that dataset EDITSIN does not exist, and that the DataInterchange Utility parameters are specified in SYSTSIN. See "DataInterchange for MVS and DB2 Attachment" on page 3-2 for more information.

```
/******  
/* Specify the parameters to DB2 to execute the utility          *  
/*                                                                *  
/* *KEY* These are the statements which run the DB2 program.    *  
/*                                                                *  
/* 1. If necessary, change "DSN" to your local DB2 subsystem ID. *  
/* 2. If necessary, change the PLAN (EDIENU31) to match the      *  
/*    the plan name you used when binding the plan during install.*  
/* 3. If necessary, change the parm "LANGID=ENU" to match        *  
/*    the national language you are using.                      *  
/* 4. If necessary, change the parm "SYSID=DIENU" to match       *  
/*    your DI/MVS RACF installation.                             *  
/* 5. If necessary, change the parm "APPLID=EDIFFS" to match     *  
/*    the desired DI/MVS application ID.                        *  
/* 6. If necessary, change the parm "PLAN=EDIENU31" to match     *  
/*    the plan name you used when binding the plan during install.*  
/* 7. If necessary, change the parm "SYSTEM=DSN" to match        *  
/*    your local DB2 subsystem ID.                              *  
/******  
/*  
/*SYSTSIN DD *  
/* DSN SYSTEM (DSN)  
/* RUN PROG (EDIFFUT) -
```

```

PLAN (EDIENU31) -
PARM('LANGID=ENU SYSID=DIENU APPLID=EDIFFS PLAN=EDIENU31 SYSTEM=DSN')
END
/*

```

Utility Data Sets Required

Table E-1 outlines the sections of DD statements that may be necessary based on the requested DataInterchange Utility. The numbers in the table refer to the previous section headings. An asterisk (*) after the number indicates that the DD statement is optional.

Table E-1 (Page 1 of 2). JCL Sections Required by the DataInterchange Utility

DEENVELOPE	1	2	3	4	5	6	8	13	19	20		
ENVELOPE	1	2	3	4	5	6	8	9*	10	11	20	
ENVELOPE AND SEND	1	2	3	4	5	6	8	9*	10	11	12	20
ENVELOPE DATA EXTRACT	1	2	3	4	5	6	16	20				
EXPORT	1	2	3	4	5	6	18	20				
HOLD	1	2	3	4	5	6	20					
IMPORT	1	2	3	4	5	6	18	20				
MIGRATION MAPPING	1	2	3	4	5	6	17	20				
NETWORK ACTIVITY DATA EXTRACT	1	2	3	4	5	6	16	20				
PRINT	1	2	3	4	5	6	15	20				
PRINT CUSTOM LAYOUT	1	2	3	4	5	6	20					
PURGE	1	2	3	4	5	6	20					
QUERY	1	2	3	4	5	6	16	20				
RECEIVE	1	2	3	4	5	6	12	13	20			
RECEIVE AND DEENVELOPE	1	2	3	4	5	6	8	12	13	19*	20	
RECEIVE AND TRANSLATE	1	2	3	4	5	6	8	12	13	14	19*	20
REENVELOPE	1	2	3	4	5	6	8	9*	10	11	20	
RELEASE	1	2	3	4	5	6	20					
REMOVE TRANSACTIONS	1	2	3	4	5	6	20					
RETRANSLATE TO APPLICATION	1	2	3	4	5	6	8	14	20			
SEND	1	2	3	4	5	6	11	12	20			
TRADING PARTNER CAPABILITY DATA EXTRACT	1	2	3	4	5	6	16	20				
TRADING PARTNER PROFILE DATA EXTRACT	1	2	3	4	5	6	16	20				
TRANSACTION ACTIVITY DATA EXTRACT	1	2	3	4	5	6	16	20				
TRANSACTION DATA EXTRACT	1	2	3	4	5	6	16	20				
TRANSLATE TO APPLICATION	1	2	3	4	5	6	8	14	20			
TRANSLATE AND ENVELOPE	1	2	3	4	5	6	7	8	9*	10	11	20
TRANSLATE TO STANDARD	1	2	3	4	5	6	7	8	9*	10	20	

Table E-1 (Page 2 of 2). JCL Sections Required by the DataInterchange Utility

TRANSLATE AND SEND	1	2	3	4	5	6	7	8	9*	10	11	12	20
UNPURGE	1	2	3	4	5	6	20						
UPDATE STATUS	1	2	3	4	5	6	12	20					

Archive VSAM event log entries (EDIELARV)

This section shows the VSAM event log archive/removal JCL as it is distributed in the JCL distribution library (EDI.V3R1M0.SEDIINS1). For additional information, see “Removing and Archiving Event Log Entries” on page 1-33.

```
//EDIELARV JOB (INSTALLATION DEPENDENCIES, REGION=4096K)
//*
//*****
//* THIS SAMPLE JCL WILL ARCHIVE AND PURGE LOG ENTRIES FROM A VSAM *
//* EVENT LOG. THIS JCL MUST BE MODIFIED FOR YOUR LOCAL ENVIRONMENT. *
//*****
//* 1. CHANGE JOB CARD STATEMENT TO LOCAL REQUIREMENTS *
//* 2. CHANGE ARCHIVE FILE (ARCFILE) AND THE HOLD FILE (HLDFILE) DD *
//* STATEMENTS AS NECESSARY. *
//* 3. CHANGE THE DASD VOLUME ID (VVVVVV) FOR YOUR LOCAL SYSTEM *
//* 4. IF NECESSARY, CHANGE THE PARM "DIENU" IN THE STEP "ARCHIVE" *
//* FOR YOUR DI/MVS RACF INSTALLATION. *
//*****
//*
//*****
//* THE ENTRIES SELECTED FOR REMOVAL WILL BE COPIED TO THE ARCHFILE, *
//* AND ALL OTHER ENTRIES WILL BE COPIED TO THE HOLDFILE. *
//*****
//CREATE EXEC PGM=IEFBR14
//ARCFILE DD DSN=EDIENU.V3R1M0.ARCFILE,DISP=(NEW,CATLG),
//          DCB=(RECFM=V,LRECL=4096,BLKSIZE=0),
//          UNIT=SYSALLDA,SPACE=(CYL,(10,5))
//HLDFILE DD DSN=EDIENU.V3R1M0.HLDFILE,DISP=(NEW,CATLG),
//          DCB=(RECFM=V,LRECL=4096,BLKSIZE=0),
//          UNIT=SYSALLDA,SPACE=(CYL,(10,5))
//*
//ARCHIVE EXEC PGM=EDIFFUT,PARM='SYSID=DIENU APPLID=EDIFFS LANGID=ENU'
//STEPLIB DD DSN=EDI.V3R1M0.SEDIIMD1,DISP=SHR  <--EDI LOAD LIBRARY
//*
//*****
//* SPECIFY EDI REQUIRED FILES *
//* THE PHYSICAL FILE NAMES ARE INSTALLATION DEPENDENT. *
//* LOGFFS IS THE FLAT FILE SUPPORT PRIVATE LOG FILE *
//*****
//DOCMAP DD DSN=EDIENU.V3R1M0.DOCMAP,DISP=SHR
//EDIMSG DD DSN=EDIENU.V3R1M0.MSGTEXT,DISP=SHR
//EDIPROF DD DSN=EDIENU.V3R1M0.EDIVPROF,DISP=SHR
//EDITPXRF DD DSN=EDIENU.V3R1M0.TPPXREF,DISP=SHR
//EDITSTU1 DD DSN=EDIENU.V3R1M0.EDITSTU1,DISP=SHR
//EDIVTSEV DD DSN=EDIENU.V3R1M0.EDIVTSEV,DISP=SHR
//EDIVTSGP DD DSN=EDIENU.V3R1M0.EDIVTSGP,DISP=SHR
//EDIVTSTH DD DSN=EDIENU.V3R1M0.EDIVTSTH,DISP=SHR
//LOGFFS DD DSN=EDIENU.V3R1M0.LOGFFS,DISP=SHR
```

```

| //PROFDAT DD DSN=EDIENU.V3R1M0.PROFDAT,DISP=SHR
| //PROFDEF DD DSN=EDIENU.V3R1M0.PROFDEF,DISP=SHR
| //TABLDAT DD DSN=EDIENU.V3R1M0.TABLDAT,DISP=SHR
| //TABLDEF DD DSN=EDIENU.V3R1M0.TABLDEF,DISP=SHR
| //SYSTSPRT DD SYSOUT=*
| //SYSPRINT DD SYSOUT=*
| //ARCFIL DD DSN=EDIENU.V3R1M0.ARCFILE,DISP=SHR
| //HLDFILE DD DSN=EDIENU.V3R1M0.HLDFILE,DISP=SHR
| //PRTFILE DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=132)
| /*
| //SYSIN DD *
| PERFORM UNLOAD LOG ENTRIES WHERE APPLID(EDIFFS)
| LOGFILE(LOGFFS) LOGDATE(09/15/98) TO(09/30/98)
| ARCHIVEFILE(ARCFIL) HOLDFILE(HLDFILE)
|
| /*
| //*****
| /* DEFINE THE EDI LOGFFS VSAM CLUSTER *
| //*****
| //DEFINE EXEC PGM=IDCAMS,COND=(4,LT,ARCHIVE)
| //SYSPRINT DD SYSOUT=*
| //SYSIN DD *
| DELETE (EDIENU.V3R1M0.LOGFFS) -
| CLUSTER -
| PURGE
| IF LASTCC = 8 THEN SET MAXCC = 0
| DEFINE CLUSTER -
| (NAME(EDIENU.V3R1M0.LOGFFS) -
| VOL(VVVVVV) -
| NIXD -
| CYL(1 1) -
| RECSZ(512 32750) -
| SHR(3 3) -
| CISZ(4096) -
| SPANNED) -
| DATA -
| (NAME(EDIENU.V3R1M0.LOGFFS.DATA))
|
| /*
| //RELOAD EXEC PGM=EDIFFUT,PARM='SYSID=DIENU APPLID=EDIFFS LANGID=ENU',
| // COND=(4,LT,ARCHIVE)
| //STEPLIB DD DSN=EDI.V3R1M0.SEDILMD1,DISP=SHR <--EDI LOAD LIBRARY
| /*
| //*****
| /* SPECIFY EDI REQUIRED FILES *
| /* THE PHYSICAL FILE NAMES ARE INSTALLATION DEPENDENT. *
| /* LOGFFS IS THE FLAT FILE SUPPORT PRIVATE LOG FILE *
| //*****
| //DOCMAP DD DSN=EDIENU.V3R1M0.DOCMAP,DISP=SHR
| //EDIMSG DD DSN=EDIENU.V3R1M0.MSGTEXT,DISP=SHR
| //EDIPROF DD DSN=EDIENU.V3R1M0.EDIVPROF,DISP=SHR
| //EDITPXRF DD DSN=EDIENU.V3R1M0.TPPXREF,DISP=SHR
| //LOGFFS DD DSN=EDIENU.V3R1M0.LOGFFS,DISP=SHR
| //PROFDAT DD DSN=EDIENU.V3R1M0.PROFDAT,DISP=SHR
| //PROFDEF DD DSN=EDIENU.V3R1M0.PROFDEF,DISP=SHR
| //TABLDAT DD DSN=EDIENU.V3R1M0.TABLDAT,DISP=SHR
| //TABLDEF DD DSN=EDIENU.V3R1M0.TABLDEF,DISP=SHR
| //SYSTSPRT DD SYSOUT=*
| //SYSPRINT DD SYSOUT=*

```

```

| //PRTFILE DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=132)
| //HLDFILE DD DSN=EDIENU.V3R1M0.HLDFILE,DISP=SHR
| /*
| //SYSIN DD *
| PERFORM LOAD LOG ENTRIES WHERE APPLID(EDIFFS)
| LOGFILE(LOGFFS) HOLDFILE(HLDFILE)
| /*

```

Archive DB2 event log entries (EDIELARD)

This section shows the DB2 event log archive/removal JCL as it is distributed in the JCL distribution library (EDI.V3R1M0.SEDIINS1). For additional information, see "Removing and Archiving Event Log Entries" on page 1-33.

```

| //EDIELARD JOB (INSTALLATION DEPENDENCIES)
| /*
| //*****
| /* THIS SAMPLE JCL WILL ARCHIVE AND PURGE LOG ENTRIES FROM A DB2 *
| /* EVENT LOG. THIS JCL MUST BE MODIFIED FOR YOUR LOCAL ENVIRONMENT. *
| //*****
| /* 1. CHANGE JOB CARD STATEMENT TO LOCAL REQUIREMENTS *
| /* 2. CHANGE ARCHIVE FILE (ARCFE) AND THE HOLD FILE (HLDFE) DD *
| /* STATEMENTS AS NECESSARY. *
| /* 3. IF NECESSARY, CHANGE THE PARM "DIENU" IN THE STEP "ARCHIVE" *
| /* FOR YOUR DI/MVS RACF INSTALLATION. *
| /* 4. IF NECESSARY, CHANGE DB2 "SYSTEM", "PLAN", AND "LIB" *
| /* PARAMETER VALUE IN SYSTSIN DATA STREAMS. *
| //*****
| /*
| //*****
| /* THE ENTRIES SELECTED FOR REMOVAL WILL BE COPIED TO THE ARCHFE, *
| /* AND ALL OTHER ENTRIES WILL BE COPIED TO THE HOLDFE. *
| //*****
| //CREATE EXEC PGM=IEFBR14
| //ARCFE DD DSN=EDIENU.V3R1M0.ARCFE,DISP=(NEW,CATLG),
| // DCB=(RECFM=V,LRECL=4096,BLKSIZE=0),
| // UNIT=SYSALLDA,SPACE=(CYL,(10,5))
| //HLDFE DD DSN=EDIENU.V3R1M0.HLDFE,DISP=(NEW,CATLG),
| // DCB=(RECFM=V,LRECL=4096,BLKSIZE=0),
| // UNIT=SYSALLDA,SPACE=(CYL,(10,5))
| /*
| //ARCHIVE EXEC PGM=IKJEFT01,DYNAMNBR=50
| /*
| //STEPLIB DD DSN=DSN.DSNLOAD,DISP=SHR <--DB2 LOAD LIBRARY
| // DD DSN=EDI.V3R1M0.SEDILMD1,DISP=SHR <--EDI LOAD LIBRARY
| /*
| //*****
| /* SPECIFY EDI REQUIRED FILES *
| /* THE PHYSICAL FILE NAMES ARE INSTALLATION DEPENDENT. *
| /* LOGFFS IS THE FLAT FILE SUPPORT PRIVATE LOG FILE *
| //*****
| //DOCMAP DD DSN=EDIENU.V3R1M0.DOCMAP,DISP=SHR
| //SYTSPT DD SYSOUT=*
| //SYSPRINT DD SYSOUT=*
| //PRTFILE DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=132)
| //ARCFE DD DSN=EDIENU.V3R1M0.ARCFE,DISP=SHR

```

```

| //HLDFILE DD DSN=EDIENU.V3R1M0.HLDFILE,DISP=SHR
| /*
| //SYSIN DD *
| PERFORM UNLOAD LOG ENTRIES WHERE APPLID(EDIFFS)
| LOGDATE(10/01/98) TO(12/31/98)
| ARCHIVEFILE(ARCFIL) HOLDFILE(HLDFILE)
| /*
| //SYSTSIN DD *
| DSN SYSTEM (DSN)
| RUN PROG (EDIFFUT) -
| PLAN (EDIENU31) -
| PARM('LANGID=ENU SYSID=DIENU APPLID=EDIFFS PLAN=EDIENU31 SYSTEM=DSN')
| END
| /*
| //*****
| /* REORG THE DB2 TABLE EDIELOG (EVENT LOG) *
| //*****
| //REORG EXEC PGM=DSNUTILB,PARM='DSN,DSNTEX',COND=(4,LT,ARCHIVE)
| //STEPLIB DD DSN=DSN.DSNLOAD,DISP=SHR <--DB2 LOAD LIBRARY
| //SYSPRINT DD SYSOUT=*
| //UTPRINT DD SYSOUT=*
| //SORTOUT DD UNIT=SYSALLDA,SPACE=(4000,(20,20),,,ROUND)
| //SORTWK01 DD UNIT=SYSALLDA,SPACE=(4000,(20,20),,,ROUND)
| //UNLD DD UNIT=SYSALLDA,SPACE=(4000,(20,20),,,ROUND)
| //WORK DD UNIT=SYSALLDA,SPACE=(4000,(20,20),,,ROUND)
| //SYSUT1 DD UNIT=SYSALLDA,SPACE=(4000,(20,20),,,ROUND)
| //SYSIN DD *
| REORG TABLESPCE (EDIENU31.EDIELOG)
| UNLDDN (UNLD)
| WORKDDN (UNLD)
| REORG INDEX (EDIENU31.EDIELOGX)
| /*

```


Appendix F. Space Calculation Examples

Space Requirements for DataInterchange Tables and Files

Table F-1 describes the formulas for determining the space requirements for each of the DataInterchange tables and files:

Where:

Record Name

Identifies the DB2 table names and VSAM file names of the DI database records that you have installed.

DI Product

Specifies if the record applies to the DI DB2 or VSAM product, or to BOTH.

DB Type

Specifies if the database record is defined as a DB2 table or as a VSAM file.

Description

Specifies the database record and formulas to assist in estimating DASD allocation. Where database records apply to both VSAM and DB2, any references to other database records are given using the VSAM naming convention of EDIVxxxx. For example, the reference name is the same for the DI DB2 Product; except the V is dropped, EDIxxxx.

Table F-1 (Page 1 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
MSGTEXT	VSAM/VSAM File	DataInterchange Messages - Contains the text for all messages issued from DataInterchange. Static file except perhaps during PTF applies when messages may be added or updated.
SCREENS	Both/VSAM File	DataInterchange Screens - Contains the text for all screens displayed by DataInterchange. Static file except perhaps during PTF applies when screens may be added or updated.
HELPS	Both/VSAM File	DataInterchange Help Text - Contains the help text for all online help provided by DataInterchange. Static file except perhaps during PTF applies when help text may be added or updated.
DOCMAP	Both/VSAM File	Document Layout - The document layout utility produces a document layout (DOCMAP) record and a print file. IEF uses the DOCMAP record to set initial values when a document is entered or copied and to format a document for viewing or printing.
EDITPXRF	VSAM/VSAM File	DataInterchange TPPROF cross reference - Contains cross-reference keys to the TPPROF profile members for quick determination of the TP nickname given to either the Interchange Qualifier/ID or the Account/User ID. Each entry in the file is 81 bytes and the size of the file depends on the number of Interchange Qualifier/ID and Account/User ID combinations you have defined in TPPROF.
LOGEDI	VSAM/VSAM File DB2/DB2 Table	Data file for Event Logging for messages during product administration

Table F-1 (Page 2 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description																																				
LOGFFS	VSAM/VSAM File DB2/DB2 Table	Data file for Event Logging for messages during utility function																																				
PROFDEF	VSAM/VSAM File	DataInterchange Profile Definitions - Contains the definitions of the DataInterchange profiles. This is a static file that does not change in size.																																				
PROFDAT	VSAM/VSAM File	<div><div>DataInterchange Profile Data - Contains the profile members that are created during EDI customization. Primary profiles subject to change/expansion within PROFDAT and the trading partner profile (TPPROF), the envelope profiles (EITUX), and the requestor profile (REQPROF). These profile entries are given to help you with your space estimates.</div><table><thead><tr><th>Profile Entry</th><th>Size</th></tr></thead><tbody><tr><td>ACTLOGS</td><td>88 bytes</td></tr><tr><td>ADAMCTL</td><td>92 bytes</td></tr><tr><td>APPDEFS</td><td>92 bytes</td></tr><tr><td>CONTRECV</td><td>220 bytes</td></tr><tr><td>E</td><td>531 bytes</td></tr><tr><td>I</td><td>253 bytes</td></tr><tr><td>LANGPROF</td><td>96 bytes</td></tr><tr><td>MQSERIES</td><td>176 bytes</td></tr><tr><td>NETOP</td><td>168 bytes</td></tr><tr><td>NETPROF</td><td>288 bytes</td></tr><tr><td>REQPROF</td><td>252 bytes</td></tr><tr><td>SECUPROF</td><td>144 bytes</td></tr><tr><td>SYSPROF</td><td>76 bytes</td></tr><tr><td>T</td><td>284 bytes</td></tr><tr><td>TPPROF</td><td>1520 bytes</td></tr><tr><td>U</td><td>268 bytes</td></tr><tr><td>X</td><td>275 bytes</td></tr></tbody></table><div>Note: The sizes shown here are the actual lengths of individual profiles. They are not the lengths of the data blocks used by DataInterchange to access the profiles. When DataInterchange accesses the profiles, a data block with 12 extra bytes on the front is used. For API purposes, the data blocks associated with the profiles, include the 12 extra bytes.</div></div>	Profile Entry	Size	ACTLOGS	88 bytes	ADAMCTL	92 bytes	APPDEFS	92 bytes	CONTRECV	220 bytes	E	531 bytes	I	253 bytes	LANGPROF	96 bytes	MQSERIES	176 bytes	NETOP	168 bytes	NETPROF	288 bytes	REQPROF	252 bytes	SECUPROF	144 bytes	SYSPROF	76 bytes	T	284 bytes	TPPROF	1520 bytes	U	268 bytes	X	275 bytes
Profile Entry	Size																																					
ACTLOGS	88 bytes																																					
ADAMCTL	92 bytes																																					
APPDEFS	92 bytes																																					
CONTRECV	220 bytes																																					
E	531 bytes																																					
I	253 bytes																																					
LANGPROF	96 bytes																																					
MQSERIES	176 bytes																																					
NETOP	168 bytes																																					
NETPROF	288 bytes																																					
REQPROF	252 bytes																																					
SECUPROF	144 bytes																																					
SYSPROF	76 bytes																																					
T	284 bytes																																					
TPPROF	1520 bytes																																					
U	268 bytes																																					
X	275 bytes																																					
TABLDEF	VSAM/VSAM File	Validation and Translation Table Definitions - Contains the definitions of validation and translation tables. Subject to large additions when new standards are applied to your system or when new applications are added.																																				
TABLDAT	VSAM/VSAM File	Validation and Translation Table Data - Contains validation and translation table entries. Subject to quite large additions when new standards are applied to your system or when new applications are added.																																				
EDIVCSTX EDICSTX	VSAM/VSAM File DB2/DB2 Table	<div>Mapping Control String - Control string used to speed execution during translation. A good starting point is to assume 1 of these for each EDIVTDID entry and 4 of these for each EDIVTPTX entry.</div> <div><ul style="list-style-type: none">CSTXNUM = TDIDNUM + (4 * TPTXNUM)</div>																																				

Table F-1 (Page 3 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description																										
EDICSTXX	DB2/DB2 Index	Mapping Control String Unique Index - Unique index created on the key value to EDICSTX. One of these exists for each EDICSTX entry. <ul style="list-style-type: none">• CSTXXNUM = CSTXNUM																										
EDIVSCDE EDISCDE	VSAM/VSAM File DB2/DB2 Table	Standard Data Element Definition - Provides information about a data element within a standard. One entry exists for each data element defined in each standard. These are some of the standards shipped by DataInterchange: <table><tr><th>Standard</th><th>Value</th></tr><tr><td>EDI902</td><td>304</td></tr><tr><td>TDCC28</td><td>863</td></tr><tr><td>UCSV3R1</td><td>323</td></tr><tr><td>X12V2R2</td><td>710</td></tr><tr><td>X12V2R3</td><td>765</td></tr><tr><td>X12V2R4</td><td>823</td></tr><tr><td>X12V3R1</td><td>901</td></tr><tr><td>E</td><td>34</td></tr><tr><td>I</td><td>22</td></tr><tr><td>T</td><td>22</td></tr><tr><td>U</td><td>21</td></tr><tr><td>X</td><td>28</td></tr></table> <ul style="list-style-type: none">• SCDENUM = one of the above values or your own estimate	Standard	Value	EDI902	304	TDCC28	863	UCSV3R1	323	X12V2R2	710	X12V2R3	765	X12V2R4	823	X12V3R1	901	E	34	I	22	T	22	U	21	X	28
Standard	Value																											
EDI902	304																											
TDCC28	863																											
UCSV3R1	323																											
X12V2R2	710																											
X12V2R3	765																											
X12V2R4	823																											
X12V3R1	901																											
E	34																											
I	22																											
T	22																											
U	21																											
X	28																											
EDISCDEX	DB2/DB2 Index	Standard Data Element Definition Unique Index - Unique index created on the key value to EDISCDE. One of these for each EDISCDE entry. <ul style="list-style-type: none">• SCDEXNUM = SCDENUM																										
EDIVSCDU EDISCDU	VSAM/VSAM File DB2/DB2 Table	Standard Data Element Usage - Records usage of a data element within a segment. One of these exists each time an element is used within a segment. These are some of the standards shipped by DataInterchange: <table><tr><th>Standard</th><th>Value</th></tr><tr><td>EDI902</td><td>1012</td></tr><tr><td>TDCC28</td><td>3677</td></tr><tr><td>UCSV3R1</td><td>1083</td></tr><tr><td>X12V2R2</td><td>2830</td></tr><tr><td>X12V2R3</td><td>3001</td></tr><tr><td>X12V2R4</td><td>3382</td></tr><tr><td>X12V3R1</td><td>3707</td></tr><tr><td>E</td><td>46</td></tr><tr><td>I</td><td>26</td></tr><tr><td>T</td><td>22</td></tr><tr><td>U</td><td>25</td></tr><tr><td>X</td><td>32</td></tr></table> <ul style="list-style-type: none">• SCDUNUM = one of the above values or your own estimate	Standard	Value	EDI902	1012	TDCC28	3677	UCSV3R1	1083	X12V2R2	2830	X12V2R3	3001	X12V2R4	3382	X12V3R1	3707	E	46	I	26	T	22	U	25	X	32
Standard	Value																											
EDI902	1012																											
TDCC28	3677																											
UCSV3R1	1083																											
X12V2R2	2830																											
X12V2R3	3001																											
X12V2R4	3382																											
X12V3R1	3707																											
E	46																											
I	26																											
T	22																											
U	25																											
X	32																											
EDISCDEX	DB2/DB2 Index	Standard Data Element Usage Unique Index - Unique index created on the key value to EDISCDU. One of these exists for each EDISCDU entry. <ul style="list-style-type: none">• SCDUXNUM = SCDUNUM																										

Table F-1 (Page 4 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description																										
SCDUAIX1	VSAM/VSAM Index	<p>Standard Data Element Usage Alternate Index - An alternate index created in the VSAM environment only, which keeps track of which segments use a data element. Worst case is there will be as many SCDUAIX1 entries as there are EDIVSCDU entries. This would not be the case if the same field is used multiple times within the same segment.</p> <ul style="list-style-type: none">• SCDUAIX1NUM = SCDUNUM																										
EDIVSCSG EDISCSG	VSAM/VSAM File DB2/DB2 Table	<p>Standard Segment Definition - Provides information about a segment within a standard. One entry exists for each segment defined in each standard. These are some of the standards shipped by DataInterchange:</p> <table><tr><th>Standard</th><th>Value</th></tr><tr><td>EDI902</td><td>60</td></tr><tr><td>TDCC28</td><td>514</td></tr><tr><td>UCSV3R1</td><td>155</td></tr><tr><td>X12V2R2</td><td>400</td></tr><tr><td>X12V2R3</td><td>433</td></tr><tr><td>X12V2R4</td><td>464</td></tr><tr><td>X12V3R1</td><td>520</td></tr><tr><td>E</td><td>6</td></tr><tr><td>I</td><td>6</td></tr><tr><td>T</td><td>6</td></tr><tr><td>U</td><td>6</td></tr><tr><td>X</td><td>6</td></tr></table> <ul style="list-style-type: none">• SCSGNUM = one of the above values or your own estimate	Standard	Value	EDI902	60	TDCC28	514	UCSV3R1	155	X12V2R2	400	X12V2R3	433	X12V2R4	464	X12V3R1	520	E	6	I	6	T	6	U	6	X	6
Standard	Value																											
EDI902	60																											
TDCC28	514																											
UCSV3R1	155																											
X12V2R2	400																											
X12V2R3	433																											
X12V2R4	464																											
X12V3R1	520																											
E	6																											
I	6																											
T	6																											
U	6																											
X	6																											
EDISCSGX	DB2/DB2 Index	<p>Standard Segment Definition Unique Index - Unique index created on the key value to EDISCSG. One of these exists for each EDISCSG entry.</p> <ul style="list-style-type: none">• SCSGXNUM = SCSGNUM																										
EDIVSCST EDISCST	VSAM/VSAM File DB2/DB2 Table	<p>Standard definition - Defines the name, version, release, and default envelope type that should be used for a particular standard. One entry exists for each EDI standard and for each Envelope standard defined on the system.</p> <p>Suspect that at least 2 of the X, E, I, T, or U envelope standards will be installed plus whatever EDI standards are applied and/or copied.</p> <p>Note: Applying or coping a standard adds entries to all EDIVSCxx and EDISCxx tables.</p> <ul style="list-style-type: none">• SCSTNUM = Number of standards installed																										
EDISCSTX	DB2/DB2 Index	<p>Standard Definition Unique Index - Unique index created on the key value to EDISCST. One of these exists for each EDISCST entry.</p> <ul style="list-style-type: none">• SCSTXNUM = SCSTNUM																										

Table F-1 (Page 5 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description																
EDIVSCSU EDISCSU	VSAM/VSAM File DB2/DB2 Table	<p>Standard Segment Usage - Records usage of a segment within a transaction. One of these exists for each time a segment is used within a transaction. These are some of the standards shipped by DataInterchange:</p> <table><tr><th>Standard</th><th>Value</th></tr><tr><td>EDI902</td><td>838</td></tr><tr><td>TDCC28</td><td>2299</td></tr><tr><td>UCSV3R1</td><td>708</td></tr><tr><td>X12V2R2</td><td>923</td></tr><tr><td>X12V2R3</td><td>1161</td></tr><tr><td>X12V2R4</td><td>1287</td></tr><tr><td>X12V3R1</td><td>1622</td></tr></table> <ul style="list-style-type: none">SCSUNUM = one of the above values or your own estimate	Standard	Value	EDI902	838	TDCC28	2299	UCSV3R1	708	X12V2R2	923	X12V2R3	1161	X12V2R4	1287	X12V3R1	1622
Standard	Value																	
EDI902	838																	
TDCC28	2299																	
UCSV3R1	708																	
X12V2R2	923																	
X12V2R3	1161																	
X12V2R4	1287																	
X12V3R1	1622																	
EDISCSUX	DB2/DB2 Index	<p>Standard Segment Usage Unique Index - Unique index created on the key value to EDISCSU. One of these exists for each EDISCSU entry.</p> <ul style="list-style-type: none">SCSUXNUM = SCSUNUM																
SCSUAIX1	VSAM/VSAM Index	<p>Standard Segment Usage Alternate Index - An alternate index created in the VSAM environment only, which keeps track of where a segment is used. Worst case is there will be as many SCSUAIX1 entries as there are EDIVSCSU entries. This would not be the case if the same segment is used multiple times within the same transaction.</p> <ul style="list-style-type: none">SCSUAIX1NUM = SCSUNUM																
EDIVSCTX EDISCTX	VSAM/VSAM File DB2/DB2 Table	<p>Standard Transaction Definition - Provides information about a transaction within a standard. One entry exists for each transaction defined in a standard. These are some of the standards shipped by DataInterchange:</p> <table><tr><th>Standard</th><th>Value</th></tr><tr><td>EDI902</td><td>15</td></tr><tr><td>TDCC28</td><td>127</td></tr><tr><td>UCSV3R1</td><td>32</td></tr><tr><td>X12V2R2</td><td>19</td></tr><tr><td>X12V2R3</td><td>25</td></tr><tr><td>X12V2R4</td><td>28</td></tr><tr><td>X12V3R1</td><td>39</td></tr></table> <ul style="list-style-type: none">SCTXNUM = one of the above values or your own estimate	Standard	Value	EDI902	15	TDCC28	127	UCSV3R1	32	X12V2R2	19	X12V2R3	25	X12V2R4	28	X12V3R1	39
Standard	Value																	
EDI902	15																	
TDCC28	127																	
UCSV3R1	32																	
X12V2R2	19																	
X12V2R3	25																	
X12V2R4	28																	
X12V3R1	39																	
EDISCTXX	DB2/DB2 Index	<p>Standard Transaction Definition Unique Index - Unique index created on the key value to EDISCTX. One of these exists for each EDISCTX entry.</p> <ul style="list-style-type: none">SCTXXNUM = SCTXNUM																

Table F-1 (Page 6 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIVTPDD EDITPDD	VSAM/VSAM File DB2/DB2 Table	<p>Trading Partner Data Element Definition - Copy of information in EDIVSCDE and EDIVSCDU entries made when a data element is mapped. In order to estimate the number of entries you need to know the average number of segments that are mapped and the average number of fields per segment. Multiply these two values together to get the average number of EDIVTPDD entries per EDIVTPTX entry.</p> <ul style="list-style-type: none"> SSMAP = Average number of MAPPED segments per map ELSS = Average number of elements per selected segment (mapped or not mapped) TPDDNUM = TPTXNUM * SSMAP * ELSS
EDITPDDX	DB2/DB2 Index	<p>Trading Partner Data Element Definition Unique Index - Unique index created on the key value to EDITPDD. One of these exists for each EDITPDD entry.</p> <ul style="list-style-type: none"> TPDDXNUM = TPDDNUM
TPDDAIX1	VSAM/VSAM Index	<p>Trading Partner Data Element Definition Alternate Index - An alternate index created in the VSAM environment only which keeps track of which EDIVTPTX entries have mapped a particular data element. Worst case is there will be as many TPDDAIX1 entries as there are EDIVTPDD entries. This would not be the case if the same data element is used multiple times within the same selected segment.</p> <ul style="list-style-type: none"> TPDDAIX1NUM = TPDDNUM
EDIVTPRT EDITPRT	VSAM/VSAM File DB2/DB2 Table	<p>Trading Partner Receive Usage - Records the use of a mapping by a particular trading partner. This is only used for transactions being received and there will be one entry for each trading partner using a particular transaction.</p> <ul style="list-style-type: none"> TPTXNUM = Total number of mappings PRECV = Percentage of mappings (TPTXNUM) that are receive mappings TPPROFNUM = Total number of trading partners TPRECV = Percentage of tradings partners you receive data from TPRTNUM = (TPTXNUM * PRECV) * (TPPROFNUM * TPRECV)
EDITPRTX	DB2/DB2 Index	<p>Trading Partner Receive Usage Unique Index - Unique index created on the key value to EDITPRT. One of these exists for each EDITPRT entry.</p> <ul style="list-style-type: none"> TPRTXNUM = TPRTNUM
TPRTAIX1	VSAM/VSAM Index	<p>Trading Partner Receive Usage Alternate Index 1 - An alternate index created in the VSAM environment only, which is used to make sure there is only one usage per mapping for a given trading partner and application sender/receiver combination. Worst case is there will be as many TPRTAIX1 entries as there are EDIVTPRT entries. This would not be true if the same trading partner and application sender/receiver were used for multiple transaction types.</p> <ul style="list-style-type: none"> TPRTAIX1NUM = TPRTNUM

Table F-1 (Page 7 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
TPRTAIX2	VSAM/VSAM Index	<p>Trading Partner Receive Usage Alternate Index 2 - An alternate index created in the VSAM environment only, which is used by the translator to locate the mapping for a given transaction being received. There will be as many TPRTAIX2 entries as there are EDIVTPRT entries.</p> <ul style="list-style-type: none"> TPRTAIX2NUM = TPRTNUM
EDIVTPRU EDITPRU	VSAM/VSAM File DB2/DB2 Table	<p>Trading Partner Rules - Keeps track of special mapping characteristics for a data element. One of these entries is created unless the mapping is a simple move between application and standard fields (no literal, tables, edits). For space calculations a good assumption is you have one entry for each field mapped so you need to know the average number of fields mapped per transaction.</p> <ul style="list-style-type: none"> TPDDNUM = Total number of fields SPMAP = Percentage of fields that are mapped with either literal values, accumulators, edits, validation/translation tables, sub-string/concatenate, user exits TPRUNUM = TPDDNUM * SPMAP
EDITPRUX	DB2/DB2 Index	<p>Trading Partner Rules Unique Index - Unique index created on the key value to EDITPRU. One of these exists for each EDITPRU entry.</p> <ul style="list-style-type: none"> TPRUXNUM = TPRUNUM
EDIVTPSG EDITPSG	VSAM/VSAM File DB2/DB2 Table	<p>Trading Partner Segment Usage - Copied from EDIVSCSG and EDIVSCSU entries and keeps track of the use of a segment in the mapping. There is one of these for each segment defined in the transaction and one for each segment selected in a repeated mapping.</p> <ul style="list-style-type: none"> SSSTD = Average number of segments in a transaction (mapped and not mapped) SSRMAP = Average number of segments MAPPED in a REPEATED mapping TPSGNUM = TPTXNUM * (SSSTD + SSRMAP)
EDITPSGX	DB2/DB2 Index	<p>Trading Partner Segment Usage Unique Index - Unique index created on the key value to EDITPSG. One of these exists for each EDITPSG entry.</p> <ul style="list-style-type: none"> TPSGXNUM = TPSGNUM
TPSGAIX1	VSAM/VSAM Index	<p>Trading Partner Segment Usage Alternate Index 1 - An alternate index created in the VSAM environment only, which keeps track of the number of times a particular segment has been mapped in the transaction. There will be as many TPSGAIX1 entries as there are EDIVTPSG entries.</p> <ul style="list-style-type: none"> TPSGAIX1NUM = TPSGNUM
TPSGAIX2	VSAM/VSAM Index	<p>Trading Partner Segment Usage Alternate Index 2 - An alternate index created in the VSAM environment only, which keeps track of the transactions that have a segment within a standard mapped. Worst case is there will be as many TPSGAIX2 entries as there are EDIVTPSG entries. This would not be the case if the same segment occurs more than once in the same transaction.</p> <ul style="list-style-type: none"> TPSGAIX2NUM = TPSGNUM

Table F-1 (Page 8 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIVTPST EDITPST	VSAM/VSAM File DB2/DB2 Table	<p>Trading Partner Send Usage - Records the use of a mapping by a particular trading partner. This is only used for transactions being sent and there will be one entry for each trading partner using this transaction with a given application data format ID and internal trading partner ID.</p> <p>Usually there is a single internal trading partner ID defined per trading partner.</p> <ul style="list-style-type: none"> • TPTXNUM = Total number of mappings • PSEND = Percentage of mappings (TPTXNUM) that are send mappings • TPPROFNUM = Total number of trading partners • TPSEND = Percentage of trading partners you send to • TPSTNUM = (TPTXNUM * PSEND) * (TPPROFNUM * TPSEND)
EDITPSTX	DB2/DB2 Index	<p>Trading Partner Send Usage Unique Index - Unique index created on the key value to EDITPST. One of these exists for each EDITPST entry.</p> <ul style="list-style-type: none"> • TPSTXNUM = TPSTNUM
TPSTAIX1	VSAM/VSAM Index	<p>Trading Partner Send Usage Alternate Index 1 - An alternate index created in the VSAM environment only, used to make sure there is only one usage per mapping for a given trading partner and internal trading partner ID. There will be one entry for each EDIVTPST entry unless the same internal trading partner ID and trading partner nickname is used in other EDIVTPST entries.</p> <ul style="list-style-type: none"> • TPSTAIX1NUM = TPSTNUM
TPSTAIX2	VSAM/VSAM Index	<p>Trading Partner Send Usage Alternate Index 2 - An alternate index created in the VSAM environment only, used by the translator to find the mapping that applies for the current transaction. There will be one entry for each EDIVTPST entry.</p> <ul style="list-style-type: none"> • TPSTAIX2NUM = TPSTNUM
TPSTAIX3	VSAM/VSAM Index	<p>Trading Partner Send Usage Alternate Index 3 - An alternate index created in the VSAM environment only, used by IEF to return a list of internal trading partner ID's for a given trading partner and application data format. There will be one full entry for each EDIVTDID entry that has been mapped and a 32 byte entry for all other EDIVTPST entries.</p> <ul style="list-style-type: none"> • TPSTAIX3NUM = TPSTNUM (should be close enough)
EDIVTPTX EDITPTX	VSAM/VSAM File DB2/DB2 Table	<p>Trading Partner Transaction - Contains information relative to the mapping between an application data format and a standard transaction. There will be one entry for each trading partner transaction created. A good starting point for this number might be the number of application data formats (EDIVTDID) that you have plus the number of standard transactions that you are interested in (EDIVSCTX).</p> <ul style="list-style-type: none"> • TPTXNUM = Number of mappings

Table F-1 (Page 9 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDITPTXX	DB2/DB2 Index	Trading Partner Transaction Unique Index - Unique index created on the key value to EDITPTX. One of these exists for each EDITPTX entry. <ul style="list-style-type: none">• TPTXXNUM = TPTXNUM
TPTXAIX1	VSAM/VSAM Index	Trading Partner Transaction Alternate Index 1 - An alternate index created in the VSAM environment only, which keeps track of where a standard is used. There will be one entry for each EDIVTPTX entry. <ul style="list-style-type: none">• TPTXAIX1NUM = TPTXNUM
TPTXAIX2	VSAM/VSAM Index	Trading Partner Transaction Alternate Index 2 - An alternate index created in the VSAM environment only used to keep track of where a standard transaction is used. There will be one entry for each EDIVTPTX entry. <ul style="list-style-type: none">• TPTXAIX2NUM = TPTXNUM
TPTXAIX3	VSAM/VSAM Index	Trading Partner Transaction Alternate Index 3 - An alternate index created in the VSAM environment only used to keep track of where an application data format is used. At worst there will be one full entry for each EDIVTDID entry that has been mapped and one 16 byte entry for all other EDIVTPTX entries. <ul style="list-style-type: none">• TPTXAIX3NUM = TPTXNUM
EDIVTDID EDITDID	VSAM/VSAM File DB2/DB2 Table	Application Data Format Definition - Contains information about an application data format. There is one entry for each data format defined. An estimate of the number of input/output data definitions contained in the applications that are going to be EDI enabled. This corresponds to a 01 level-number in COBOL. <ul style="list-style-type: none">• TDIDNUM = Number of application data formats
EDITDIDX	DB2/DB2 Index	Application Data Format Definition Unique Index - Unique index created on the key value to EDITDID. One of these exists for each EDITDID entry. <ul style="list-style-type: none">• TDIDXNUM = TDIDNUM
EDIVTDST EDITDST	VSAM/VSAM File DB2/DB2 Table	Application Data Format Structure/Field Definition - Contains information about the definition of a field or structure within an application data format. There are two entries for each structure defined and one entry for each field defined. A structure correlates to a COBOL group item and a field correlates to a COBOL elementary item. <ul style="list-style-type: none">• TDIDNUM = Number of application data formats• STFMT = Average number of structures per data format• FLSTR = Average number of fields per structure• TDSTNUM = ((STFMT * 2) + (STFMT * FLSTR)) * TDIDNUM
EDITDSTX	DB2/DB2 Index	Application Data Format Structure/Field Definition Unique Index - Unique index created on the key value to EDITDST. One of these exists for each EDITDST entry. <ul style="list-style-type: none">• TDSTXNUM = TDSTNUM

Table F-1 (Page 10 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIPROF	VSAM/VSAM File	Trading Partner Profile Control Numbers - One entry exists for each sender/receiver combination actually used. <ul style="list-style-type: none"> PROFNUM = number of different sender/receiver combinations used.
EDIPROF	DB2/DB2 Table	
EDIPROFX	DB2/DB2 Index	Trading Partner Profile Control Number Table Index - One row exists for each sender/receiver pair combination actually used. <ul style="list-style-type: none"> PROFXNUM = PROFNUM
EDIPSAC	DB2/DB2 Table	Activity Log Profile - One entry exists for each log file; DataInterchange creates two log files; additional log files are created by the user. <ul style="list-style-type: none"> PSACNUM = 2 + number of user log files
EDIPSACX	DB2/DB2 Index	Activity Log Unique Index - One entry exists for each Activity Log Profile. <ul style="list-style-type: none"> PSACXNUM = PSACNUM
EDIPSAD	DB2/DB2 Table	User Exit Information Profile - One entry exists for each user exit referenced; DataInterchange creates two entries for its own use; additional entries are created by the user to identify user exits utilized by the installation. <ul style="list-style-type: none"> PSADNUM = 2 + number of user installation exit programs
EDIPSADX	DB2/DB2 Index	User Exit Profile Unique Index - One entry for each User Exit Profile. <ul style="list-style-type: none"> PSADXNUM = PSADNUM
EDIPSAP	DB2/DB2 Table	Application Definition Profile - This profile is used to establish settings at an application level - as opposed to the system level - for an invocation of DataInterchange; one entry exists for each application ID. <ul style="list-style-type: none"> PSAPNUM = 2 + number of user-specified applications
EDIPSAPX	DB2/DB2 Index	Application Definition Profile Unique Index - One entry for each Application Definition Profile. <ul style="list-style-type: none"> PSAPXNUM = PSAPNUM
EDIPSCR	DB2/DB2 Table	Continuous Receive Profile - One entry exists for each unique path by which data is received from a VAN. <ul style="list-style-type: none"> PSCRNUM = Number of paths to VAN
EDIPSCRX	DB2/DB2 Index	Continuous Receive Unique Index - One entry for each Continuous Receive Profile. <ul style="list-style-type: none"> PSCRXNUM = PSCRNUM
EDIPSDI	DB2/DB2 Table	DataInterchange Control File - This table is static.
EDIPSDIX	DB2/DB2 Index	DataInterchange Control File Unique Index - This table is static.
EDIPSLG	DB2/DB2 Table	Log Data Profile - One entry exists for each Activity Log Profile defined. <ul style="list-style-type: none"> PSLGNUM = PSACNUM

Table F-1 (Page 11 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIPSLGX	DB2/DB2 Index	Log Data Profile Unique Index - One entry for each Log Data Profile. <ul style="list-style-type: none"> PSLGXNUM = PSLGNUM
EDIPSLP	DB2/DB2 Table	Language Profile - One entry exists for each language used; presently only English (ENU) is supported. <ul style="list-style-type: none"> PSLPNUM = 1
EDIPSLPX	DB2/DB2 Index	Language Profile Unique Index - One entry for each Language Profile. <ul style="list-style-type: none"> PSACXNUM = PSACNUM
EDIPSNO	DB2/DB2 Table	Network Operation Profile - One entry exists for each line of a Network Operation statement (or Network Command); DataInterchange supplies several commands to support the Continuous Receive process. <ul style="list-style-type: none"> PSNONUM = 200 + number of user-supplied statements
EDIPSNOX	DB2/DB2 Index	Network Operation Profile Log Unique Index - One entry for each Network Operation Profile. <ul style="list-style-type: none"> PSNOXNUM = PSNONUM
EDIPSNP	DB2/DB2 Table	Network Profile - One entry for each network defined to the system; DataInterchange supplies the definition of eight networks. <ul style="list-style-type: none"> PSNPNUM = 8 + number of additional networks
EDIPSNPX	DB2/DB2 Index	Network Profile Unique Index - One entry for each Network Profile. <ul style="list-style-type: none"> PSNPXNUM = PSNPNUM
EDIPSPD	DB2/DB2 Table	Profile Definitions - This table is static.
EDIPSPDX	DB2/DB2 Index	Profile Definitions Unique Index - One entry for each Profile Definitions Profile. <ul style="list-style-type: none"> PSPDXNUM = PSPDNUM
EDIPSRQ	DB2/DB2 Table	Requestor Profile - One entry exists for each mailbox used to receive data. <ul style="list-style-type: none"> PSRQNUM = Number of mailboxes
EDIPSRQX	DB2/DB2 Index	Requestor Profile Unique Index - One entry for each Requestor Profile. <ul style="list-style-type: none"> PSRQXNUM = PSRQNUM
EDIPSSP	DB2/DB2 Table	Security Profile - One entry exists for each grouping of security processing; the names of exit programs to be utilized for encryption, authorization, filtering, and compression are specified in a named security profile. <ul style="list-style-type: none"> PSSPNUM = Number of security processing groups
EDIPSSPX	DB2/DB2 Index	Security Profile Unique Index - One entry for each System Profile. <ul style="list-style-type: none"> PSSPXNUM = PSSPNUM

Table F-1 (Page 12 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIPSSY	DB2/DB2 Table	System Profile - One entry exists for each group of settings of a CICS Persistent Environment; Persistent Environment provides a means of improving performance in CICS. <ul style="list-style-type: none"> • PSSYNUM = Number of setting groupings
EDIPSSYX	DB2/DB2 Index	System Profile Unique Index - One entry for each System Profile <ul style="list-style-type: none"> • PSSYXNUM = PSSYNUM
EDIPSTD	DB2/DB2 Table	Table Definitions - One entry exists for each validation and translation table in DataInterchange; DataInterchange provides 49 tables at installation; importing standards will increase the number of tables by the number of Code Lists accompanying the Standard. <ul style="list-style-type: none"> • PSTDNUM = 49 + user-created or imported tables
EDIPSTDY	DB2/DB2 Index	Table Definitions Unique Index - One entry for each Table Definition entry. <ul style="list-style-type: none"> • PSTDXNUM = PSTDNUM
EDIPSTP	DB2/DB2 Table	Trading Partner Profile - One record exists for each trading partner. <ul style="list-style-type: none"> • PSTPNUM = Number of trading partners
EDIPSTPX	DB2/DB2 Index	Trading Partner Profile Unique Index - One entry for each Trading Partner Profile. <ul style="list-style-type: none"> • PSTPXNUM = PSTPNUM
EDIPSTP1	DB2/DB2 Index	Trading Partner Profile Unique Index - One entry for each Trading Partner Profile. <ul style="list-style-type: none"> • PSTP1NUM = PSTPNUM
EDIPSTP2	DB2/DB2 Index	Trading Partner Profile Unique Index - One entry for each Trading Partner Profile. <ul style="list-style-type: none"> • PSTP2NUM = PSTPNUM
EDIPSTP3	DB2/DB2 Index	Trading Partner Profile Unique Index - One entry for each Trading Partner Profile. <ul style="list-style-type: none"> • PSTP3NUM = PSTPNUM
EDIPSTT	DB2/DB2 Table	Translation Table - One record exists for each code of a translation table; DataInterchange supplies 953 values in the 49 tables loaded at install. <ul style="list-style-type: none"> • PSTTNUM = (Average number of values * PSTDNUM) + 953
EDIPSTTX	DB2/DB2 Index	Translation Table Entry Unique Index - One entry for each translation-table entry. <ul style="list-style-type: none"> • PSTTXNUM = PSTTNUM
EDIPSTT1	DB2/DB2 Index	Translation Table Entry Unique Index - One entry for each translation-table entry. <ul style="list-style-type: none"> • PSTT1NUM = PSTTNUM

Table F-1 (Page 13 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIPSTV	DB2/DB2 Table	Validation Table - One record exists for each validation table; DataInterchange supplies three tables at installation with a total number of 294 value entries. <ul style="list-style-type: none"> • $PSTVNUM = (Average\ number\ of\ values * PSTDNUM) + 294$
EDIPSTVX	DB2/DB2 Index	Validation Table Entry Unique Index - One entry for each validation-table entry. <ul style="list-style-type: none"> • $PSTVXNUM = PSTVNUM$
EDIVBDDE EDIBDDE	VSAM/VSAM file DB2/DB2 Table	Business Document Table
EDIBDDEX	DB2/DB2 Index	Business Document Unique Index
BDDEAIX1	VSAM/VSAM Index	Business Document Alternate Index
BDDEAIX2	VSAM/VSAM Index	Business Document Alternate Index
EDIVBDDF EDIBDDF	VSAM/VSAM File DB2/DB2 Table	Business Document Definition Table
EDIBDDFX	DB2/DB2 Index	Business Document Definition Unique Index
EDIVBDDT EDIBDDT	VSAM/VSAM File DB2/DB2 Table	Business Document Data Table
EDIBDDTX	DB2/DB2 Index	Business Document Data Unique Index
EDIVBDFR EDIBDFR	VSAM/VSAM File DB2/DB2 Table	Business Document File Request Table
EDIBDFRX	DB2/DB2 Index	Business Document File Request Unique Index
BDFRAIX1	VSAM/VSAM Index	Business Document File Request Alternate Index
EDIVBDMS EDIBDMS	VSAM/VSAM File DB2/DB2 Table	Business Document Message Table
EDIBDMSX	DB2/DB2 Index	Business Document Message Unique Index
BDMSAIX1	VSAM/VSAM Index	Business Document Message Alternate Index
EDIOWNR	DB2/DB2 Table	Management Reporting Table Owner User ID
EDIVTPCM EDITPCM	VSAM/VSAM File DB2/DB2 Table	Comment Table - One record exists for each trading partner with comment data keyed in the DataInterchange Client system; if DataInterchange Client is not used, this table is empty. <ul style="list-style-type: none"> • $TPCMNUM = PSTPNUM$
EDITPCMX	DB2/DB2 Index	Comment Table Unique Index <ul style="list-style-type: none"> • $TSPCMXNUM = TPCMNUM$
EDIVTPCN EDITPCN	VSAM/VSAM File DB2/DB2 Table	Trading Partner Comment Table - One record exists for each Trading Partner / Contact relationship; if DataInterchange Client is not used, this table is empty. <ul style="list-style-type: none"> • $TPCNNUM = PSTPNUM * average\ number\ of\ contacts\ per\ trading\ partner$
EDITPCNX	DB2/DB2 Index	Trading Partner Contact Table Unique Index <ul style="list-style-type: none"> • $TPCNXNUM = TPCNNUM$

Table F-1 (Page 14 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
TPCNAIX1	VSAM/VSAM Index	Trading Partner Contact Alternate Index - One record shows all trading partners for a given contract. <ul style="list-style-type: none"> • TPCN1NUM = TPCTNUM
EDIVTPCT EDITPCT	VSAM/VSAM File DB2/DB2 Table	Contact Table - One record exists for each contact; if DataInterchange Client is not used, this table is empty. <ul style="list-style-type: none"> • TPCMNUM = Number of contacts
EDITPCTX	DB2/DB2 Index	Contact Table Unique Index <ul style="list-style-type: none"> • TSPCTXNUM = TPCTNUM
EDITSLT	DB2/DB2 Table	Transaction Store Lock Table - This table has only one record; it is used as a "lock" table when update activity involves the Transaction Store. <ul style="list-style-type: none"> • TSLTNUM = 1
EDIMSGS	DB2/DB2 Table	Message Table - Contains the text for all messages issued from DataInterchange. This is a static table except perhaps during PTF applies when messages may be added or updated. <ul style="list-style-type: none"> • MSGNUM = 2448 (for base 3.1)
EDIMSGSX	DB2/DB2 Index	Message Table Index - One row exists for each DataInterchange message. <ul style="list-style-type: none"> • MSGSXNUM = MSGSNUM
EDIELOG	DB2/DB2 Table	Event Log Table - Contains all the event log entries for all the application IDs used. <ul style="list-style-type: none"> • ELOGNUM = Number of total event log entries
EDIELOGX	DB2/DB2 Index	Event Log Table Index - One row exists for each event log entry by application ID. <ul style="list-style-type: none"> • ELOGXNUM = ELOGNUM
EDIELOG1	DB2/DB2 Index	Event Log Table Index - One row exists for each event log entry by application ID and user ID. <ul style="list-style-type: none"> • ELOG1NUM = ELOGNUM
EDIPSEE	DB2/DB2 Table	E Envelope Profile Table - One row exists for each E envelope profile member (EDIFACT). <ul style="list-style-type: none"> • PSEENUM - Number of E envelope profile members
EDIPSEEX	DB2/DB2 Index	E Envelope Profile Table Index - One row exists for each E envelope profile member (EDIFACT). <ul style="list-style-type: none"> • PSEEXNUM = PSEENUM
EDIPSIE	DB2/DB2 Table	I Envelope Profile Table - One row exists for each I envelope profile member (ICS). <ul style="list-style-type: none"> • PSIENUM = Number of I envelope profile members
EDIPSIEX	DB2/DB2 Index	I Envelope Profile Table Index - One row exists for each I envelope profile member (ICS). <ul style="list-style-type: none"> • PSIEXNUM = PSIENUM
EDIPSIE	DB2/DB2 Table	T Envelope Profile Table - One row exists for each T envelope profile member (UN/TDI). <ul style="list-style-type: none"> • PSTENUM = Number of T envelope profile members

Table F-1 (Page 15 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIPSTEX	DB2/DB2 Index	T Envelope Profile Table Index - One row exists for each T envelope profile member (UN/TDI). <ul style="list-style-type: none"> • PSTEXNUM = PSTENUM
EDIPSUE	DB2/DB2 Table	U Envelope Profile Table - One row exists for each U envelope profile member (UCS). <ul style="list-style-type: none"> • PSUENUM = Number of U envelope profile members
EDIPSUEX	DB2/DB2 Index	U Envelope Profile Table Index - One row exists for each U envelope profile member (UCS). <ul style="list-style-type: none"> • PSUEXNUM = PSUENUM
EDIPSXE	DB2/DB2 Table	X Envelope Profile Table - One row exists for each X envelope profile member (X12). <ul style="list-style-type: none"> • PSXENUM = Number of X envelope profile members
EDIPSXEX	DB2/DB2 Index	X Envelope Profile Table Index - One row exists for each X envelope profile member (X12). <ul style="list-style-type: none"> • PSXEXNUM = PSXENUM
EDIPSMQ	DB2/DB2 Table	MQSeries Profile Table - One row for each MQSeries profile member. <ul style="list-style-type: none"> • PSMQNUM = Number of MQSeries profile members
EDIPSMQX	DB2/DB2 Index	MQSeries Profile Table Index - One row for each MQSeries profile member. <ul style="list-style-type: none"> • PSMQXNUM = PSMQNUM

Note:

Transaction Store Tables

The following tables are all part of the Transaction Store. The number of entries in the Transaction Store is dependent on two major factors.

1. The length of time transactions remain in the database before being purged. This value can be provided when transactions are added to the Transaction Store and if not provided defaults to 30 days.
2. The number of days between runnings of the Transaction Store remove utility.
 - TRXLIFE = number of days before a transaction may be purged
 - PRGSPAN = number of days between running purge utility
 - TSLIFE = TRXLIFE + PGRSPAN

Along with TRXLIFE and PRGSPAN the other important number to estimate is the number of transactions per day that you will be processing. All of the examples below are based on knowing these three pieces of information.

EDIVTSTH	VSAM/VSAM File	Transaction Store Transaction Handle - Contains detailed information relative to a transaction. There is one entry for each transaction send translated or de-enveloped <ul style="list-style-type: none"> • TRXPDAY = Number of transactions per day • TSTHNUM = TRXPDAY * TSLIFE
EDITSTH	DB2/DB2 Table	
EDITSTHX	DB2/DB2 Index	Transaction Store Transaction Handle Unique Index - Unique index created on the key value to EDITSTH. One of these exists for each EDITSTH entry. <ul style="list-style-type: none"> • TSTHXNUM = TSTHNUM

Table F-1 (Page 16 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDITSTHO	DB2/DB2 Index	Transaction Store Table Unique Index <ul style="list-style-type: none"> TSTHONUM = TSTHNUM
TSTHAIX1	VSAM/VSAM Index	Transaction Store Transaction Handle Alternate Index 1 - An alternate index created in the VSAM environment only, which keeps track of the transactions that use a particular set of envelope override values. There will be one entry for each EDIVTSTH entry. <ul style="list-style-type: none"> TSTHAIX1NUM = TSTHNUM
TSTHAIX2	VSAM/VSAM Index	Transaction Store Transaction Handle Alternate Index 2 - An alternate index created in the VSAM environment only, which keeps track of bundles. There will be one entry for each EDIVTSTH entry. <ul style="list-style-type: none"> TSTHAIX2NUM = TSTHNUM
EDIVSSTK EDISSTK	VSAM/VSAM File DB2/DB2 Table	SAP tracking file - Maintains SAP control information and EDI subsystem status to create the SAP status record. <ul style="list-style-type: none"> SSTKNUM = TSTHNUM
EDISSTKX	DB2/DB2 Table	SAP Tracking Alternate Index - One record for each SAP record <ul style="list-style-type: none"> SSTKXNUM = SSTKNUM
SSTKAIX1	VSAM/VSAM File	SAP tracking file alternate index. An alternate index used to maintain the EDIVSSTK file. <ul style="list-style-type: none"> SSTKAIXNUM = TSTHNUM
EDISSTXX	DB2/DB2 Table	SAP tracking file alternate index. An alternate index used to maintain the EDISSTK table. <ul style="list-style-type: none"> SSTXXNUM = TSTHNUM
EDIVTSTI EDITSTI	VSAM/VSAM File DB2/DB2 Table	Transaction Store Transaction Image - Contains the STANDARD transaction image for a transaction. There will be one entry for each EDIVTSTH entry. <ul style="list-style-type: none"> TSTINUM = TSTHNUM
EDITSTIX	DB2/DB2 Index	Transaction Store Transaction Image Unique Index - Unique index created on the key value to EDITSTI. One of these exists for each EDITSTI entry. <ul style="list-style-type: none"> TSTIXNUM = TSTINUM
EDIVTSTO EDITSTO	VSAM/VSAM File DB2/DB2 Table	Transaction Store Transaction Override - Contains the envelope override values for a particular set of transactions. This only applies to send transactions and only for those transactions that supply override values in the C record or through the application programming interface. The number of these records is either very easy to estimate or extremely difficult. If overrides are not used then the number is 0 (easy). If overrides are used then you have at least one record for each PERFORM TRANSLATE TO STANDARD executed. If the value of the overrides change then you will have one record each time any of the overrides change in value from one transaction to the next. <ul style="list-style-type: none"> TSTONUM = 0

Table F-1 (Page 17 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDITSTOX	DB2/DB2 Index	Transaction Store Transaction Override Unique Index - Unique index created on the key value to EDITSTO. One of these exist for each EDITSTO entry. <ul style="list-style-type: none">• TSTOXNUM = TSTONUM
EDIVTSAU EDITSAU	VSAM/VSAM File DB2/DB2 Table	Transaction Store Application Usage - Contains information relative to the translation of a transaction by an application. There will be one entry for each EDIVTSTH entry unless there are a lot of transactions that are de-enveloped but never translated. <ul style="list-style-type: none">• TSAUNUM = TSTHNUM
EDITSAUX	DB2/DB2 Index	Transaction Store Application Usage Unique Index X - Unique index created on the key value to EDITSAU. One of these exists for each EDITSAU entry. <ul style="list-style-type: none">• TSAUXNUM = TSAUNUM
EDITSAUY	DB2/DB2 Index	Transaction Store Application Usage Index Y - Index created on the BATCHID value used to retrieve transactions with a given BATCHID. There will be one entry per EDITSAU entry with a different BATCHID value. <ul style="list-style-type: none">• TSAUYNUM = TSAUNUM
TSAUAIX1	VSAM/VSAM Index	Transaction Store Application Usage Alternate Index 1 - An alternate index created in the VSAM environment only, used to retrieve transactions with a given BATCHID. Good assumption is there will be one entry per EDIVTSAU entry unless there is a lot of RETRANSLATE activity that occurs with the same BATCHID value. <ul style="list-style-type: none">• TSAUAIX1NUM = TSAUNUM
TSAUAIX2	VSAM/VSAM Index	Transaction Store Application Usage Alternate Index 2 - An alternate index created in the VSAM environment only, used to keep track of transactions with a given APPLID. Good assumption is there will be one entry per EDIVTSAU entry unless there is a lot of RETRANSLATE activity that occurs with the same APPLID value. <ul style="list-style-type: none">• TSAUAIX2NUM = TSAUNUM
EDIVTSEV EDITSEV	VSAM/VSAM File DB2/DB2 Table	Transaction Store Transaction Envelope - An envelope contains information about an interchange. There will be one entry created any time an envelope or de-envelope function is requested. <ul style="list-style-type: none">• TRXPENV = Average number of transactions per envelope• TSEVNUM = TSTHNUM / TRXPENV
EDITSEVX	DB2/DB2 Index	Transaction Store Transaction Envelope Unique Index - Unique index created on the key value to EDITSEV. One of these exists for each EDITSEV entry. <ul style="list-style-type: none">• TSEVXNUM = TSEVNUM
EDITSEV1	DB2/DB2 Table	Transaction Store Envelope Unique Index <ul style="list-style-type: none">• TSEV1NUM = TSEVNUM

Table F-1 (Page 18 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
TSEVAIX1	VSAM/VSAM Index	<p>Transaction Store Transaction Envelope Alternate Index 1 - An alternate index created in the VSAM environment only, defined on trading partner nickname, message user class, message name , and message sequence number. It is used to retrieve an envelope for network status update operations. There will be one entry per EDIVTSEV entry.</p> <ul style="list-style-type: none"> • TSEVAIX1NUM = TSEVNUM
TSEVAIX2	VSAM/VSAM Index	<p>Transaction Store Transaction Envelope Alternate Index 2 - An alternate index created in the VSAM environment only, defined on message ID concatenated with the EDIVTSEV primary key. It is used to retrieve an envelope for network status update operations. There will be one entry per EDIVTSEV entry.</p> <ul style="list-style-type: none"> • TSEVAIX2NUM = TSEVNUM
EDIVTSGP EDITSGP	VSAM/VSAM File DB2/DB2 Table	<p>Transaction Store Group - Maintains information about a functional group. There will be at least one entry for each interchange sent or received.</p> <ul style="list-style-type: none"> • GRPPENV = Average number of groups per envelope (minimum of 1) • TSGPNUM = TSEVNUM * GRPPENV
EDITSGPX	DB2/DB2 Index	<p>Transaction Store Group Unique Index - Unique index created on the key value to EDITSGP. One of these exists for each EDITSGP entry.</p> <ul style="list-style-type: none"> • TSGPXNUM = TSGPNUM
EDIVTSTU EDITSTU	VSAM/VSAM File DB2/DB2 Table	<p>Transaction Store Transaction Usage - Maintains information about the use of a transaction in an interchange. There will be one entry each time a transaction is added to or extracted from an interchange. Good assumption is one entry for each EDIVTSTH entry unless there are a lot of transactions with translation errors or are never enveloped for some other reason.</p> <ul style="list-style-type: none"> • TSTUNUM = TSTHNUM
EDITSTUX	DB2/DB2 Index	<p>Transaction Store Transaction Usage Unique Index X - Unique index created on the key value to EDITSTU. One of these exists for each EDITSTU entry.</p> <ul style="list-style-type: none"> • TSTUXNUM = TSTUNUM
EDITSTUY	DB2/DB2 Index	<p>Transaction Store Transaction Usage Index Y - Index created on the transaction handle value used to keep track of which transaction is being enveloped. Good assumption is there will be one of these for each EDITSTU entry unless a lot of REENVELOPE activity occurs.</p> <ul style="list-style-type: none"> • TSTUYNUM = TSTUNUM

Table F-1 (Page 19 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
TSTUAIX1	VSAM/VSAM Index	<p>Transaction Store Transaction Usage Alternate Index 1 - An alternate index created in the VSAM environment only, used to link an EDIVTSTU entry with an EDIVTSTH entry. There will be one entry per EDIVTSTU entry.</p> <ul style="list-style-type: none"> • TSTUAIX1NUM = TSTUNUM <p>Management Reporting Tables</p> <p>The following database records are all part of Management Reporting. If the Management Reporting option is specified as 'N' (no) for the active Application Definition Profile (APPDEFS), no statistic entries will be generated. If the Management Reporting option is specified as 'Y' (yes), then statistic entries will be generated.</p> <p>Management Reporting database records are defined for the major categories of 'measurements' and 'pending'.</p> <p>Measurement records reflect statistics current as of the effective date of the last PERFORM UPDATE STATISTICS command that was processed. Pending records contain statistic entries that have been created by DI processes, but have not yet been updated to the measurement records by the PERFORM UPDATE STATISTICS process.</p> <p>The number of entries in the Measurement database is dependent on two major factors.</p> <ol style="list-style-type: none"> 1. MEASLIFE - the number of days that entries are stored in the measurement database before they are purged during the PERFORM REMOVE STATISTICS process. 2. MEASPDAY - The number of statistic entries that are written to the measurement database per EDI processing day. <p>The number of entries in the Pending database is dependent on two major factors.</p> <ol style="list-style-type: none"> 1. PENDLIFE - the number of days that entries are stored in the pending database before they are purged during the PERFORM UPDATE STATISTICS process. 2. PENDPDAY - The number of statistic entries that are written to the pending database per EDI processing day. <p>All of the examples below are based on knowing these pieces of information.</p>

Table F-1 (Page 20 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIVMRCM EDIMRCM	VSAM/VSAM File DB2/DB2 Table	<p>Management Reporting Communications Measurement - Contains measurement information relative to an interchange. For each active Requestor ID, you can have two cumulative entries (send & receive). Also, each Requestor ID can have a maximum of two daily entries (one send and one receive) per EDI processing day. If a Requestor ID has multiple interchanges during a day, the daily entry created for that day will be updated for each interchange.</p> <ul style="list-style-type: none"> • RIDNUM = Number of Requestor ID's (RID) • CUMENTS = Number of cumulative entries = RIDNUM * 2 • PRIDINT = Daily percentage of RID's having an interchange • MEASPDAY = (RIDNUM * 2) * PRIDINT • MRCMNUM = (MEASPDAY * MEASLIFE) + CUMENTS
EDIMRCMX	DB2/DB2 Index	<p>Management Reporting Communications Measurement Unique Index - Unique index created on the key value to EDIMRCM. One of these exists for each EDIMRCM entry.</p> <ul style="list-style-type: none"> • MRCMXNUM = MRCMNUM
EDIVMRPC EDIMRPC	VSAM/VSAM File DB2/DB2 Table	<p>Management Reporting Pending Communications - Contains pending measurement information relative to an interchange. When executing the PERFORM UPDATE STATISTICS command, this data is applied to the EDIVMRCM database and then is purged. There is no cumulative record concept in the pending database, so for each active Requestor ID, you can have multiple send and receive entries per EDI processing day.</p> <ul style="list-style-type: none"> • RIDNUM = Number of Requestor ID's (RID) • SENDNUM = Avg # of send interchanges per RID per day • RECVNUM = Avg # of receive interchanges per RID per day • PENDPDAY = (SENDNUM + RECVNUM) * RIDNUM • MRPCNUM = PENDPDAY * Pendlife
EDIMRPCX	DB2/DB2 Index	<p>Management Reporting Pending Communications Unique Index - Unique index created on the key value to EDIMRPC. One of these exists for each EDIMRPC entry.</p> <ul style="list-style-type: none"> • MRPCXNUM = MRPCNUM
EDIVMRRT EDIMRRT	VSAM/VSAM File DB2/DB2 Table	<p>Management Reporting Receive Usage Measurement - Contains measurement information relative to a receive usage. For each active receive usage you can have one cumulative entry, plus one daily entry per EDI processing day. This table is directly related to the EDIVTPRT table. Use the table entry to see how to calculate TPRTNUM, total number of receive usages. The following calculations assume you will have at least one transaction per active receive usage (EDIVTPRT entry) per day.</p> <ul style="list-style-type: none"> • PACTIVE = Daily percentage of receive usages having activity • MEASPDAY = TPRTNUM * PACTIVE • MRRTNUM = (MEASPDAY * MEASLIFE) + TPRTNUM

Table F-1 (Page 21 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIMRRTX	DB2/DB2 Index	<p>Management Reporting Receive Usage Measurement Unique Index - Unique index created on the key value to EDIMRRT. One of these exists for each EDIMRRT entry.</p> <ul style="list-style-type: none"> • $MRRTXNUM = MRRTNUM$
EDIVMRPR EDIMRPR	VSAM/VSAM File DB2/DB2 Table	<p>Management Reporting Pending Receive Usage - Contains pending measurement information relative to a receive usage. When executing the PERFORM UPDATE STATISTICS command, this data is applied to the EDIVMRRT database and then is purged. There is no cumulative record concept in the pending database, so for each active receive usage, you can have multiple receive entries per EDI processing day.</p> <p>This table is hard to size because the size of the table depends on the RECOVERY scope that is used. RECOVERY scope is specified during translations and you can either have a transaction recovery scope or an interchange recovery scope. A transaction recovery scope is easiest to estimate because with transaction recovery you have an EDIVMRPR entry for each received transaction. With an interchange recovery scope you will have one entry for each receive usage that is used within the interchange. As an example, suppose that an interchange contains 100 transactions. With transaction recovery scope, 100 EDIVMRPS entries would be created (easy). With an interchange recovery scope it could be as few as 1 EDIVMRPR entry (all transactions with the same receive usage) or as many as 100 EDIVMRPR entries (each transaction has a different receive usage). Two formulas are given below. One that assumes transaction level recovery and one that assumes interchange level recovery.</p> <p>TRANSACTION LEVEL RECOVERY:</p> <ul style="list-style-type: none"> • $TRXPDAY$ = Number of transactions per day. This value is also used in space calculations for EDIVTSTH • $PRECV$ = Percentage of transactions that are receive transactions • $MRPRNUM = TRXPDAY * PRECV * PENDLIFE$ <p>INTERCHANGE LEVEL RECOVERY:</p> <ul style="list-style-type: none"> • $TRXPDAY$ = Number of transactions per day. This value is also used in space calculations for EDIVTSTH • $PRECV$ = Percentage of transactions that are receive transactions • $TRXPENV$ = Average number of transactions per interchange. This value is also used in space calculations for EDIVTSEV • $ENVPDAY = \text{Number of interchanges received per day} = (TRXPDAY * PRECV) / TRXPENV$ • $AVGUSGS$ = Average number of usages that occur in each interchange • $MRPRNUM = ENVPDAY * AVGUSGS * PENDLIFE$
EDIMRPRX	DB2/DB2 Index	<p>Management Reporting Pending Receive Usage Unique Index - Unique index created on the key value to EDIMRPR. One of these exists for each EDIMRPR entry.</p> <ul style="list-style-type: none"> • $MRPRXNUM = MRPRNUM$

Table F-1 (Page 22 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIVMRST EDIMRST	VSAM/VSAM File DB2/DB2 Table	<p>Management Reporting Send Usage Measurement - Contains measurement information relative to send usage of a trading partner mapping transaction ID. For each active send usage you can have one cumulative entry, plus one daily entry per EDI processing day. This table is directly related to the EDIVTPST table. Use the table entry to see how to calculate TPSTNUM, total number of send usages. The following calculations assume you will have at least one transaction per active send usage (EDIVTPST entry) per day.</p> <ul style="list-style-type: none"> • PACTIVE = daily percentage of send usages having activity • MEASPDAY = TPSTNUM * PACTIVE • MRSTNUM = (MEASPDAY * MEASLIFE) + TPSTNUM
EDIMRSTX	DB2/DB2 Index	<p>Management Reporting Send Usage Measurement Unique Index - Unique index created on the key value to EDIMRST. One of these exists for each EDIMRST entry.</p> <ul style="list-style-type: none"> • MRSTXNUM = MRSTNUM

Table F-1 (Page 23 of 23). DataInterchange Database Records

Record Name	DI Product/DB Type	Description
EDIVMRPS EDIMRPS	VSAM/VSAM File DB2/DB2 Table	<p>Management Reporting Pending Send Usage - Contains pending measurement information relative to a send usage. When executing the PERFORM UPDATE STATISTICS command, this data is applied to the EDIVMRST database and then is purged. There is no cumulative record concept in the pending database, so for each active send usage, you can have multiple send entries per EDI processing day.</p> <p>This table is hard to size because the size of the table depends on the RECOVERY scope that is used. RECOVERY scope is specified during translations and you can either have a transaction recovery scope or an interchange recovery scope. A transaction recovery scope is easiest to estimate because with transaction recovery you have an EDIVMRPS entry for each send transaction. With an interchange recovery scope you will have one entry for each send usage that is used within the interchange. As an example, suppose that an interchange contains 100 transactions. With transaction recovery scope, 100 EDIVMRPS entries would be created (easy). With an interchange recovery scope it could be as few as 1 EDIVMRPS entry (all transactions with the same send usage) or as many as 100 EDIVMRPS entries (each transaction has a different send usage). Two formulas are given below. One that assumes transaction level recovery and one that assumes interchange level recovery.</p> <p>TRANSACTION LEVEL RECOVERY:</p> <ul style="list-style-type: none"> • $TRXPDAY$ = Number of transactions per day. This value is also used in space calculations for EDIVTSTH. • $PSEND$ = Percentage of transactions that are send transactions • $MRPSNUM = TRXPDAY * PSEND * PENDLIFE$ <p>INTERCHANGE LEVEL RECOVERY:</p> <ul style="list-style-type: none"> • $TRXPDAY$ = Number of transactions per day. This value is also used in space calculations for EDIVTSTH. • $PSEND$ = Percentage of transactions that are send transactions • $TRXPENV$ = Average number of transactions per interchange. This value is also used in space calculations for EDIVTSEV. • $ENVPDAY = \text{Number of interchanges sent per day} = (TRXPDAY * PSEND) / TRXPENV$ • $AVGUSGS$ = Average number of usages that occur in each interchange. • $MRPSNUM = ENVPDAY * AVGUSGS * PENDLIFE$
EDIMRPSX	DB2/DB2 Index	<p>Management Reporting Pending Send Usage Unique Index - Unique index created on the key value to EDIMRPS. One of these exists for each EDIMRPS entry.</p> <ul style="list-style-type: none"> • $MRPSXNUM = MRPSNUM$

DataInterchange Allocation Tables

Table F-2 and Table F-3 on page F-27 describe the DB2 and VSAM allocation parameters shipped with DataInterchange. They also show the two allocation values. The secondary parameter supports allocation of additional storage up to a maximum of 123 extents.

Note: These tables also provide information used in the “Space Calculation Scenario” on page F-29.

Supplied Database Allocation for DB2

Table F-2 describes all database records used by DataInterchange. It also shows the number of records the allocation values will accommodate.

Formula

This formula was used to determine the number of records per page:

rsz = Size of a record

ups = usable page size = 4074

rpp = Records per page = $\text{FLOOR}(\text{ups}/\text{rsz})$

This formula was used to determine the number of records per page for the index tables:

ksz = Size of the index (record size)

ups = Usable page size = 4067

spp = Number of sub-pages = 8

rpp = Records per page = $\text{FLOOR}((\text{ups} - (\text{spp} * (\text{ksz} + 21))) / (\text{ksz} + 4))$

This formula was used to determine the number of records in the Primary Allocation for non-index tables:

nkb = Number of Kilobytes in Primary Allocation

rpa = Number of records in Primary Allocation = $((\text{nkb} / 4) - 2) * \text{rpp}$

Notes:

1. FLOOR specifies the operation of discarding the fractional part of a number, as shown in Table F-2, Table F-3 on page F-27, Table F-4 on page F-30, Table F-5 on page F-35.
2. CEILING specifies the operation of taking the next largest integer, if the number has a fractional part, as shown in Table F-5 on page F-35.
3. Table 250 calculations are made assuming PCTFREE=0 and FREEPAGE=0.
4. DB/2 will always have at least one data page. If the $(\text{nkb}/4) - 2$ is less than one, one will be used in the calculation.

Table F-2 (Page 1 of 4). DataInterchange Supplied Database Allocation for DB2

Table Name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per Page	Size of Records	Records in Primary Allocation
edibdde	480	240	21	187	2478
edibddex	176	88	60	53	2520
edibddf	480	240	23	175	2714
edibddfx	64	32	188	16	2632
edibddt	480	240	35	115	4130
edibddtx	200	100	59	54	2832
edibdfr	480	240	35	115	4130

Table F-2 (Page 2 of 4). DataInterchange Supplied Database Allocation for DB2

Table Name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per Page	Size of Records	Records in Primary Allocation
edibdfrx	52	24	319	8	3509
edibdms	480	240	2	2000	236
edibdmsx	28	14	319	8	1595
edicstx	4800	2400	1	4032	1198
edicstxx	32	16	143	22	858
edielog	1000	500	9	440	2270
edimrcm	100	25	101	40	2323
edimrcmx	100	25	111	29	2553
edimrpc	100	25	92	44	2116
edimrpcx	100	25	101	32	2323
edimrrt	100	25	42	96	966
edimrrtx	100	25	34	88	782
edimrpr	100	25	40	100	920
edimrprx	100	25	101	32	2323
edimrst	100	25	46	88	1058
edimrstx	100	25	39	79	897
edimrps	100	25	44	92	1012
edimrpsx	100	25	101	32	2323
edimsgs	1000	500	12	336	2970
ediownr	4	2	339	12	1017
ediprof	500	100	1	4096	120
edipsac	10	5	50	80	50
edipsacx	10	5	319	8	319
edipsad	50	10	50	81	550
edipsadx	10	5	319	8	319
edipsap	10	5	49	82	49
edipsapx	10	5	319	8	319
edipscr	10	5	21	193	21
edipscrx	10	5	188	16	188
edipsdi	10	5	6	591	6
edipsdix	10	5	778	1	778
edipsee	100	50	7	531	180
edipsie	100	50	16	253	390
edipslg	10	5	6	614	6
edipslgx	10	5	319	8	319
edipslp	10	5	46	87	46
edipslpx	10	5	385	6	385
edipsmq	100	50	24	166	600
edipsno	150	50	25	158	900
edipsnox	50	10	132	24	1452
edipsnp	10	5	15	264	15
edipsnpv	10	5	319	8	319
edipspd	10	5	1	4046	1
edipspx	10	5	319	8	319
edipsrq	50	10	17	234	187
edipsrqx	10	5	188	16	188
edipssp	10	5	30	133	30
edipsspx	10	5	319	8	319

Table F-2 (Page 3 of 4). DataInterchange Supplied Database Allocation for DB2

Table Name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per Page	Size of Records	Records in Primary Allocation
edipssy	10	5	60	67	60
edipste	100	50	14	284	350
edipssyx	10	5	319	8	19
edipstd	500	100	44	92	5412
edipstdx	100	50	319	8	319
edipstp	500	100	4	1012	492
edipstpx	100	50	188	16	4324
edipstp1	100	50	49	64	1127
edipstp2	100	50	83	39	1909
edipstp3	100	50	127	25	2921
edipstt	200	100	38	106	1824
edipsttx	100	50	75	43	1725
edipstt1	100	50	44	71	1012
edipstv	200	100	38	106	1824
edipstvx	100	50	75	43	1725
edipsue	100	50	15	268	370
edipsxe	100	50	14	275	360
ediscde	1400	480	45	90	15560
ediscdex	320	106	188	16	14664
ediscdu	2400	2400	104	39	62192
ediscdux	1908	1908	118	27	56050
ediscsg	2400	2400	61	66	36478
ediscsgx	728	728	188	16	33840
ediscst	50	15	32	126	320
ediscstx	20	5	188	16	564
ediscsu	2400	2400	101	40	60398
ediscsux	1094	1094	118	27	32096
edisctx	2400	2400	54	75	32292
edisctxx	640	640	188	16	29704
edisstk	40	10	25	159	200
edisstkx	20	5	155	20	465
edisstxx	20	5	48	65	144
editpcm	150	50	2	2000	72
editpcmx	50	20	75	43	825
editpcn	150	100	45	90	1620
editpcnx	50	10	57	56	627
editpct	300	100	2	2000	146
editpctx	30	10	81	40	486
editpdd	600	120	37	108	5476
editpddx	150	30	132	24	4752
editprt	40	10	18	217	144
editprtx	20	5	41	76	123
editpru	40	10	20	199	160
editprux	150	30	132	24	4752
editpsg	250	50	40	101	2420
editpsgx	75	20	155	20	2635
editpst	40	10	12	324	96
editpstx	20	5	47	67	141

Table F-2 (Page 4 of 4). DataInterchange Supplied Database Allocation for DB2

Table Name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per Page	Size of Records	Records in Primary Allocation
editptx	40	10	15	263	120
editptxx	20	5	188	16	564
editdid	40	10	20	198	160
editdidx	20	5	188	16	564
editdst	2400	2400	72	56	43056
editdsx	1700	1700	92	35	38916
editsth	600	120	12	323	1776
editsthx	150	30	272	10	9792
editstho	24	8	123	26	492
editsti	600	120	2	2000	296
editstix	150	30	210	14	7560
editsto	600	120	17	237	2516
editstox	150	30	272	10	9792
editsau	600	120	58	70	8485
editsaux	150	30	210	14	7560
editsauy	150	30	171	18	6156
editsev	600	120	8	501	1184
editsevx	150	30	48	66	1728
editsev1	3	3	319	8	319
editsgp	600	120	11	341	1628
editsgpx	150	30	38	80	1368
editstu	600	120	16	241	2368
editstux	150	30	32	94	1152
editstuy	150	30	272	10	9792
editslt	10	5	4074	1	4074

Supplied Database Allocation for VSAM

Table F-3 describes all database records used by DataInterchange. It also shows the number of records the allocation values will accommodate.

Formula

This formula was used to determine the number of records per control interval:

rsz = Size of a record

ciz = CI size = 4096

rci = Records per CI = $\text{FLOOR}((\text{ciz}-4)/(\text{rsz}+3))$

Note: Assume 10 control intervals per track.

Table F-3 (Page 1 of 3). DataInterchange Supplied Database Allocation for VSAM

Table Name	VSAM Primary Allocation (Tracks)	VSAM Secondary Allocation (Tracks)	Record Size	Records per Control Interval	Records in Primary Allocation
edivcstx	30	5	4032	1	300
edivscde	40	10	90	44	17600
edivscdu	100	20	64	61	61000
scduaix1	60	15	56	69	41400

Table F-3 (Page 2 of 3). DataInterchange Supplied Database Allocation for VSAM

Table Name	VSAM Primary Allocation (Tracks)	VSAM Secondary Allocation (Tracks)	Record Size	Records per Control Interval	Records in Primary Allocation
edivscsg	20	5	66	59	11800
edivscst	10	2	126	31	3100
edivscsu	50	10	68	57	28500
scsuaix1	30	15	56	69	20700
edivscsx	20	5	75	52	10400
edivtpdd	50	10	149	26	13000
tpddaix1	15	15	142	28	4200
edivtprt	10	5	294	13	1300
tpptaix1	15	15	149	26	3900
tpptaix2	15	15	142	28	4200
edivtpu	10	5	200	20	2000
edivtpsg	50	10	156	25	12500
tpsgaix1	30	15	44	87	26100
tpsgaix2	30	15	57	68	20400
edivtpst	10	5	422	9	900
tpstaix1	15	15	139	28	4200
tpstaix2	15	15	125	31	4650
tpstaix3	15	15	104	38	5700
edivtpx	10	5	320	12	1200
tpptaix1	15	15	45	85	12750
tpptaix2	15	15	53	73	109050
tpptaix3	15	15	37	102	15300
edivtdid	10	5	204	19	1900
edivtdst	40	10	56	69	27600
edivtsth	60	15	332	12	7200
tsptaix1	15	15	35	107	16050
tsptaix2	15	15	35	107	16050
edivtsti	150	75	2000	2	3000
edivtsto	10	5	380	10	1000
edivtsau	20	5	118	33	6600
tsaiaix1	15	15	45	85	12750
tsaiaix2	15	15	45	85	12750
edivtsev	60	15	660	6	3600
tseiaix1	15	15	182	22	3300
tseiaix2	15	15	145	27	4050
edivtsgp	60	15	368	11	6600
edivtstu	150	30	279	16	24000
tsaiaix1	30	15	25	146	43800
edivmrcm	15	15	40	85	14250
edivmrpc	15	15	44	78	13050
edivmrrt	15	15	96	37	6150
edivmrpr	15	15	100	35	5850
edivmrst	15	15	88	40	6600
edivmrps	15	15	92	38	6450
edivsstk	10	5	159	25	2500
sstkaix1	15	15	85	46	6900
edivbdde	10	5	296	13	1300

Table F-3 (Page 3 of 3). DataInterchange Supplied Database Allocation for VSAM

Table Name	VSAM Primary Allocation (Tracks)	VSAM Secondary Allocation (Tracks)	Record Size	Records per Control Interval	Records in Primary Allocation
bddeaix1	15	15	106	37	5550
bddeaix2	15	15	116	34	5100
edivbddf	10	5	185	21	2100
edivbddt	10	5	4049	1	100
edivbdf	10	5	113	35	3500
bdfraix1	15	15	29	127	4350
edivbdms	10	5	4004	1	100
bdmsaix1	15	15	29	127	4350
edivtpcm	10	5	468	8	800
edivtpcn	10	5	90	44	4400
tpcnaix1	15	15	56	69	10350
edivtpct	10	5	468	8	800
editpxrf	5	2	81	48	2400
docmap	15	15	1000	4	600
helps	105	10	2000	2	2100
logedi	15	15	512	7	1050
logffs	15	15	512	7	1050
msgtext	26	5	32	116	30160
profd	30	5	60	64	19200
profdef	45	15	400	10	4500
ediprof	10	5	90	44	4400
screens	20	5	2000	2	400
tabldat	200	20	100	39	78000
tbldef	15	5	252	16	2400

Space Calculation Scenario

The items below provide the information required for the example space calculation. The items are followed by tables showing the DB2 and VSAM space allocations necessary for the example. These tables are followed by blank tables which may be copied and used for your own estimates.

Note: When you are attempting to do a space estimate do not waste your time agonizing over how many fields there are per structure or how many elements there are per segment. Pick a value that seems reasonable and go with it. The majority of the DASD requirement for DataInterchange is in the Transaction Store and the majority of the Transaction Store revolves around how many transactions are processed per day and the average size of a transaction image. These values that will have the most impact on your space calculations.

- 100 trading partners
- 50 requestor IDs
- 10 transactions sets used but the entire X12V3R1 standard has been applied
- Average transaction has 30 segments defined with 10 fields per segment and all of these are mapped
- Half of the data elements mapped required a literal, accumulator or other special action (validation table, date edit, translate table, and so forth)
- 25,000 transactions per month
- Each transaction is 2000 characters in length

- 10 application data formats
- Transaction level recovery specified during translation
- One mapping for each transaction set and that mapping is used by the 100 trading partners. Half of these will be used for receive processing and the other half will be used for send processing.
- 30 segments mapped per transaction with 10 fields per segment.
- 30 structures per application data format with 10 fields per structure
- The average number of transactions per envelope is 8
- Each interchange contains 1 functional group
- Retention of transactions for 1 month with a transaction remove run once a month
- Retention of management reporting statistics for 2 months with UPDATE STATISTICS run one a week

Formula

This formula was used to determine the DB2 primary allocation amount (PRIQTY):

rpp = records per page, as shown in Table F-2 on page F-24
 psz = DB2 page size = 4096
 alu = DB2 allocation unit = 1024 bytes
 rnb = total number of records wanted
 pri = Primary allocation =
 $((\text{FLOOR}(\text{rnb}/\text{rpp}) + 2) * \text{psz}) / \text{alu}$

DataInterchange/DB2 Database Allocation Required for Space Calculation Scenario

Table F-4 describes the allocation required given the set of assumptions as explained in "Space Calculation Scenario" on page F-29.

Table F-4 (Page 1 of 5). DataInterchange/DB2 Database Allocation Required for Space Calculation Scenario

Table Name	Number of Records Required	DB2 Primary Allocation	Comments
edicstx	50	208	<ul style="list-style-type: none"> • CSTXNUM = TDIDNUM + (4 * TPTXNUM) • CSTXNUM = 10 + (4 * 10) • CSTXNUM = 50
edicstxx	50	12	<ul style="list-style-type: none"> • CSTXXNUM = CSTXNUM • CSTXXNUM = 50
ediscde	901	96	<ul style="list-style-type: none"> • SCDENUM = EDISCDE entries in X12V3R1
ediscdex	901	28	<ul style="list-style-type: none"> • SCDEXNUM = SCDENUM • SCDEXNUM = 901
ediscdx	3707	184	<ul style="list-style-type: none"> • SCDUNUM = EDISCDU entries in X12V3R1
ediscdx	3707	136	<ul style="list-style-type: none"> • SCDUXNUM = SCDUNUM • SCDUXNUM = 3707
edismsg	520	48	<ul style="list-style-type: none"> • SCSGNUM = EDISMSG entries in X12V3R1
edismsgx	520	20	<ul style="list-style-type: none"> • SCSGXNUM = SCSGNUM • SCSGXNUM = 520
edismsg	1	12	<ul style="list-style-type: none"> • SCSTNUM = 1 standard loaded

Table F-4 (Page 2 of 5). DataInterchange/DB2 Database Allocation Required for Space Calculation Scenario

Table Name	Number of Records Required	DB2 Primary Allocation	Comments
ediscstx	1	8	<ul style="list-style-type: none"> SCSTXNUM = SCSTNUM SCSTXNUM = 1
ediscsu	1622	88	<ul style="list-style-type: none"> SCSUNUM = EDISCSU entries in X12V3R1
ediscsux	1622	64	<ul style="list-style-type: none"> SCSUXNUM = SCSUNUM SCSUXNUM = 1622
edisctx	39	12	<ul style="list-style-type: none"> SCTXNUM = EDISCTX entries in X12V3R1
edisctxx	39	12	<ul style="list-style-type: none"> SCTXXNUM = SCTXNUM SCTXXNUM = 39
editpdd	3000	352	<ul style="list-style-type: none"> SSMAP = Average number of MAPPED segments = 30 ELSS = Average number elements per segment = 10 TPDDNUM = TPTXNUM * SSMAP * ELSS TPDDNUM = 10 * 30 * 10 = 3000
editpddx	3000	100	<ul style="list-style-type: none"> TPDDXNUM = TPDDNUM TPDDXNUM = 3000
editprt	500	96	<ul style="list-style-type: none"> TPTXNUM = Total number of mappings = 10 PRECV = Percentage of receive mappings = 50% TPPROFNUM = Total number of trading partners = 100 TPRECV = Percentage trading partners receiving = 100% TPRTNUM = (TPTXNUM * PRECV) * (TPPROFNUM * TPRECV) TPRTNUM = (10 * .5) * (100 * 1) TPRTNUM = 500
editprtx	500	60	<ul style="list-style-type: none"> TPDDXNUM = TPDDNUM TPDDXNUM = 3000
editpru	1500	324	<ul style="list-style-type: none"> TPDDNUM = Total number of fields = 3000 SPMAP = Percentage of fields that are mapped with either literal values, accumulators, edits, validation/translation tables, sub-string/concatenate, user exits = 50% TPRUNUM = TPDDNUM * SPMAP TPRUNUM = 3000 * .5 = 1500
editprux	1500	56	<ul style="list-style-type: none"> TPRUXNUM = TPRUNUM TPRUXNUM = 1500
editpsg	300	44	<ul style="list-style-type: none"> SSSTD = Average number of segments in a transaction = 30 SSRMAP = Average number repeated mappings = 0 TPSGNUM = TPTXNUM * (SSSTD + SSRMAP) TPSGNUM = 10 * (30 + 0) = 300
editpsgx	300	16	<ul style="list-style-type: none"> TPSGXNUM = TPSGNUM TPSGXNUM = 300

Table F-4 (Page 3 of 5). DataInterchange/DB2 Database Allocation Required for Space Calculation Scenario

Table Name	Number of Records Required	DB2 Primary Allocation	Comments
editpst	500	152	<ul style="list-style-type: none"> • TPTXNUM = Total number of mappings = 10 • PSEND = Percentage of mappings for send = 50% • TPPROFNUM = Total number of trading partners = 100 • TPSEND = Percentage of trading partners you send to = 100% • TPSTNUM = (TPTXNUM * PSEND) * (TPPROFNUM * TPSEND) • TPSTNUM = (10 * .5) * (100 * 1) • TPSTNUM = 500
editpstx	500	52	<ul style="list-style-type: none"> • TPSTXNUM = TPSTNUM • TPSTXNUM = 500
editptx	10	12	<ul style="list-style-type: none"> • TPTXNUM = Number of mappings = 10
editptxx	10	12	<ul style="list-style-type: none"> • TPTXXNUM = TPTXNUM • TPTXXNUM = 10
editdid	10	12	<ul style="list-style-type: none"> • TDIDNUM = Number of application data formats = 10
editdidx	10	12	<ul style="list-style-type: none"> • TDIDXNUM = TPIDNUM • TDIDXNUM = 10
editdst	3600	240	<ul style="list-style-type: none"> • TDIDNUM = Number of application data formats = 10 • STFMT = Average number of structures per data format = 30 • FLSTR = Average number of fields per structure = 10 • TDSTNUM = ((STFMT * 2) + (STFMT * FLSTR)) * TDIDNUM • TDSTNUM = ((30 * 2) + (30 * 10)) * 10 • TDSTNUM = 3600
editdstx	3600	168	<ul style="list-style-type: none"> • TDSTXNUM = TPSTNUM • TDSTXNUM = 3600
editsth	50000	16676	<ul style="list-style-type: none"> • TRXPDAY = Number of transactions per day = 25000/30 • TSTHNUM = TRXPDAY * TSLIFE • TSTHNUM = 833.3 * 60 = 50000 <p>Note:</p> <ul style="list-style-type: none"> • TRXLIFE = days before purge = 30 • PRGSPAN = frequency of remove utility = 30 • TSLIFE = TRXLIFE + PGRSPAN • TSLIFE = 30 + 30 = 60
editsthx	50000	744	<ul style="list-style-type: none"> • TSTHXNUM = TSTHNUM • TSTHXNUM = 50000
editsti	50000	100008	<ul style="list-style-type: none"> • TSTINUM = TSTHNUM = 50000
editstix	50000	964	<ul style="list-style-type: none"> • TSTIXNUM = TSTINUM • TSTIXNUM = 50000
editsto	0	8	<ul style="list-style-type: none"> • TSTONUM = 0 (overrides not being used)
editstox	0	8	<ul style="list-style-type: none"> • TSTOXNUM = TSTONUM • TSTOXNUM = 0
editsau	50000	3856	<ul style="list-style-type: none"> • TSAUNUM = TSTHNUM = 50000

Table F-4 (Page 4 of 5). DataInterchange/DB2 Database Allocation Required for Space Calculation Scenario

Table Name	Number of Records Required	DB2 Primary Allocation	Comments
editsaux	50000	964	<ul style="list-style-type: none"> TSAUXNUM = TSAUNUM TSAUXNUM = 50000
editsauy	50000	1188	<ul style="list-style-type: none"> TSAUYNUM = TSAUNUM TSAUYNUM = 50000
editsev	6250	3136	<ul style="list-style-type: none"> TRXPENV = Average number of transactions per envelope = 8 TSEVNUM = TSTHNUM / TRXPENV TSEVNUM = 50000 / 8 = 6250
editsevx	6250	532	<ul style="list-style-type: none"> TSEVXNUM = TSEVNUM TSEVXNUM = 6250
editsgp	6250	2284	<ul style="list-style-type: none"> GRPPENV = Average groups per envelope = 1 TSGPNUM = TSEVNUM * GRPPENV TSGPNUM = 6250 * 1 = 6250
editsgpx	6250	668	<ul style="list-style-type: none"> TSGPXNUM = TSGPNUM TSGPXNUM = 6250
editstu	50000	12508	<ul style="list-style-type: none"> TSTHNUM = TSTHNUM = 50000
editstux	50000	6260	<ul style="list-style-type: none"> TSTUXNUM = TSTUNUM TSTUXNUM = 50000
editstuy	50000	744	<ul style="list-style-type: none"> TSTUYNUM = TSTUNUM TSTUYNUM = 50000
edimrcm	3100	178	<ul style="list-style-type: none"> RIDNUM = Number of Requestor ID's (RID) = 50 CUMENTS = Number of cumulative entries = RIDNUM * 2 CUMENTS = 50 * 2 = 100 PRIDINT = Daily percentage of RID's having an interchange = 50% MEASPDAY = (RIDNUM * 2) * PRIDINT MEASPDAY = (50 * 2) * .5 = 50 MRCMNUM = (MEASPDAY * MEASLIFE) + CUMENTS MRCMNUM = (50 * 60) + 100 = 3100 <p>Note:</p> <ul style="list-style-type: none"> MEASLIFE = 60 PENDLIFE = 7 RIDNUM = 50 TPRTNUM = 500 TPSTNUM = 500
edimrcmx	3100	122	<ul style="list-style-type: none"> MRCMXNUM = MRCMNUM MRCMXNUM = 3100

Table F-4 (Page 5 of 5). DataInterchange/DB2 Database Allocation Required for Space Calculation Scenario

Table Name	Number of Records Required	DB2 Primary Allocation	Comments
edimrpc	7000	435	Transaction level recovery formula: <ul style="list-style-type: none"> RIDNUM = Number of Requestor ID's (RID) = 50 SENDNUM = Avg # of send interchanges per RID per day = 10 RECVNUM = Avg # of receive interchanges per RID per day = 10 PENDPDAY = (SENDNUM + RECVNUM) * RIDNUM PENDPDAY = (10 + 10) * 50 = 1000 MRPCNUM = PENDPDAY * Pendlife MRPCNUM = 1000 * 7 = 7000
edimrpcx	7000	302	<ul style="list-style-type: none"> MRPCXNUM = MRPCNUM MRPCXNUM = 7000
edimrrt	15500	1926	<ul style="list-style-type: none"> PACTIVE = Daily percentage of receive usages having activity = 50% MEASPDAY = TPRTNUM * PACTIVE MEASPDAY = 500 * .5 = 250 MRRTNUM = (MEASPDAY * MEASLIFE) + TPRTNUM MRRTNUM = (250 * 60) + 500 = 15500
edimrrtx	15500	1983	<ul style="list-style-type: none"> MRRTXNUM = MRRTNUM MRRTXNUM = 15500
edimrpr	2919	385	Transaction level recovery formula: <ul style="list-style-type: none"> TRXPDAY = Number of transactions per day = 25000/30 = 834 PRECV = Percentage of transactions that are receive transactions = 50% MRPRNUM = TRXPDAY * PRECV * Pendlife MRPRNUM = 834 * .5 * 7 = 2919
edimprx	2919	125	<ul style="list-style-type: none"> MRPRXNUM = MRPRNUM MRPRXNUM = 2919
edimrst	15500	1774	<ul style="list-style-type: none"> PACTIVE = Daily percentage of send usages having activity = 50% MEASPDAY = TPSTNUM * PACTIVE MEASPDAY = 500 * .5 = 250 MRSTNUM = (MEASPDAY * MEASLIFE) + TPSTNUM MRSTNUM = (250 * 60) + 500 = 15500
edimrstx	15500	1728	<ul style="list-style-type: none"> MRSTXNUM = MRSTNUM MRSTXNUM = 15500
edimrps	2919	353	Transaction level recovery formula: <ul style="list-style-type: none"> TRXPDAY = Number of transactions per day = 25000/30 = 834 PRECV = Percentage of transactions that are receive transactions = 50% MRPRNUM = TRXPDAY * PRECV * Pendlife MRPRNUM = 834 * .5 * 7 = 2919
edimrpsx	2919	126	<ul style="list-style-type: none"> MRPSXNUM = MRPSNUM MRPSXNUM = 2919

DataInterchange/VSAM Database Allocation Required for Space Calculation Scenario

Table F-5 describes the allocation required given the set of assumptions as explained in “Space Calculation Scenario” on page F-29.

Formula

The VSAM primary allocation amount (TRKS) is determined with:

rci = records per control interval (see Table F-3 on page F-27)

ppt = Number of control intervals per TRACK = 10

(3380=10, 3330=3, 3340=2, 3350=4, 3375=8)

rnb = total number of records wanted

pri = Primary allocation =

$\text{CEILING}(((\text{FLOOR}(\text{rnb}/\text{rci}))/\text{ppt}))$

Note: A minimum value of 2 tracks are allocated.

Table F-5 (Page 1 of 2). DataInterchange/VSAM Database Allocation Required for Space Calculation Scenario

Table Name	Number of Records Required	VSAM Primary Allocation (TRACKS)
edivcstx	50	5
edivscde	901	3
edivscdu	3707	6
scduaix1	3707	6
edivscsg	520	2
edivscst	1	2
edivscsu	1622	3
scsuaix1	1622	3
edivscstx	39	2
edivtpdd	3000	12
tpddaix1	3000	11
edivtprt	500	4
tpртаix1	500	2
tpртаix2	500	2
edivtpu	1500	8
edivtpsg	300	2
tpsgaix1	300	2
tpsgaix2	300	2
edivtpst	500	6
tpstaix1	500	2
tpstaix2	500	2
tpstaix3	500	2
edivtpx	10	2
tptxaix1	10	2

Table F-5 (Page 2 of 2). DataInterchange/VSAM Database Allocation Required for Space Calculation Scenario

Table Name	Number of Records Required	VSAM Primary Allocation (TRACKS)
tptxaix2	10	2
tptxaix3	10	2
edivtdid	10	2
edivtdst	3600	6
edivtsth	50000	455
tsthaix1	50000	47
tsthaix2	50000	47
edivtsti	50000	2500
edivtsto	0	2
edivtsau	50000	152
tsauaix1	50000	59
tsauaix2	50000	59
edivtsev	6250	100
tsevaix1	6250	28
tsevaix2	6250	23
edivtsgp	6250	55
edivtstu	50000	313
tstuaix1	50000	35
edivmrcm	3100	4
edivmrpc	7000	9
edivmrtr	15500	38
edivmrpr	2919	8
edivmrst	15500	36
edivmrps	2919	7

Note: These tables are not fixed in size however, in this example the tables did not contribute sufficient data to cause an increase over the size of the tables as they are shipped with the product:

profdat	30
tbldef	15
tbldat	200

Note: These tables are basically fixed in size:

profdef	3
edimsg	25
ediscrn	20
edimsg	105

Worksheets

These worksheets will provide guidance for calculating your space estimates using the DB/2 and VSAM database records supplied.

DataInterchange/DB2 Database Allocation Worksheet

Table F-6 (Page 1 of 4). DataInterchange/DB2 Database Allocation Worksheet

Table Name	Number of Records Required	DB2 Primary Allocation
edibdde		
edibddex		
edibddf		
edibddfx		
edibddt		
edibddtx		
edibdfr		
edibdfrx		
edibdms		
edibdmsx		
edicstx		
edicstxx		
ediscde		
ediscdex		
ediscdu		
ediscdux		
ediscsg		
ediscsgx		
ediscst		
ediscstx		
ediscsu		
ediscsux		
ediscst		
ediscstx		
editpdd		
editpddx		
editprt		
editprtx		
editpru		
editprux		
editpsg		

Table F-6 (Page 2 of 4). DataInterchange/DB2 Database Allocation Worksheet

Table Name	Number of Records Required	DB2 Primary Allocation
editpsgx		
editpst		
editpstx		
editptx		
editptxx		
editdid		
editdidx		
editdst		
editdstx		
editsth		
editsthx		
editsti		
editstix		
editsto		
editstox		
editsau		
editsaux		
editsaui		
editsev		
editsevx		
editsgp		
editsgpx		
editstu		
editstux		
editstuy		
edimrcm		
edimrcmx		
edimrpc		
edimrpcx		
edimrrt		
edimrrtx		
edimrpr		
edimrprx		
edimrst		
edimrstx		
edimrps		
edimrpsx		

Table F-6 (Page 3 of 4). DataInterchange/DB2 Database Allocation Worksheet

Table Name	Number of Records Required	DB2 Primary Allocation
editpcm		
editpcmx		
editpcn		
editpcnx		
editpct		
editpctx		
ediprof		
editslt		
ediownr		
edipsac		
edipsacx		
edipsad		
edipsadx		
edipsap		
edipsapx		
edipscr		
edipscrx		
edipsdi		
edipsdix		
edipsep		
edipsepx		
edipslg		
edipslgx		
edipslp		
edipslpx		
edipsno		
edipsnox		
edipsnp		
edipsnpv		
edipspd		
edipspx		
edipsrq		
edipsrqx		
edipssp		
edipsspx		
edipssy		
edipssyx		

Table F-6 (Page 4 of 4). DataInterchange/DB2 Database Allocation Worksheet

Table Name	Number of Records Required	DB2 Primary Allocation
edipstd		
edipstdx		
edipstp		
edipstpx		
edipstp1		
edipstp2		
edipstp3		
edipstt		
edipsttx		
edipstt1		
edipstv		
edipstvx		
edisstk		
edisstkx		
edisstxx		

DataInterchange/VSAM Database Allocation Worksheet

Table F-7 (Page 1 of 3). DataInterchange/VSAM Database Allocation Worksheet

Table Name	Number of Records Required	DB2 Primary Allocation
edibdde		
edibddf		
edibddt		
edibdfr		
edibdms		
edivcstx		
edivscde		
edivscdu		
scduaix1		
edivscsg		
edivscst		
edivscsu		
scsuaix1		
edivscst		
edivtpdd		
tpddaix1		

Table F-7 (Page 2 of 3). DataInterchange/VSAM Database Allocation Worksheet

Table Name	Number of Records Required	DB2 Primary Allocation
edivtprt		
tpртаix1		
tpртаix2		
edivtpru		
edivtpsg		
tpsgaix1		
tpsgaix2		
edivtpst		
tpstaix1		
tpstaix2		
tpstaix3		
edivtpt		
tptxaix1		
tptxaix2		
tptxaix3		
edivtdid		
edivtdst		
edivtsth		
tsthaix1		
tsthaix2		
edivtsti		
edivtsto		
edivtsau		
tsauaix1		
tsauaix2		
edivtsev		
tsevaix1		
tsevaix2		
edivtsgp		
edivtstu		
tstuaix1		
edivmrcm		
edivmrpc		
edivmrrt		
edivmrpr		
edivmrst		
edivmrps		

Table F-7 (Page 3 of 3). DataInterchange/VSAM Database Allocation Worksheet

Table Name	Number of Records Required	DB2 Primary Allocation
editpcm		
editpcn		
tpcnaix1		
editpct		
edisstk		
sstkaix1		

Appendix G. Performance Considerations

This Appendix provides performance information on DataInterchange for MVS and DataInterchange for CICS to help you plan your system requirements.

DataInterchange Optimization

The following considerations and tuning techniques are highly recommended to achieve optimum DataInterchange for MVS performance:

- General optimization techniques for either batch or CICS
- DataInterchange for MVS considerations for batch
- DataInterchange for CICS considerations for CICS only
- File maintenance techniques
- Transaction Store query techniques
- VS COBOL II Field Exit considerations

General Optimization Techniques

- High-volume EDI customers using a DB2 repository should define multiple DASD volumes to DB2 storage group EDISTG01 to facilitate spreading DB2 data sets across multiple packs. Transaction Store DB2 tables EDITSAU, EDITSTH, EDITSTI, EDITSTU, EDITSEV, EDITSGP are heavily referenced and should be defined with large DASD allocations. Monitor the status of DB2 Table data sets on a regular basis (at least twice monthly, more often if possible). Use LISTCAT and DB2 Utilities such as CHECK, RUNSTAT, and REORG to monitor and tune the DB2 table data sets. Add volumes to DB2 storage groups, increase table space DASD, and reorganize DB2 Table data sets when CI and especially CA splits are encountered.
- High-volume EDI customers using a VSAM repository should place data, index, and alternate indexes on separate DASD packs to reduce data set contention. VSAM files EDIVTSAU, EDIVTSTH, EDIVTSTI, EDIVTSTU, EDIVTSEV, EDIVTSGP are heavily referenced and should be defined with large DASD allocations. Monitor the status of VSAM data sets on a regular basis (at least twice monthly, more often if possible) using LISTCAT function. Any data set having over 50 CI splits or 10 CA splits should be reorganized. Control CI and CA splits by redefining problem data sets with FREESPACE(20 10) or more as required. When in doubt, reorganize the data set. Reorganizations must be done in DISP=OLD mode to ensure that no one is accessing the data set while it is being reorganized.
- Spread VSAM Data, Index, and alternate indexes across multiple packs to reduce data set contention. The user profile data set PROFDAT is heavily referenced and should be defined on a low-use disk away from other DI data sets.
- Define DASD space for all VSAM data sets (except PROFDEF) specified in CYLINDERS instead of TRACKS. This results in a CYLINDER CA size being specified that results in fewer INDEX records. Fewer index records improve I/O performance.
- Use cache DASD. If cache DASD is used, VSAM data sets should NOT be defined with IMBED or REPLICATE functions.
- Define Master Catalog for the high-level data set name qualifier on a pack separate from all other MVS data sets.

- Only log application data and standard data images during the testing period. The TPPROF profile controls logging of interchange images. The trading partner transaction usage controls the logging of application images. Additionally, do not log profile access attempts if not necessary for your environment. The Log action on the Profile menu (panel ad01) controls this.
- Only log events during the testing period. The ACTLOGS profile, member EDIFFS, log flag controls whether normal events are logged. Error conditions and events that change a transaction's status are recorded even if logging is turned off.
- Minimize translator error messages by making use of the &DIERRFILTER keyword.
- Use commands that combine several functions in one; for example, PERFORM TRANSLATE AND ENVELOPE, rather than PERFORM TRANSLATE TO STANDARD and PERFORM ENVELOPE.
- If concurrency is an issue and multiple DIs involve the same trading partner, they can single thread through the trading partner profile. This is true for outbound translation and when generating functional acknowledgments. Transaction level recovery, rather than envelope level recovery, may be advisable here to help concurrency and improve throughput. This can be accomplished using the RECOVERY keyword.
- Specify the DataInterchange Application Data Structures as Passed Separately = N, whenever possible. Passed Separately = N requires less interactions, less overhead, and less I/O than does Passed Separately = Y.
- Although conditional logic mapping is a valuable feature — if used excessively, it degrades translation performance. The special keywords &E, &IF, &ERR, and &ASSERT are expensive at run time. Avoid excessive use of these in highly repetitive loop mappings.
- Generate control strings for your envelope standards to reduce I/O associated with retrieving envelope definitions.
- If you are currently using a mapping validation level of 2, consider the use of special keyword literals &DIVALTYPE and &DIVALLEVEL as an alternative. You can control validation of only particular, important elements using these keywords.

Note: PROFDEF, PROFDAT, TABLDEF, TABLDAT, and EDITPXRF are used only in the VSAM version of DataInterchange.

DataInterchange for MVS Considerations

- Define FFSWORK data set in the DataInterchange for MVS utility JCL as virtual I/O to drastically reduce I/O EXCPs relating to the work data set.

```
//FFSWORK DD DSN=&FFSWORK,DISP=(NEW,DELETE,DELETE),
//          DCB=(RECFM=V,BLKSIZE=32760),
//          UNIT=VIO,SPACE=(TRK,(3,3))
```
- Use a record length greater than 512 with a large block size for envelope files to reduce I/O during envelope and de-envelope operations.
- Define the transaction input data sets (APDATA01, APDATA02, and so forth) on DASD separate from other DI for MVS data sets to greatly reduce data set contention.
- If translating large files, the following formula can be used to determine virtual storage requirements. (The same formula pertains to both inbound and outbound translation.)

virtual storage required	=	largest application trx image	+	EDI Interchange length	+	4Meg overhead
--------------------------------	---	-------------------------------------	---	------------------------------	---	---------------

Largest application trx image: length of the largest transaction in application format (Input to Translate and Envelope, or output from Deenvelope and Translate).

If multiple EDI transactions are being processed, only consider the largest transaction in application format because DataInterchange frees the application image as each transaction is processed. In most cases, the application image is many times larger than the EDI counterpart because application fields are fixed length. Also, application data format structures defined as NOT passed separately tend to drastically increase this length due to exorbitant padding.

EDI Interchange image: length of entire EDI interchange, including header and trailer segments.

EDIFACT – UNB through UNZ

ANSI X12 – ISA through IEA

UCS – BG through EG

UNTDI – STX through SCH

ICS – ICS through ICE

4Meg overhead: amount of storage required for loading programs, I/O buffering, etc. DataInterchange obtains most storage above the 16Meg line.

Note: The MVS default limit for above-the-line storage is 32 MB. Specifying a JCL REGION of greater than 16 MB and less than 32 MB only reduces this default limit. Only change the JCL REGION specification to accommodate your estimation when it exceeds 32 MB, or use REGION=8 MB to remove the limit. Please refer to MVS JCL documentation for details regarding the REGION parameter.

DataInterchange for CICS Considerations

- For CICS/ESA installations, the DataInterchange Persistent Environment can be used. The Persistent Environment is designed to minimize read operations involved in translating data.
- Make sure transaction EDIX is enabled. This is done with CEMT I TRAN(EDIX).
- Make the VSAM share options (2,3) for the EDIMSG file. This is done using the IDCAMS ALTER or the TSO ALTER command.
- To reduce actual I/O on VSAM ESDS data sets, use multiple buffers. This is done by adding AMORG=('BUFND=nn') to the corresponding DD statements in the CICS startup JCL.
- The DataInterchange Utility single threads through print files, exception files, report files, tracking files, and query files. The names of these files can be specified in the Utility Control Information block passed by the application to the Utility. If concurrency is an issue, choose unique names for these files.
- If concurrency is an issue, spread functional acknowledgments to separate, temporary storage queues by making use of the FUNACKFILE keyword.
- For CICS/ESA installations, have EDIMSG defined as a CICS data table, if possible. Use the CICS RDO facility to do this. However, be aware that changes made in batch or TSO are not reflected immediately in CICS data tables.
- If you run many small, concurrent DataInterchange translations involving numerous trading partners, you can develop HOT-DI applications to drive DataInterchange.
- If translating large files in CICS, use the same formula as mentioned above in "DataInterchange for MVS Considerations" on page G-2. In CICS, most storage will be obtained from extended DSA (this is a CICS 4.1 EDSALIN SIT parameter). You must also consider other CICS activity, such as concurrent DataInterchange for CICS tasks. Concurrent tasks each require 3 MB of additional overhead.
- Maximize throughput by using concurrent DataInterchange for CICS tasks. There should not be more than six of these at any one time. Use the CICS RDO TRANClass command to limit concurrent activity.

File Maintenance Techniques

Delete or archive unused and outdated data from DI for MVS data sets on a regular basis. Small data sets result in better performance.

- Do not track unneeded, outdated transactions in the Transaction Store database. Performance is degraded when a database query range includes these unimportant records. Execute `PERFORM PURGE` and `PERFORM REMOVE` regularly to delete unneeded records. Also, use an appropriate purge interval. For more information on the file content and space calculations, see Appendix F, "Space Calculation Examples" on page F-1.
- A `WHERE` clause can be specified for `PERFORM REMOVE`. If used properly, this reduces the scope of the database query to transactions known to be in a `PURGE - DATE EXPIRED` status. Note that the remove function requires a database query to determine the remove eligibility of expired transactions. Narrowing the query to only eligible transactions improves performance; for example, if the default purge interval of 30 days is used, it is obvious that transactions added in the last 30 days are not eligible for removal. The command could be: `PERFORM REMOVE TRANSACTIONS WHERE HANDLE(*-999) TO(*-30)`
- If the Transaction Store information is not used or needed for your environment, turn off the writing of these records through options in the `APPDEFS` profile. Be aware that if `PERFORM ENVELOPE` or `TRANSLATE TO APPLICATION` is used, it is essential that Transaction Store records be written as this ensures their retrieval.
- Archive event log entries regularly. For more information on the archive utility, see Appendix E, "Using Sample JCL" on page E-1. Or, if records therein are not needed, use the sample install JCL to `DELETE`, `DEFINE`, and `LOAD` the log file. Sample JCL is contained in target library `SEDIINS1`. Member `EDIJVSDF` contains JCL to delete and define the log. Member `EDIJVSLD` contains JCL to load/prime the VSAM file. Only run the applicable job step for the event log in question.

Transaction Store Query Techniques

The `WHERE` option specified during a database query plays a significant role in performance. A database query takes place with various `PERFORM` statements, the most common of which are `ENVELOPE`, `TRANSLATE TO APPLICATION`, `PRINT`, `PURGE`, and `REMOVE`. These are Transaction Store utility functions that require a search and retrieval of information contained therein.

The following techniques apply to all `PERFORM` commands that require a database selection/query. These same techniques can be utilized with the interactive Transaction Store facility. The following list contains criteria and the order in which the criteria are checked. (All fields at the same indentation level are checked in the order listed.)

Transaction Handle
Batch ID
Direction

All information in Transaction Handle Record(s)
Table: EDITSTH

Application Control field
Direction
Standard Transaction ID
Transaction Handle
Date added to store
Time added to store
Trading Partner Nickname
Internal Trading Partner ID
Network ID
Envelope Type
Earliest purge date
Earliest envelope date

All information in Application Record(s)
Table: EDITSAU

Application ID
Application Data Format ID
Batch ID
Translation Error Level
Delivered date
Delivered time

All information in Transaction Usage Record(s)
Table: EDITSTU

Interchange Control Number
Group Control Number
Transaction Control Number
Interchange Receiver ID

All information in Group Usage Record(s)
Table: EDITSGP

Application Sender ID
Application Receiver ID
Functional Acknowledgment Pending

All information in Envelope Usage Record(s)
Table: EDITSEV

Interchange Sender ID
Send date
Send time
Network Status
Network Acknowledgment Pending
Envelope date
Envelope time

It is not necessary to understand all the relationships between the files/records to utilize the preceding information. However, it is important to know that generally there is one record for each EDI transaction in each of the previous tables. With this in mind, correctly applying the following two rules can significantly improve query performance:

1. Directly limit the query to a specific range of transactions using a handle range, batch ID, and/or direction. Considering that the primary file is the transaction handle file (EDITSTH) and that the primary key to this file is the transaction handle, specifying a handle range reduces the number of records accessed. When specifying a handle range, the selection program is given a partial key, thereby excluding inapplicable records.

Note: When executing the DI Utility it is possible to use (*-n) within the from and to values of the HANDLE range. An * represents the current system date and n represents the number of days before.

Handle, Direction, and Batch ID are the only fields that directly narrow the search window. Further criteria specifications, such as Trading Partner Nickname or Standard Transaction ID, require a scan of all records directly obtained using Handle, Direction, or Batch ID.

2. Limit the query using the transaction handle record (EDITSTH) if applicable. The criteria within the transaction handle record are the next best way to limit the selection. These values are not keys, but are better than the transaction usage record, group usage record, and envelope usage record. For example, if you translate and send many transactions to a specific trading partner and want to retrieve these specific transactions, use the trading partner nickname rather than the interchange receiver ID. The trading partner nickname is easier to retrieve than the interchange receiver ID and, therefore, is more efficient for queries.

VS COBOL II Field Exit Considerations

If DataInterchange for MVS translation performance degrades after adding a VS COBOL II field exit, it is probably because of a tremendous increase in the EXCP count. This degradation can probably be attributed to the LIBKEEP run time parameter. By default, the run time parm LIBKEEP is N. For every program or sub-program execution, the run time environment is reinitialized. If a field exit is called by DI thousands of times, the EXCPs go up and performance goes down. The execution option of LIBKEEP can be changed to 'Y' in macro IGZEOPD; thus, keeping the run time environment in memory after a GOBACK from the field exit. However, this is a global change and affects all COBOL applications using this run time lib. Consider the implications of this:

1. The system programmer needs to balance EXCPs with region size constraints. The more sophisticated a COBOL program is, the more it takes to initialize the run time environment. In the case of DI field exits, this probably is minimal because these exits typically only work with storage. However, keeping the environment in storage may effect the region sizes of other larger COBOL applications.
2. Consider the environmental issue of both COBOL OS/VS and VSCOBOL II. Because LIBKEEP may keep the COBOL II run time in memory, COBOL OS/VS programs may access this rather than strictly COBOL OS/VS. The only impact is that this forces NO END JOB in a COBOL OS/VS program.

The problem of the IGZEOPD macro being global can be bypassed by using a separate CSI for a copy of the VS COBOL II run time library. A special run time library can be installed where LIBKEEP=Y. This library can then be STEPLIBed in the DataInterchange for MVS job, so it is not accessed by the rest of the MVS system. This eliminates the implications on other COBOL applications.

The run-time parameters are documented in the *VS COBOL II Application Programming Guide*. The options of interest are LIBKEEP and RTEREUS. They are related, with the exception that RTEREUS provides reusability for concurrent jobs. Both are run time options and not compiler or link-edit options.

Glossary

A

AAR. Association of American Railroads. Represents the railroad industry in areas such as standards, public relations, and advertising.

acknowledgment. See *functional acknowledgment*, *network acknowledgment*.

action bar. The area at the top of a panel that contains choices currently available in the application that is running. Compare to the *function key area*, which contains actions common to all programs.

ADF. See application data format.

ANSI. American National Standards Institute.

ANSI ASC X12. ANSI Accredited Standards Committee X12, which develops and maintains generic standards for business transactions for EDI.

application. A program that processes business information. An application that requests services from DataInterchange is an *enabled application*.

application data. The actual data in a transaction.

application data format. A description of the application data for a particular transaction. An application data format is composed of data structures and fields.

B

base structure. The data structure that contains all the data structures and data fields that define the application data for a single transaction.

binary format (BIN). Representation of a decimal value in which each field must be 2 or 4 bytes long. The sign (+ or –) is in the far left bit of the field, and the number value is in the remaining bits of the field. Positive numbers have a 0 in the sign bit. Negative numbers have a 1 in the sign bit and are in twos complement form.

C

CICS. Customer Information Control System.

CLIST. Command list.

command line. The line at the bottom of the panel that provides an alternate way of requesting services, rather than using the *Action* column of the panel body.

Command list (CLIST). A list of commands and statements designed to perform a specific function for the user.

composite data element. In EDIFACT standards, a group of related subelements, such as the elements that make up a name and address.

control string. A compiled map; a machine-readable representation of a map generation that optimizes performance during translation.

control structure. The beginning and ending segments (header and trailer) of standard enveloped transmissions.

Customer Information Control System (CICS). An IBM-licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs.

customize. To alter to suit the needs of a company, such as removing from an EDI standard the segments and data elements that the company does not use.

D

DASD. Direct access storage device.

data dictionary. A file containing the definitions of all the data elements of an EDI standard.

data element. A single item of data in a standard, such as a purchase order number. Corresponds to a data field in a data format.

data element delimiter. A character, such as an asterisk (*), that follows the segment identifier and separates each data element in a segment. See also *element separator* and *segment ID separator*.

data field. A single item of data in a data format, such as a purchase order number. Corresponds to a data element in a standard.

data format. See *application data format*.

data structure. A group of related data fields in a data format, such as the fields making up the line item of an invoice. Corresponds to a segment in a standard.

| **DB2.** Database 2, an IBM relational database management system.

ddname. Data definition name.

decimal notation. The character that represents a decimal point in an envelope standard.

| **DI Client.** DataInterchange Client, the Windows-based, client/server interface for DataInterchange.

direct access storage device (DASD). A device in which access time is effectively independent of the location of the data.

distribution libraries. Supplied partitioned data sets on tape containing one or more components used to transfer data to a new system.

distribution tape. A magnetic tape that contains the distribution libraries for installing a new system.

domain. The data structure or group of data structures in a data format to and from which you should restrict the mapping of EDI repeating segments and loops.

E

EDI. Electronic data interchange.

EDIA. Electronic Data Interchange Association.

EDI administrator. The person responsible for setting up and maintaining DataInterchange.

| **EDIFACT.** Electronic Data Interchange for Administration Commerce and Support.

electronic data interchange (EDI). A method of transmitting business information over a network, between business associates who agree to follow approved national or industry standards in translating and exchanging information.

electronic transmission. The means by which information is transferred between parties, such as over a public network.

element. See *data element*.

element separator. A character that separates the data elements in a segment. See also *data element delimiter*.

encryption. The encoding and scrambling of data. Data is encrypted by the sender and decrypted by the

receiver using a predetermined program and unique electronic key.

event. An occurrence that is important to a user's computer task, such as a software error, sending a transaction, or acknowledging a message.

F

field. See *data field*.

functional acknowledgment. An electronic acknowledgment returned to the sender to indicate acceptance or rejection of EDI transactions.

functional group. One or more transaction sets of a similar type transmitted from the same location, enclosed by functional group header and trailer segments.

function key. A key that causes a specified sequence of operations to be performed when it is pressed. Generally used to refer to keys labelled F_n , where n is a number from 1 to 24.

function key area. Two lines at the bottom of the panel that list the active function keys for the panel.

G

GDDM. Graphical data display manager.

graphical data display manager (GDDM). An IBM licensed program that allows pictures to be defined and displayed on your monitor.

H

header. A control structure that indicates the start of an electronic transmission.

| **hierarchical loop.** A mapping construct in which a multi-generational parent/child structure of a repeating nature can be described; also called HL.

| **HL.** See hierarchical loop.

I

IBM Global Network. The worldwide IBM communications network that provides network solutions and a global information infrastructure through IBM and other companies.

ICS. International Control Segments

Interactive System Productivity Facility (ISPF). An IBM-licensed program that serves as a full-screen editor and dialog manager.

interchange. The exchange of information between trading partners.

ISPF. Interactive System Productivity Facility.

J

JCL. Job Control Language

K

key. In a profile member, the field that identifies the member. For example, the key for members of the trading partner profile is the trading partner nickname.

L

Link Pack Area (LPA). In MVS, an area of main storage containing reenterable routines from system libraries. Their presence in main storage saves loading time.

literal. In transaction mapping, a value that is constant for each occurrence of the transaction. If you provide the literal value during mapping, the translator does not have to refer to an application field to obtain the value.

log file. A file in which events are recorded.

logging. The recording of events in time sequence.

loop. A group of related segments in a transaction set.

loop ID. A unique code identifying a loop and the number of times the group can be repeated.

loop repeat. A number indicating the maximum number of times a loop can be used in succession.

LPA. Link pack area.

M

| **map.** A set of instructions that describes how to
| transform application data into EDI data.

maximum use. A number indicating the maximum number of times a segment can be used in a transaction set.

member. A collection of data for one entry in a profile. For example, a member of the trading partner profile contains data about one trading partner.

message. A free-form, usually short, communication to a trading partner. In EDIFACT standards, a transaction set.

| **MQSeries.** A product of the IBM company; used to
| implement messaging and queueing of data groups.

multiple-occurrence mapping. A form of mapping in which all occurrences of a loop or repeating segment are mapped to the same repeating structure in the data format.

MVS. Multiple virtual storage.

N

network acknowledgment. A response from the network indicating the status of an interchange envelope, such as sent or received.

O

ODETTE. Organization for Data Exchange through Teletransmission in Europe.

P

panel body. The area in the middle of the panel that contains entry fields, lists of selectable items, menu choices, and scrollable text.

PDF. Program Development Facility.

PDS. Partitioned data set.

PDS members. Groups of related information stored in partitioned data sets.

| **profile.** A group of similar objects. Each profile can
| contain one or more objects or *profile members*. For
| example, the trading partner profile contains members
| for each of your trading partners (one member per
| specific trading partner). All members have predefined
| fields in which you may enter data. Each member of
| the trading partner profile, for example, has fields that
| will help you identify your trading partners (like company
| name and address). For some fields, such as the key
| field, the data you enter *MUST* be different for each
| member. The first field of the profile member is the *key*
| or name for referring to the member. In the trading
| partner profile, for example, the key is the trading
| partner nickname.

program directory. A document shipped with each release of a product that describes the detailed content of the product.

Q

qualifier. A data element which gives a generic segment or data element a specific meaning. Qualifiers are used in mapping single or multiple occurrences.

quiesce. To end a process by allowing operations to complete normally.

quiescing. The process of bringing a device or a system to a halt by rejection of new requests for work.

R

RACF. Resource access control facility.

release character. The character that indicates that a separator or delimiter is to be used as text data instead of as a separator or delimiter. The release character must immediately precede the delimiter.

repository data. A group of data definitions, formats, and rules, that DataInterchange uses to process your data.

Resource Access Control Facility (RACF). An IBM-licensed program that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources.

S

SAF. System Authorization Facility.

security administrator. The person who controls access to business data and program functions.

| **SAP.** A German company named Systeme,
| Anwendungen, and Produkte specializing in application
| software. A major product, SAP R/3, is a
| component-based architecture/application that
| integrates business processes such as sales, materials
| management, and distribution.

SAP R/3 supports an EDI interface subsystem. SAP R/3 generates application data in the SAP R/3 Intermediate Document (IDOC) layout. This data file is then sent to EDI subsystem via a file transfer product such as FTP or TCP/IP.

segment. A group of related data elements. A segment is a single line in a transaction set, beginning with a function identifier and ending with a segment terminator delimiter. The data elements in the segment are separated by data element delimiters.

segment directory. A file containing the format of all segments in a standard.

segment identifier. A unique identifier at the beginning of each segment consisting of two or three alphanumeric characters.

segment ID separator. The character that separates the segment identifier from the data elements in the segment.

segment terminator. The character that marks the end of a segment.

single-occurrence mapping. A form of mapping in which each occurrence of a loop or repeating segment is mapped to a different part of the data format.

SMP/E. System Modification Program Extended.

SQL. Structured query language.

standards. The industry-supplied, national, or international formats to which information is converted, allowing different computer systems and applications to interchange information.

structure. See *data structure*.

subelement. In EDIFACT standards, a data element that is part of a composite data element. For example, a data element and its qualifier are subelements of a composite data element.

subelement separator. A character that separates the subelements in a composite data element.

System Authorization Facility (SAF).

System Modification Program Extended (SMP/E). An IBM-licensed program used to install software and software changes on OS/VS1 and OS/VS2 systems.

T

tag. In EDIFACT standards, the segment identifier.

TDCC. Transportation Data Coordinating Committee.

TDQ. Transient data queue.

temporary storage queue (TS). Storage locations reserved for immediate results in CICS. They are deleted after the task that created them is complete and they are no longer necessary.

| **Time Sharing Option (TSO).** A component of the IBM
| MVS operating system which allows users full access to
| MVS functionality, but shares machine resources across
| users.

Time Sharing Option Extensions (TSO/E). The base for all TSO enhancements. It provides MVS users with additional functions, improved usability, and better performance.

| **TPT.** See trading partner transaction.

trading partner profile. The profile that defines your trading partners, including information about network account numbers, user IDs, who pays for network charges, etc.

trading partners. Business associates, such as a manufacturer and a supplier, who agree to exchange information using electronic data interchange.

trading partner transaction. A transaction set customized to match the format that two trading partners have agreed to use for exchanging one type of transaction.

trailer. A control structure that indicates the end of an electronic transmission.

transaction. A single business document, such as an invoice.

transaction set. A group of standard data segments, in a predefined sequence, needed to provide all of the data required to define a complete transaction, such as an invoice or purchase order.

transient data queue. A sequential data set used by the Folder Application Facility in MVS/CICS to log system messages.

translation. The process of converting information from a data format to a standard format, or from a standard format to a data format.

translation table. A user-defined table that translates data values that differ between trading partners. For example, if a manufacturer and supplier have different part numbers for the same item, each company can use its own part number and have it converted to the other company's part number during translation.

TSO. See Time Sharing Option.

TSO/E. Time Sharing Option/Extended.

| **TSQ.** Temporary storage queue; a CICS mechanism
| for storing data within the CICS data space; also TS
| queue.

U

UCS. Uniform Communication Standard.

unary operator. An operator that changes the sign of a numeric value.

Uniform Communication Standard (UCS). The EDI standard used in the grocery industry.

UNTDI. United Nations Trade Data Interchange.

V

validation table. A table, supplied by DataInterchange or defined by the user, which contains all acceptable values for a single data field.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number.

VSAM. Virtual Storage Access Method.

W

WINS. Warehouse Information Network Standard.

X

| **X12.** ANSI ASC X12 standard used mostly in North
| American business.

Index

A

- abend CICS return codes 3-3
- ACCTID keyword 1-69
- ACFIELD keyword 1-69
- ACKFILE keyword 1-69
- acknowledgment
 - functions for
 - delayed enveloping 1-82
 - separate file 1-85
 - image printing 1-41
- ACKTYPE keyword 1-69
- activity log profile member (ACTLOGS) 2-51
- activity summary report, example 1-41
- ACTLOGS profile 3-17
- ACTUSAGE keyword 1-70
- ADDRLN1 keyword 1-70
- ADDRLN2 keyword 1-70
- agency code 3-97
- allocating space for records
 - as shipped F-24—F-29
 - calculating F-1—F-23
 - example F-29
- alternate key formats
 - account number/user ID 3-130
 - description 3-130
 - interchange qualifier/ID 3-130
- altnetphone (REQDB) A-56, A-61
- ANY keyword 3-98
- API
 - assembler calls 3-9
 - business tasks 3-12
 - C calls 3-8
 - cancel operations 3-121
 - close and queue interchange 3-84
 - COBOL calls 3-5
 - combined with DataInterchange Utility 3-13
 - communication services 3-105—3-125
 - communications 6-1
 - data extraction services 3-101—3-105
 - de-envelope API 3-86
 - description 3-1
 - end translation/enveloping 3-85
 - enveloping services 3-69—3-100
 - environmental services 3-17—3-19
 - get envelope service 3-136
 - initialize CICS 3-18
 - initialize SYNC 3-134
 - internal calls
 - process network acknowledgments 3-125
 - queue standard data 3-124
 - issue commit 3-98

API (continued)

- link edit 3-2
- load module relationships 3-4
- PL/I calls 3-6
- put envelope service 3-136
- receive 3-117
- retrieve detailed data 3-102
- retrieve functional acknowledgment image 3-105
- retrieve group header 3-100
- retrieve interchange header 3-99
- retrieve transaction acknowledgment image 3-105
- retrieve transaction header 3-100
- retrieve transaction image 3-104
- return codes B-17
- return filename 3-123
- send files 3-115
- send transactions 3-109
- services overview 3-13—3-17
- status update 3-126
- status update services 3-126—3-131
- supported languages 3-1
- syncpoint services 3-132—3-136
- translate to standard 3-27
- translation services 3-20—3-69
- utility services
 - when to use 3-12
- APPDEFS profile 3-17
- APPFILE keyword 1-70
- application data format definition 2-37
- application data format structure 2-38
- application definition profile Member (APPDEFS) 2-51
- application file 2-3
- APPLICATION keyword 1-71
- application program interface (API)
 - assembler calls 3-9
 - business tasks 3-12
 - C calls 3-8
 - cancel operations 3-121
 - close and queue interchange 3-84
 - COBOL calls 3-5
 - combined with DataInterchange Utility 3-13
 - communication services 3-105—3-125
 - communications 6-1
 - data extraction services 3-101—3-105
 - de-envelope API 3-86
 - description 3-1
 - end translation/enveloping 3-85
 - enveloping services 3-69—3-100
 - environmental services 3-17—3-19
 - get envelope service 3-136
 - initialize CICS 3-18
 - initialize SYNC 3-134

- application program interface (API) (*continued*)
 - internal calls
 - process network acknowledgments 3-125
 - queue standard data 3-124
 - issue commit 3-98
 - link edit 3-2
 - load module relationships 3-4
 - PL/I calls 3-6
 - put envelope service 3-136
 - receive 3-117
 - retrieve detailed data 3-102
 - retrieve functional acknowledgment image 3-105
 - retrieve group header 3-100
 - retrieve interchange header 3-99
 - retrieve transaction acknowledgment image 3-105
 - retrieve transaction header 3-100
 - retrieve transaction image 3-104
 - return codes B-17
 - return filename 3-123
 - send files 3-115
 - send transactions 3-109
 - services overview 3-13—3-17
 - status update 3-126
 - status update services 3-126—3-131
 - supported languages 3-1
 - syncpoint services 3-132—3-136
 - translate to standard 3-27
 - translation services 3-20—3-69
 - utility services
 - when to use 3-12
- APPLID keyword 1-71
- APPLID parameter, MVS 1-116
- APPLID, switching 1-71, A-5
- APPRECID keyword 1-72
- APPSEC keyword 1-72
- APPSNDID keyword 1-72
- APPTYPE keyword 1-73
- archive JCL (EDIELARV)
- ARCHIVEFILE keyword 1-73
- ARCHIVETYPE keyword 1-73
- ASCII filter example D-72—D-78
- ASCII/BAUDOT filter example D-78—D-85
- assembler calls 3-9
- ASSERTLVL keyword 1-74
- ASSOBJ field
 - export 2-14
 - import 2-15
- ATSID 3-26
- authentication routine
 - description 4-22
 - parameters
 - assembler definition 4-25
 - C definition 4-25
 - COBOL definition 4-24
 - description 4-23
 - sample programs D-85—D-95

B

- batch control file (CTLFIL) 2-12
- batch ID, assignment 1-74
- BATCH keyword 1-74
- BATCHID keyword
- BATCHSET keyword 1-74
- BATFLG field 5-20, 5-22
- BNDLFLAG field 2-67
- boundary 3-44
- buffer size, for security exit routines 4-18
- bundled transactions 3-69
 - BNDLFLAG for translating
 - enveloping 3-82

C

- C calls 3-8
- C record
 - for translating to standard
 - field labels 2-63—2-67, 2-69
 - format 2-61
 - returned with received transactions
 - field labels
 - format
- CAF 3-2
- calls
 - assembler 3-9
 - C 3-8
 - COBOL 3-5
 - first for transaction
 - translate file 3-61
 - translate to application 3-52
 - translate to standard 3-30, 3-36
 - first of session
 - translate file 3-58
 - translate to application 3-50
 - translate to standard 3-28
 - last for transaction
 - translate file 3-68
 - translate to application 3-57
 - translate to standard
 - last of session
 - de-envelope API 3-96
 - envelope API 3-81
 - translate file 3-68
 - translate to application 3-57
 - translate to standard 3-39
 - other than first or last
 - translate file 3-67
 - translate to application 3-56
 - translate to standard 3-36
 - PL/I 3-6
- canceled operations
 - CMCB initialization 3-122
 - description 3-121

- canceling operations (*continued*)
 - information returned 3-122
- CATEGORY field 2-13
- CCEXCEPTION keyword 1-75
- CICS abend return codes 3-3
- CICS envelope queue alternatives 5-4
- CICS region, for running DataInterchange Utility 5-11
- CICS region, Information Exchange (IE) session
 - mailbox 5-35
 - perform network function 5-35
 - session cleanup 5-37
- CICS transactions, supplied by DataInterchange 5-43
- CICS, interface between DataInterchange for CICS, Expedite/CICS and Information Exchange 5-35
- CLEARFILE keyword 1-75
- CLOSE MAILBOX command 1-63
- clustered transactions 3-69
 - BNDLFLAG for translating enveloping 3-82
- CMDFVAL value 6-12
- CMMTLN1 keyword 1-75
- CMMTLN2 keyword 1-76
- CMPPYNM keyword 1-76
- CNTCTNM keyword 1-76
- CNTCTPH keyword 1-76
- COBOL calls 3-5
- COBOL HOT-DI examples D-49
- COBOL response program example D-56
- command field value (CMDFVAL) 6-12
- command file (EDISYSIN, SYSIN) 2-1
- command language
 - criteria 1-3
 - description 1-1
 - keywords 1-115
 - options 1-3
 - PERFORM statement 1-2
 - range criteria 1-3
 - SELECTING clause 1-2
 - syntax 1-1
 - validating input 1-4
 - WHERE clause 1-2
- comments definition 2-33
- COMMIT function
 - description 3-135
 - issue commit API 3-98
 - return codes B-29
- common control block (CCB)
 - API function 3-1
 - copy books
 - ASSEMBLER C-36
 - COBOL C-2
 - description A-2
 - field descriptions A-3—A-4
 - include files
 - C C-25
 - PL/I C-13
- common control block (CCB) (*continued*)
 - layout A-3
- communication
 - return codes B-27—B-29
- communication control block (CMCB)
 - copy books
 - ASSEMBLER C-38
 - COBOL C-4
 - field descriptions A-35—A-42
 - include files
 - C C-26
 - PL/I C-15
 - initialization for
 - cancel API 3-122
 - process network acknowledgments API 3-125
 - receive API 3-118
 - return filename API 3-124
 - send files API 3-115
 - send transactions API 3-110
 - layout A-34
 - output fields for communication services 3-108
- communication data block (DATABLK)
 - copy books
 - ASSEMBLER C-45
 - COBOL C-11
 - field descriptions
 - include files
 - C C-34
 - PL/I C-22
 - layout A-52
- communication routine 6-38
 - for CICS
 - description 6-14
 - invoking 6-14
 - network interface 6-2
 - parameters from network interface 6-9
- communication service, In*Touch Gateway 6-21
- communication services
 - cancel API 3-121
 - CMCB output fields 3-108
 - communication routine 3-106
 - description 3-105
 - internal API calls 3-124
 - message handler 3-106
 - network program 3-106
 - overview 3-15
 - receive API 3-117
 - return codes 3-109
 - return filename API 3-123
 - send files API 3-115
 - send transactions API 3-109
 - trading partner profile data block (TPPDB) 3-107
- communications API 6-1
- compression routine
 - C C-25
 - CCB C-25
 - CMCB C-26

compression routine (*continued*)

C (*continued*)

DATABLK C-34

FCB C-26

HUGEBLK C-34

NPDB C-30

REQDB C-33

SNB C-25

SPDB C-34

TPPDB C-31

TRCB C-27

utility control information block C-35

description 4-26

parameters

assembler definition 4-29

C definition 4-29

COBOL definition 4-28

description 4-27

CONCATENATE keyword 1-76

condition codes

DataInterchange Utility B-2—B-17

JCL job step condition codes B-2

noncommunications B-2

nontranslation B-2

UTILCCODE B-2

UTILSEV B-2

contacts definition 2-33

continuous receive 1-65

cleanup 5-39

considerations 5-23

description 6-19

invoking 6-19

processing of data 5-25

reporting 1-67

response applications 5-28

selection criteria 5-24

starting and stopping requests 5-38

using EDI1 TD queue 5-26

without Expedite/CICS 5-26

CONTRECV member 2-57

control blocks

common (CCB) A-2—A-4

communication data block (DATABLK) A-52

communication interface (CMCB) A-34—A-42

function (FCB) A-4—A-6

network profile (NPDB) A-53—A-55

requestor profile (REQDB) A-56—A-61

service name (SNB) A-1—A-2

trading partner profile (TPPDB) A-42—A-52

translator (TRCB) A-6—A-29

translator input data (TRIDB) A-29—A-31

translator output data (TRODB) A-31—A-33

control information

format for DataInterchange Utility

description 5-18

field descriptions 5-20—5-23

control information (*continued*)

message handler 6-17

network program 6-15

passing to DataInterchange Utility 5-2

control record

for translating to standard

field labels 2-63—2-67, 2-69

format 2-61

returned with received transactions

field labels

format

copy books

ASSEMBLER C-36—C-47

CCB C-36

CMCB C-38

DATABLK C-45

FCB C-37

HUGEBLK C-45

NPDB C-42

REQDB C-44

SNB C-37

SPDB C-46

TPPDB C-43

TRCB C-39

utility control information block C-46

COBOL C-2—C-13

CCB C-2

CMCB C-4

DATABLK C-11

FCB C-3

HUGEBLK C-11

NPDB C-8

REQDB C-10

SNB C-3

SPDB C-12

TPPDB C-9

TRCB C-5

utility control information block C-12

PL/I C-13—C-24

CCB C-13

CMCB C-15

DATABLK C-22

FCB C-14

HUGEBLK C-22

NPDB C-19

REQDB C-21

SNB C-14

SPDB C-23

TPPDB C-20

TRCB C-16

utility control information block C-23

CTLFILE

description 2-12

fields 2-13

CTLTYPE keyword 1-76
customization data, exporting/importing 1-63

D

D records
 description 2-70
 format of
 structures passed separately 2-71
 structures passed together 2-70
data area, export/import 2-17
data element errors B-20
data extract exit routine 4-32
data extract record layouts
 acknowledgment image 2-87
 application 2-86
 group 2-83
 interchange 2-81
 network activity 2-78
 trading partner capability 2-77
 trading partner profile 2-76
 transaction 2-83—2-86
 transaction activity 2-79
 transaction image 2-87
 transaction store common key 2-81
data extract report generators D-27—D-44
data extraction services
 description 3-101
 initializing control blocks for 3-101
 retrieve detailed data API 3-102
 retrieve functional acknowledgment image
 API 3-105
 retrieve transaction acknowledgment image
 API 3-105
 retrieve transaction image API 3-104
data modes for translate to standard 3-25
data records
 description 2-70
 format of
 structures passed separately 2-71
 structures passed together 2-70
DataInterchange
 DB2 command file 2-2
DataInterchange DB2 command file 2-2
DataInterchange Utility
 condition codes B-2—B-17
 noncommunications B-2
 nontranslation B-2
 continuous receive
 cleanup 5-39
 considerations 5-23
 processing of data 5-25
 selection criteria 5-24
 starting and stopping requests 5-38
 using EDI1 TD queue 5-26
 without Expedite/CICS 5-26

DataInterchange Utility (*continued*)
 control information
 field descriptions 5-20—5-23
 format 5-18
 description 1-1
 JCL job step condition codes B-2
 MVS parameters 1-116
 RECOVERY keyword 5-10
 response applications
 description 5-27
 getting results from DataInterchange 5-27
 types 5-27—5-31
 ways of starting 5-27
 return codes B-17
 running in CICS
 DB2 setup considerations 5-11
 description 5-1
 ensuring serial processing 5-5
 getting information from DataInterchange 5-2
 passing control information 5-2
 recovery 5-6
 separate region 5-11
 storage mechanisms 5-3
 units of work 5-6
 ways to start 5-1
 temporary storage queues 5-33—5-35
 terminal applications 5-11
 transient data queues 5-33—5-35
 unit or work
 as part of application's 5-10
 description 5-7
 ensuring 5-10
 UTILCCODE B-2
 UTILSEV B-2
DataInterchange Utility JCL (EDIUTILV)
 date and time format, utility keywords 1-4
 DAYS keyword 1-77
 DB/2 Timeout / Deadlock processing 3-132
 DB2 and DI utility 2-2
 DB2 Attachment 3-2
 de-enveloping
 API 3-86
 description 3-86
 initializing
 FCB 3-87
 SNB 3-87
 TRCB 3-87
 TRIDB 3-90
 TRODB 3-90
 last call of session 3-96
 transactions 3-90
 TRCB return fields 3-91—3-96
DEENVELOPE AND TRANSLATE command
 description 1-25
 examples 1-26
 syntax 1-26

- DEENVELOPE command
 - description 1-22
 - examples 1-23
 - syntax 1-22
- DELETE PROFILE command
 - description 1-64
 - examples 1-64
 - syntax 1-64
- delimiter, utility commands 1-116
- description 1-67, 1-68
- DI variables
- dial connect num 6-40
- DIERRFILTER 1-4
- DIERRFILTER keyword 1-77
- DIR keyword 1-77
- DLM parameter 1-116
- DLVDATE keyword 1-78
- DLVTIME keyword 1-78
- documents, purging 1-32
- DSN 3-2
- dump, GLB 1-68
- DUPCHECK keyword 1-78
- DUPENV keyword 1-79
- DUPTRANS field
- DYNSQL keyword 1-79

E

- earliest envelope date 1-79
- ECB address 5-19
- EDIELARV JCL
- EDIFACT profile member (E) 2-52
- EDIFAENV
 - description 2-9
 - example 2-10
 - file format 2-9
 - sample entries 2-10
- EDIFFUT 2-2
- EDINTCMD 2-2
- EDIQUERY 2-7
- EDISYSIN 2-1
- EDITEST 3-25, 3-27
- EDITPXRF database record F-1
- EDITSIN 2-2
- EDITSIN examples 3-3
- EDITSIN keywords 3-2
- EDIUTILV JCL
- EDIVAX 2-8, E-16
- EDIVTSAU 3-21
- EDIVTSEV 3-21
- EDIVTSGP 3-21
- EDIVTSTH 3-20
- EDIVTSTI 3-21
- EDIVTSTO 3-21
- EDIVTSTU 3-21

- EDIXF2T sample program D-1
- EDIXTAGF sample program D-1
- EENVDATE keyword 1-79
- EIFORMAT keyword 1-79
- EJECT 3-41
- encryption routine
 - description 4-18
 - parameters
 - assembler definition 4-22
 - C definition 4-21
 - COBOL definition 4-20
 - description 4-19
 - sample programs D-95—D-117
- end transaction/interchange record
 - description 2-71
 - format 2-72
- ending
 - DataInterchange
 - return codes B-18
 - sample program D-26
 - enveloping 3-85
 - translation
 - API 3-85
 - sample program D-18
- ENVCHK 3-81
- ENVDATE keyword 1-80
- envelope
 - date 1-80
 - date, earliest 1-79
 - file 1-83, 2-3
 - header record (E) 2-75
 - options file (FAENV, EDIFAENV) 2-9
 - segments
 - EDIFACT 3-70
 - ICS 3-70
 - UCS 3-70
 - UNTDI 3-70
 - X12 3-70
- ENVELOPE AND SEND command
 - description 1-12
 - examples 1-14
 - syntax 1-13
- ENVELOPE command
 - description 1-7
 - examples 1-9
 - syntax 1-8
- ENVELOPE DATA EXTRACT command
 - description
 - examples 1-59, 1-60
 - syntax 1-59
- envelope key formats
 - account number/user ID 3-129
 - description 3-128
 - interchange qualifier/ID 3-129
 - trading partner nickname 3-128
 - transaction handle 3-129

- enveloping services
 - API 3-73
 - close and queue interchange API 3-84
 - de-envelope API 3-86
 - de-enveloping function 3-86—3-98
 - description 3-69
 - end translation/enveloping API 3-85
 - envelope segments 3-70
 - enveloping function 3-72, 3-86
 - group layer 3-71
 - interchange layer 3-71
 - issue commit API 3-98
 - last call of session
 - de-enveloping 3-96
 - enveloping 3-81
 - overview 3-14
 - retrieve group header API 3-100
 - retrieve interchange header API 3-99
 - retrieve transaction header API 3-100
 - special considerations, de-enveloping
 - locating trading partners 3-96
 - locating transaction maps 3-97
 - special considerations, enveloping 3-81—3-84
 - clustered transactions 3-82
 - envelope versus reenvelope 3-81
 - sending transaction data 3-82
 - transaction layer 3-72
- environmental errors B-25
- environmental services
 - description 3-17
 - initialization
 - description 3-17
 - parameters 3-17
 - return codes 3-18
 - overview 3-14
 - termination
 - description 3-19
 - parameters 3-19
 - return codes 3-19
- ENVLDATE 3-30
- ENVLDELAY 3-23
- ENVSERV service 3-14
- ENVTIME keyword 1-80
- ENVTYPE keyword 1-81
- EPURDATE keyword 1-81
- ERRFILTER field 3-30, A-24
- error code values A-28
- error filtering 1-4
- errors
 - combined RECEIVE commands B-17
 - data elements B-20
 - DIERRFILTER keyword 1-77
 - ENVELOPE AND SEND command B-17
 - environmental B-25
 - ERRFILTER 3-30, A-24
 - filtering 1-4
- errors (*continued*)
 - groups B-23
 - interchanges B-24
 - invalid data B-25
 - program B-25
 - REENVELOPE AND SEND command B-17
 - segments B-20
 - transactions B-21
 - TRANSLATE AND SEND command B-16
- ETYPE field 2-64
- event log printing 1-45
- examples 1-68
- exception file (FFSEXCP) 2-4
- exit program (independent)
 - translator control block (TRCB) 4-33
 - IUSERACCESS 4-33
 - IUSERAREA 4-33
 - IUSEREXIT 4-33
 - IUSERTYPE 4-33
 - utility control statements 1-87, 1-88, 1-91, 4-33
 - IACCESS 1-87, 4-33
 - IAREA 1-88, 4-33
 - IEXIT 1-88, 4-33
 - ITYPE 1-91, 4-33
- exit program (user exit)
 - translator control block (TRCB) 4-13
 - IUSERACCESS 4-13
 - IUSERAREA 4-13
 - IUSEREXIT 4-13
 - IUSERTYPE 4-13
 - utility control statements 1-87, 1-88, 1-91, 4-13
 - IACCESS 1-87, 4-13
 - IAREA 1-88, 4-13
 - IEXIT 1-88, 4-13
 - ITYPE 1-91, 4-13
- exit routines
 - data extract 4-32
 - description 4-1
 - field
 - description 4-3
 - parameter definitions 4-7—4-9
 - receive parameters 4-6
 - send parameters 4-4
 - shipped with DataInterchange 4-4
 - get/put envelope exit 4-13
 - independent programs 4-32
 - instructions for linkage editor 4-3
 - interactive with session 4-1
 - languages 4-2
 - security
 - authentication routine 4-22—4-26
 - compression routine 4-26
 - description 4-15
 - enabling during receive 4-17
 - enabling during send 4-17
 - encryption routine 4-18—4-22
 - filtering routine 4-26—4-30

- exit routines (*continued*)
 - security (*continued*)
 - parameters introduction 4-18
 - supporting routines 4-30—4-32
 - transaction
 - description 4-9
 - parameter definitions 4-11—4-13
 - parameters description 4-10
 - post-translation 4-9
 - pre-translation 4-9
 - user extensions 4-1
- EXPORT command 1-62
- export/import common control record (0C1/0C2) 2-16
- export/import common end of group record (000) 2-17
- export/import control file (CTLFIL)
 - data files 2-15
 - description 2-12
 - fields 2-13
 - maintaining export 2-15
- export/import file data area 2-17
- export/import of application data formats 2-37
- export/import of standards records 2-34
- export/import of table definitions 2-45
- export/import of trading partner profile 2-18
- export/import of trading partner transactions 2-39
- export/import trading partner profile header
 - record 2-19
- export/import trading partner profile record 2-19
- export/import utility function 2-12
- extraction services 3-15
 - See also* data extraction services

F

- FADELAY keyword 1-82
- FAENV
 - description 2-9
 - example 2-10
 - file format 2-9
 - sample entries 2-10
- FAERC field 3-64, 3-92, A-27
- FARC field 3-64, 3-92, A-27
- FFMTCBP field 5-23
- FFMTFLG field 5-22
- FFSEXCP 2-4
- FFSTRAK 2-5
- FFSWORK 2-8
- field exit routines
 - description 4-3
 - parameters
 - assembler definition 4-9
 - C definition 4-8
 - COBOL definition 4-7
 - description 4-7
 - receive 4-6
 - send 4-4

- field exit routines (*continued*)
 - sample programs D-57—D-65
 - shipped with DataInterchange 4-4
- file maintenance performance techniques G-4
- FILEID keyword 1-83
- files
 - application 2-3
 - batch control 2-12
 - command 2-1
 - CTLFIL 2-12
 - DataInterchange Utility 2-1
 - EDIFAENV 2-9
 - EDIQUERY 2-7
 - EDISYSIN 2-1
 - envelope 1-83, 2-3
 - enveloping options 2-9
 - exception 2-4
 - export/import control 2-12
 - FAENV 2-9
 - FFSEXCP 2-4
 - FFSTRAK 2-5
 - FFSWORK 2-8
 - functional acknowledgments 2-9
 - INCNTL 2-59
 - mapping migration
 - input control 2-59
 - output control 2-60
 - OUTCNTL 2-60
 - print 2-5
 - PRTFIL 2-5
 - query 2-7
 - report 2-6
 - RPTFIL 2-6
 - sending 3-115
 - SYSIN 2-1
 - tracking 2-5
 - work 2-8
- filter type, tests for D-65
- filtering errors 1-4
- filtering routine
 - description 4-26
 - parameters
 - assembler definition 4-29
 - C definition 4-29
 - COBOL definition 4-28
 - description 4-27
 - sample programs D-68—D-85
- fixed to fixed (runtime)
 - ENVELOPE AND SEND command 1-18
 - ENVELOPE command 1-16
 - REENVELOPE AND SEND command 1-18
 - REENVELOPE command 1-17
 - SEND command 1-17
 - TRANSLATE AND ENVELOPE command 1-17
 - TRANSLATE AND SEND command 1-17
 - TRANSLATE TO STANDARD command 1-16

- Fixed translation
 - format of data 1-101, 2-71, 2-72, 3-30
 - where data is written 2-4
- FIXEDFIELD keyword 1-83
- FORCETEST field 3-51, 3-60, 3-89, A-26
- FORCETEST keyword 1-84
- FORMAT keyword 1-84
- FORMATID field 2-63
- FSUPPORT member 6-12
- FUNACKFILE keyword 1-85
- FUNACKP keyword 1-85, 1-86
- function control block (FCB)
 - API function 3-1
 - copy books
 - ASSEMBLER C-37
 - COBOL C-3
 - field descriptions A-5
 - include files
 - C C-26
 - PL/I C-14
 - initializing for
 - data extract 3-102
 - de-envelope API 3-87
 - envelope API 3-74
 - translate to application API
 - translate to standard API 3-28, 3-50
 - layout A-4
- functional acknowledgments
 - delayed enveloping 1-82
 - enveloping options file (FAENV, EDIFAENV) 2-9
 - retrieving image of 3-105
 - separate file 1-85
- functional acknowledgments, overdue 1-48
- FXXZASM function 3-9
- FXXZC function 3-8
- FXXZCBL routine 3-5
- FXXZPLI routine 3-6

G

- GAPW field 2-66
- Gateway 6-21
- get envelope 4-13
- get/put envelope exit and service 4-13
- get/put envelope program 4-33
- GLB DUMP command
- GLB TRACE command
- GREL field 2-66
- GRID field 2-66
- group errors B-23
- group header (G) record 2-75
- group header, retrieving 3-100
- GROUP keyword 1-86
- group layer 3-71
- GRPCTLNO keyword 1-86

- GSID field 2-66
- GTYPE field 2-65
- GVER field 2-66

H

- handle (Transaction Store ID) 1-86
- HANDLE keyword 1-86
- Harbinger In*Touch Gateway 3-106, 6-21
- hexadecimal filter example D-68—D-72
- HOLD command
 - description 1-30
 - examples 1-31
 - syntax 1-30
- HOLDFILE keyword 1-87
- HOLDFLAG field 2-67
- HOLDTYPE keyword 1-87
- HOT-DI environment 5-12
- HOT-DI, initializing, terminating D-49

I

- IACCESS keyword 1-87
- IAPREF field 2-66
- IAREA keyword 1-88
- ICS profile member (I) 2-54
- ID keyword 1-88
- IEXIT keyword 1-88
- IFCC keyword 1-89
- IKJEFT01 2-2
- IMAGE keyword 1-88
- IMGLIFE keyword 3-89
- IMPORT command 1-63
- importing new definitions to DataInterchange 2-59
- importing, tables D-1
- In*Touch Gateway 3-106, 6-21
- inbound envelope program example D-120
- include files
- INCNTL
 - description 2-59
 - file format 2-60
- incremental translation 3-44
- information (I) record
 - description 2-74
 - requesting
- initializing
 - CICS API 3-18
 - control blocks
 - de-envelope API 3-87
 - envelope API 3-74
 - DataInterchange
 - return codes B-17
 - sample programs D-1—D-9
 - HOT-DI D-49
 - SYNC, return codes B-29

- INMEMTRANS keyword 1-89, 3-42
- input record length 6-38
- INTCTL keyword 1-89
- INTCTLNO keyword 1-89
- interchange
 - closing and queueing layer 3-71
 - retrieving header 3-99
- interchange errors B-24
- INTERCHANGE keyword 1-90
- interface, network
 - building commands for 6-9
 - CICS communication routine
 - description 6-14
 - network profile definition 6-14
 - communication routine
 - description 6-2
 - passing parameters to 6-9
 - continuous receive
 - description 6-19
 - invoking 6-19
 - data flow
 - description 6-6—6-9
 - illustration 6-6
 - description 6-1
 - example of operation 6-11
 - generalized 6-2
 - message handler
 - control information 6-17
 - description 6-13
 - error occurred 6-18
 - functions of 6-13
 - parameters passed to 6-13
 - successful condition 6-18
 - network operation profile 6-9—6-11
 - network program
 - calling 6-3
 - control information 6-15
 - error occurred 6-17
 - no data received 6-17
 - successful condition 6-17
 - point-to-point 6-4
 - supported by DataInterchange 6-1
 - writing 6-1
- internal trading partner ID, default value 1-109
- INTID keyword 1-90
- INTPID field 2-63, 3-30
- INTRECID keyword 1-90
- INTSNDID keyword 1-90
- INTYPE keyword 1-91
- invalid data errors B-25
- invoking response programs D-56
- IRID field 2-65
- ISID field 2-65

- ISPW field 2-66
- ITPBREAK keyword 1-91
- ITYPE field 2-64
- ITYPE keyword 1-91
- IUSERACCESS field 3-88, A-26
- IUSERTYPE field 3-88, A-26
- IVERREL field 2-65

J

- JCL samples
 - archive (EDIELARV)
 - changes from previous release
 - data sets for DataInterchange Utility E-17
 - DataInterchange Utility (EDIUTILV)
 - description E-1
 - modifications for DB2
 - restore JCL

K

- KEYID field 2-13
- keyword 1-76
- keywords
 - ACCTID 1-69
 - ACFIELD 1-69
 - ACKFILE 1-69
 - ACKTYPE 1-69
 - ACTUSAGE 1-70
 - ADDRLN1 1-70
 - ADDRLN2 1-70
 - ANY 3-98
 - APPFILE 1-70
 - APPLICATION 1-71
 - APPLID 1-71
 - APPRECID 1-72
 - APPSEC 1-72
 - APPSNDID 1-72
 - APPTYPE 1-73
 - BATCH 1-74
 - BATCHSET 1-74
 - CCEXCEPTION 1-75
 - CLEARFILE 1-75
 - CMMTLN1 1-75
 - CMMTLN2 1-76
 - CMPYNM 1-76
 - CNTCTNM 1-76
 - CNTCTPH 1-76
 - command language groups 1-3
 - CONCATENATE 1-76
 - CTLFILE 1-76
 - CTLTYPE 1-76
 - DataInterchange Utility 1-115
 - date format in 1-4
 - DAYS 1-77
 - DIERRFILTER 1-77

keywords (*continued*)

DIR 1-77
DLVDATE 1-78
DLVTIME 1-78
DUPCHECK 1-78
DUPENV 1-79
DYNSQL 1-79
EENVDATE 1-79
EIFORMAT 1-79
ENVDATE 1-80
ENVTIME 1-80
ENVTYPE 1-81
EPURDATE 1-81
FADELAY 1-82
FILEID 1-83
FORCETEST 1-84, 3-51
FORMAT 1-84
FUNACKFILE 1-85
FUNACKP 1-85, 1-86
GROUP 1-86
GRPTLNO 1-86
HANDLE 1-86
HANDLE format in 1-4
ID 1-88
IMAGE 1-88
INCNTL 1-89
INMEMTRANS 1-89
INTCTLNO 1-89, 1-94
INTERCHANGE 1-90
INTID 1-90
INTRECID 1-90
INTSNDID 1-90
INTYPE 1-91
ITPBREAK 1-91
KNOWN 3-98
LASTRXDATE 1-92
LEVEL 1-92
MAPCHAIN 3-51
MAPFROM 1-93
MAPTO 1-93
MAXRUNTIME
MEMBER 1-94
MERGED 1-94
MRREQID 1-94
MSGUCLASS 1-94
NETACKP 1-95
NETID 1-95
NETNAME 1-96
NETSTAT 1-96
NEWMSG 1-96
NEWSTD 1-97
NEWTRX 1-97
NUMDELS 1-97
NUMUPDTS 1-98
ONELOGICAPP 1-98
OPTRECS 1-98

keywords (*continued*)

OUTCNTL 1-99
OUTFILE 1-99
OUTFORMAT 1-99
OUTTYPE 1-99
PRIORTO 1-100
PURGINT 1-100
RAWDATA 1-101
RAWFMTID 1-101
RAWTEST 1-101
RECEIVEACKDATA 1-103
RECEIVEACKIMAGE 1-103
RECOVERY 1-103
REPLACE 1-104
REQID 1-104
REQTP 1-104
RESET 1-104, 1-105
SEGMENTED 1-105
SENDACKDATA 1-105
SENDACKIMAGE 1-106
SERVICESEGVAL 1-106
SNDDATE 1-106
SNDTIME 1-107
STANDALONE 1-107
STDDESC 1-108
STDID 1-108
STDLV 1-108
STDTRID 1-108
STDVR 1-108
STSTAT 1-108
TESTMODE 1-109
time format in 1-4
TPID 1-109
TPNICKN 1-110
TPNICKNESEND 1-111
TPTID 1-111
TRANSACTION 1-111
TRERLVL 1-111
TRXCTLNO 1-112
TRXDATE 1-112
TRXSTAT 1-113
TRXTIME 1-114
USERID 1-115
USERPGM 1-115
VERIFY 1-115

L

LANGID parameter 1-116
language profile member (LANGPROF) 2-52
languages
 API support 3-1
 exit routines 4-2
LASTTRXDATE keyword 1-92
length of records 6-38

- LEVEL keyword 1-92
- link edit 3-2
- linkage editor instructions, for exit routines 4-3
- load module relationships 3-4
- locating
 - trading partner transactions (maps) 3-97
 - trading partners in TPPROF 3-96
- LOGAEID keyword 1-92
- LOGDATE keyword 1-92
- LOGFILE keyword 1-93
- LOGFORM keyword 1-93
- LOGTIME keyword 1-93
- LOGUSER keyword 1-93

M

- management reporting 1-48
- map, locate 3-97
- MAPCHAIN field 3-51, 3-60, A-26
- MAPCHAIN keyword
- MAPFROM keyword 1-93
- mapping migration
 - input control file (INCNTL)
 - description 2-59
 - file format 2-60
 - output control file (OUTCNTL)
 - description 2-60
 - file format 2-61
- MAPPING MIGRATION command
 - description 1-61
 - examples 1-62
 - syntax 1-61
- MAPTO keyword 1-93
- MAXRUNTIME keyword 1-94
- MBRNAME field 2-15
- MEMBER keyword 1-94
- MERGED keyword 1-94
- message handler
 - control information 6-17
 - description 6-13
 - error occurred 6-18
 - functions of 6-13
 - parameters passed to 6-13
 - successful condition 6-18
- message name, sending transactions 3-114
- message user class, sending transactions 3-114
- MQSeries queue profile member (MQSERIES-P2) 2-59
- MQSeries queues 2-59, 5-3, 5-4, 5-7, 5-23, 5-24, 6-37, 6-40
- MRREQID keyword 1-94
- MSGUCLASS keyword 1-94
- multiple unit of work data mode 3-26
- MUWIND field 2-64

N

- negative return codes B-18
- net acks file 6-39
- netackfile (NPDB) A-53, A-55
- NETACKP keyword 1-95
- netcmdmbr (REQDB) A-56, A-61
- netcmds (TPPDB) 3-108, A-43, A-50
- NETID keyword 1-95
- NETNAME keyword 1-96
- NETOP profile 6-9
- netphone (NPDB) A-53, A-55
- NETSTAT keyword 1-96
- network
 - activity report 1-40
 - commands file 2-2
 - generalized 6-2
 - operation
 - example 6-11
 - receiving transactions 3-119
 - sending transactions 3-112
 - point-to-point 6-4
 - supported by DataInterchange 6-1
 - updating status 1-29
- NETWORK ACTIVITY DATA EXTRACT command
 - description
 - examples 1-56, 1-57
 - syntax 1-56
- network activity report generator D-44—D-49
- network commands file 2-2
- network commands, building for network interface 6-9
- network interface
 - building commands for 6-9
 - CICS communication routine
 - description 6-14
 - network profile definition 6-14
 - communication routine
 - description 6-2
 - passing parameters to 6-9
 - continuous receive
 - description 6-19
 - invoking 6-19
 - data flow
 - description 6-6—6-9
 - illustration 6-6
 - description 6-1
 - example of operation 6-11
 - generalized 6-2
 - message handler
 - control information 6-17
 - description 6-13
 - error occurred 6-18
 - functions of 6-13
 - parameters passed to 6-13
 - successful condition 6-18
 - network operation profile 6-9—6-11

- network interface (*continued*)
 - network program
 - calling 6-3
 - control information 6-15
 - error occurred 6-17
 - no data received 6-17
 - successful condition 6-17
 - point-to-point 6-4
 - supported by DataInterchange 6-1
 - writing 6-1
- network operation profile, for network interface 6-9—6-11
- network operations profile member (NETOP) 2-50
- network profile data block (NPDB)
 - copy books
 - ASSEMBLER C-42
 - COBOL C-8
 - field descriptions A-54—A-55
 - include files
 - C C-30
 - PL/I C-19
 - layout A-53
- network profile member (NETPROF) 2-50
- network program
 - calling 6-3
 - control information 6-15
 - error occurred 6-17
 - no data received 6-17
 - successful condition 6-17
- NEWAPPLID keyword 1-96
- NEWMSG keyword 1-96
- NEWSTD keyword 1-97
- NEWTRX keyword 1-97
- non-EDI data, receiving 1-21
- normal conditions, return codes B-19
- ntackpgm (REQDB) A-56, A-61
- NUMDELS keyword 1-97
- NUMUPDTS keyword 1-98

O

- ONELOGICAPP keyword 1-98
- optimization performance techniques G-1
- optional records
 - description 2-72
 - envelope header (E) 2-75
 - group header (G) 2-75
 - information (I) 2-74
 - queuing totals (Q) 2-76
 - transaction set header (T) 2-76
- OPTRECS keyword 1-98
- outbound envelope program example D-121
- outbound processing
 - fixed to fixed 1-16
- OUTCNTL
 - description 2-60

- OUTCNTL (*continued*)
 - file format 2-61
- OUTCNTL keyword 1-99
- OUTFILE keyword 1-99
- OUTFORMAT keyword 1-99
- OUTTYPE keyword 1-99

P

- pageable translation 1-99, 1-100, 2-8, 3-45, E-16
- parameters, network 6-38
- PERFORM commands
 - building for network interface 6-9
 - CLOSE MAILBOX 1-63
 - DATA EXTRACT 1-47
 - DEENVELOPE 1-22
 - DEENVELOPE AND TRANSLATE 1-25
 - DELETE PROFILE 1-64
 - DUPCHECK
 - EIFORMAT
 - ENVELOPE 1-7
 - ENVELOPE AND SEND 1-12
 - ENVELOPE DATA EXTRACT 1-59
 - EXPORT 1-62
 - GLB DUMP 1-67
 - GLB TRACE 1-68
 - HOLD 1-30
 - IMPORT 1-63
 - MAPPING MIGRATION 1-61
 - NETWORK ACTIVITY DATA EXTRACT 1-56
 - PRINT ACKNOWLEDGMENT IMAGE 1-41
 - PRINT ACTIVITY SUMMARY 1-40
 - PRINT CUSTOM LAYOUT 1-46
 - PRINT EVENT LOG 1-45
 - PRINT STATUS SUMMARY 1-43
 - PRINT STATUS SUMMARY2 1-44
 - PRINT TRANSACTION DETAILS 1-42
 - PRINT TRANSACTION IMAGE 1-43
 - PROCESS NETWORK ACKS 1-29
 - PURGE 1-31
 - QUERY 1-47
 - QUERY PROFILE 1-64
 - RECEIVE 1-21
 - RECEIVE AND DEENVELOPE 1-25
 - RECEIVE AND TRANSLATE 1-26
 - RECONSTRUCT 1-28
 - RECONSTRUCT AND SEND 1-28
 - REENVELOPE 1-7
 - REENVELOPE AND SEND 1-12
 - RELEASE 1-31
 - REMOVE STATISTICS 1-38
 - REMOVE TRANSACTIONS 1-32
 - REPORT CONTINUOUS RECEIVE STATUS 1-66
 - RESTART SEND 1-11
 - RETRANSLATE TO APPLICATION 1-23
 - SEND 1-10

PERFORM commands (*continued*)

- SENDFILE 1-10
- START CONTINUOUS RECEIVE 1-65
- STOP CONTINUOUS RECEIVE 1-65
- TRADING PARTNER CAPABILITY DATA EXTRACT 1-52
- TRADING PARTNER PROFILE DATA EXTRACT 1-50
- TRANSACTION ACTIVITY DATA EXTRACT 1-54
- TRANSACTION DATA EXTRACT 1-59
- TRANSLATE AND ENVELOPE 1-11
- TRANSLATE AND SEND 1-15
- TRANSLATE TO APPLICATION 1-23
- TRANSLATE TO STANDARD 1-6
- UNPURGE 1-31
- UPDATE 1-47, 1-58
- UPDATE STATISTICS 1-38
- UPDATE STATUS 1-29

PERFORM statement 1-2

performance considerations G-1

performance monitor user exit 5-44

persistent environment trace 1-68

PL/I calls 3-6

point-to-point, connecting 6-38, 6-39

pre- and post-envelope programs 5-4

PRINT ACKNOWLEDGMENT IMAGE command

- description 1-41
- syntax 1-40

PRINT ACTIVITY SUMMARY command

- description 1-40
- syntax 1-40

PRINT commands 1-39

PRINT CUSTOM LAYOUT command

- description 1-46
- examples
- syntax 1-46

PRINT EVENT LOG command

- description 1-45
- syntax 1-40

print file (PRTFILE) 2-5

PRINT STATUS SUMMARY command

- description 1-43
- syntax 1-40

PRINT STATUS SUMMARY2 command

- description 1-44
- syntax 1-40

PRINT TRANSACTION DETAILS command

- description 1-42
- syntax 1-40

PRINT TRANSACTION IMAGE command

- description 1-43
- syntax 1-40

printing Transaction Store reports

- description 1-39

PRIORTO keyword 1-100

PROCESS NETWORK ACKS command

- description 1-29
- examples 1-30
- syntax 1-30

processing multiple incoming TSQs 5-4

profile definition 2-19

profile members, deleting 1-64

profile, querying 1-64

program errors B-25

program list table (PLT)

- continuous receive processing 5-41
- sample post-initialization 5-41
- sample pre-termination 5-41
- use with DB2 attachment DSNCCOM1 5-41
- use with program EDICRSP 5-41
- use with program EDICRTS 5-41
- use with program EDIXSOX 5-41
- use with program EXPOSTRT 5-41

program samples

- authentication exit routine D-85—D-95
- data extract report generators
 - using flat file (EDISAMR1) D-27—D-35
 - using SQIFILE (EDISAMR1) D-35—D-44
- encryption exit routine D-95—D-117
- ending DataInterchange D-26—D-27
- ending translation D-18—D-19
- field exit routine D-57—D-65
- filtration exit routines
 - ASCII D-72—D-78
 - ASCII/BAUDOT filter example D-78—D-85
 - hexadecimal D-68—D-72
- initializing and terminating DataInterchange
 - C D-5—D-9
 - COBOL D-1—D-3
 - PL/I D-3—D-5
- network activity D-44—D-49
- querying the transaction store
 - COBOL D-9—D-10
 - PL/I D-11
- receiving from network D-19—D-22
- sending queued data D-15—D-18
- test for filter type D-65—D-68
- translating and sending queued data D-12—D-15
- translating received data D-22—D-25

PRTFILE 2-5

PURGE command

- description 1-31
- examples 1-31
- syntax 1-30

purge interval, setting 1-100

purging documents from the store 1-32

PURGINT keyword 1-100

put envelope 4-14

- get envelope return codes 4-15
- put envelope return codes 4-15

Q

- QTYPE field 2-65
- QUERY command
 - description 1-47
 - examples 1-47
 - syntax 1-47
- query file (EDIQUERY) 2-7
- QUERY PROFILE command
 - description 1-64
 - examples 1-64
 - syntax 1-64
- querying
 - filename
 - CMCB initialization 3-124
 - description 3-123
 - information returned 3-124
 - transaction store
 - COBOL sample program D-9—D-10
 - PL/I sample program D-11
- queuing totals (Q) record 2-76

R

- raw data
 - record format 2-72
 - test transaction
- RAWDATA keyword 1-101
- RAWDATA mode 3-26, 3-40
- RAWFMTID keyword 1-101
- RAWTEST keyword 1-101
- rebuild command
 - See RECONSTRUCT AND SEND command 1-28
 - See RECONSTRUCT command 1-28
- RECEIVE AND DEENVELOPE command
 - description 1-25
 - examples 1-25
 - syntax 1-25
- RECEIVE AND TRANSLATE command
 - description 1-26
 - examples 1-27
 - syntax 1-27
- RECEIVE command
 - description 1-21
 - examples 1-21
 - syntax 1-21
- RECEIVEACKDATA keyword 1-103
- RECEIVEACKIMAGE keyword 1-103
- receiving
 - return codes B-27—B-29
 - sample program D-19
- receiving non-EDI data 1-21
- receiving transactions
 - CMCB initialization 3-118
 - description 3-117
 - information returned 3-119

- receiving transactions (*continued*)
 - network operation 3-119
- RECID field
 - control record 2-63
 - data record 2-70
- RECONSTRUCT AND SEND command
 - description 1-28
 - examples 1-28
 - syntax 1-28
- RECONSTRUCT command
 - description 1-28
 - examples 1-28
 - syntax 1-28
- record layouts
 - acknowledgment image data extract 2-87
 - application data extract 2-86
 - group data extract 2-83
 - interchange data extract 2-81
 - network activity data extract 2-78
 - trading partner capability data extract 2-77
 - trading partner profile data extract 2-76
 - transaction activity data extract 2-79
 - transaction data extract 2-83—2-86
 - transaction image data extract 2-87
 - transaction store common key data extract 2-81
- records
 - allocation of
 - as shipped F-24—F-29
 - calculating F-1—F-23
 - example F-29
 - control (C) 2-61
 - data (D) 2-70
 - end transaction/interchange (Z) 2-71
 - envelope header (E) 2-75
 - group header (G) 2-75
 - information (I) 2-74
 - optional 2-72
 - queuing totals (Q) 2-76
 - raw data 2-72
 - transaction set header (T) 2-76
- RECOVBAD keyword 1-103
- RECOVERY keyword 1-103
- RCVFILE command 1-21
- REENVELOPE AND SEND command
 - description 1-12
 - examples 1-14
 - syntax 1-14
- REENVELOPE command
 - description 1-7
 - examples 1-9
 - syntax 1-9
- RELEASE command
 - description 1-31
 - examples 1-31
 - syntax 1-30

- Release, level 3-97
- REMOVE STATISTICS command
 - description 1-38
 - examples 1-38
 - syntax 1-38
- REMOVE TRANSACTIONS command
 - description 1-32
 - syntax 1-33
- REPLACE field 2-13
- REPLACE keyword 1-104
- REPORT CONTINUOUS RECEIVE STATUS command
 - description 1-66
 - examples 1-67
 - syntax 1-67
- report file description 2-6
- reporting, management and transaction store 1-48
- reports
 - activity summary 1-41
 - management
- REQID keyword 1-104
- REQTP 1-12
- REQTP keyword 1-104
- requestor profile data block (REQDB)
 - copy books
 - ASSEMBLER C-44
 - COBOL C-10
 - field descriptions A-57—A-61
 - include files
 - C C-33
 - PL/I C-21
 - layout A-56
- requestor profile member (REQPROF) 2-49
- RESET keyword 1-104
- RESET STATISTICS 1-38
- response applications
 - continuous receive 5-28
 - DataInterchange Utility 5-27
 - description 5-27
 - getting results from DataInterchange 5-27
 - transaction
 - description 5-29
 - where to specify 5-31
 - types 5-27—5-31
 - ways of starting 5-27
- response programs, invoking D-56
- restart receive EDI data 1-22
- RESTART SEND command
 - description 1-11
 - example 1-11
 - syntax 1-11
- RETRANSLATE TO APPLICATION command
 - description 1-23
 - examples 1-24
 - syntax 1-23
- retrieving
 - functional acknowledgment image 3-105
 - group header 3-100
 - interchange header 3-99
 - transaction acknowledgment image 3-105
 - transaction header 3-100
 - transaction image 3-104
- return codes 4-1, B-1
 - API B-17
 - combined commands
 - description B-16
 - ENVELOPE AND SEND command B-17
 - RECEIVE combinations B-17
 - REENVELOPE AND SEND command B-17
 - TRANSLATE AND SEND command B-16
 - COMMIT work request B-29
 - communication B-27—B-29
 - categories B-27
 - nonsevere errors B-28
 - severe errors B-28
 - warnings B-28
 - communication services 3-109
 - data element errors B-20
 - DataInterchange Utility B-17
 - ending DataInterchange B-18
 - environmental errors B-25
 - group errors B-23
 - initialize SYNC B-29
 - initializing DataInterchange B-17
 - interchange errors B-24
 - invalid data errors B-25
 - JCL job step condition codes B-17
 - negative values B-18
 - normal conditions B-19
 - program B-25
 - receiving B-27—B-29
 - ROLLBACK work request B-30
 - segment errors B-20
 - sending B-27—B-29
 - status update B-29
 - transaction errors B-21
 - translation B-4, B-18—B-27
 - receiving B-6
 - sending B-4
 - updating status 3-128
 - UTILCCODE B-17
 - UTILSEV B-17
 - warning conditions B-19
- return information
 - cancel API 3-122
 - receive API 3-119
 - return filename API 3-124
 - send files API 3-116
 - send transactions API 3-113

ROLLBACK function 3-135

return codes B-30

RPTFILE 2-6

S

sample JCL

archive (EDIELARV)

changes from previous release

data sets for DataInterchange Utility E-17

DataInterchange Utility (EDIUTILV)

description E-1

modifications for DB2

restore JCL

sample programs

authentication exit routine D-85—D-95

data extract report generators

using flat file (EDISAMR1) D-27—D-35

using SQIFILE (EDISAMR1) D-35—D-44

encryption exit routine D-95—D-117

ending DataInterchange D-26—D-27

ending translation D-18—D-19

field exit routine D-57—D-65

filtration exit routines

ASCII D-72—D-78

ASCII/BAUDOT filter example D-78—D-85

hexadecimal D-68—D-72

initializing and terminating DataInterchange

C D-5—D-9

COBOL D-1—D-3

PL/I D-3—D-5

network activity D-44—D-49

querying the transaction store

COBOL D-9—D-10

PL/I D-11

receiving from network D-19—D-22

sending queued data D-15—D-18

test for filter type D-65—D-68

translating and sending queued data D-12—D-15

translating received data D-22—D-25

SAP status tracking in DataInterchange 6-34

SAPSTAT keyword 1-104

SAPUPDT keyword 1-105

SCOPE 3-42

SCRIPT keyword 1-105

script name 6-40

SDM LinkPlus 6-40

security exit routines

authentication

description 4-22

parameters 4-23—4-26

buffer size 4-18

compression

description 4-26

parameters 4-27—4-30

description 4-15

security exit routines (*continued*)

enabling during

receive 4-17

send 4-17

encryption

description 4-18

parameters 4-19—4-22

filtering

description 4-26

parameters 4-27—4-30

parameters introduction 4-18

supporting routines 4-30—4-32

call exit 4-31

get data 4-30

put data 4-31

security profile data block (SPDB)

copy books

ASSEMBLER C-46

COBOL C-12

include files

C C-34

PL/I C-23

security profile member (SECUPROF) 2-49

segment errors B-20

SEGMENTED keyword 1-105

segments, envelope 3-70

SEND command

description 1-10

examples 1-10

syntax 1-10

send/receive program 6-38

SENDACKDATA keyword 1-2, 1-105

SENDACKIMAGE keyword 1-106

SENDFILE command

description 1-10

example 1-10

syntax 1-10

sending

return codes B-27—B-29

sample program D-15

sending files

CMCB initialization 3-115

description 3-115

information returned 3-116

sending transactions

CMCB initialization 3-110

description 3-109

information returned 3-113

message name, default 3-114

message user class, default 3-114

network operation 3-112

sent to network status 5-26

service name block (SNB)

API function 3-1

copy books

ASSEMBLER C-37

COBOL C-3

- service name block (SNB) (*continued*)
 - description A-1
 - field descriptions A-1—A-2
 - include files
 - C C-25
 - PL/I C-14
 - initializing for
 - data extract 3-101
 - de-envelope API 3-87
 - envelope API 3-74
 - translate to application API 3-58
 - translate to standard API 3-28
 - layout A-1
- services
 - communication 3-105—3-125
 - data extraction 3-101—3-105
 - enveloping 3-69—3-100
 - environmental 3-17—3-19
 - overview 3-13—3-17
 - status update 3-126—3-131
 - syncpoint 3-132—3-136
 - translation 3-20—3-69
- SERVICESEGVAL keyword 1-106
- setup considerations, DataInterchange Utility 5-11
- setup information, exporting/importing 1-62
- single unit of work mode 3-25, 3-68
- SNDDATE keyword 1-106
- SNDDTIME keyword 1-107
- space allocation for records
 - as shipped F-24—F-29
 - calculating F-1—F-23
 - example F-29
- STANDALONE keyword 1-107
- standard segment usage 2-36
- standard transaction definition 2-35
- standard/envelope data element definition 2-37
- standard/envelope data element usage 2-36
- standard/envelope definition 2-35
- standard/envelope segment definition 2-36
- START CONTINUOUS RECEIVE command
 - description 1-65
 - examples 1-65
 - syntax 1-65
- statistics
 - communication by requestor
 - receive usage key
 - remove
 - send usage key
 - update
- status
 - printing
 - summary 1-43
 - summary2 1-44
 - transactions, requesting
- status update services
 - description 3-126
 - overview 3-16
 - return codes B-29
- STDDESC keyword 1-108
- STDID keyword 1-108
- STDLV keyword 1-108
- STDTRID keyword 1-108
- STDVVR keyword 1-108
- STOP CONTINUOUS RECEIVE command 1-65
 - description 1-65
 - examples 1-65
 - syntax 1-65
- store time, setting 1-100
- STSTAT keyword 1-108
- switch APPLID A-5
- SYNCPOINT services
 - description 3-132
 - initialize SYNC
 - API request format 3-134, 3-136
 - return codes 3-134
 - overview 3-16
 - work requests
 - COMMIT 3-135
 - ROLLBACK 3-135
- SYNCSERV 3-16
- syntax 1-67, 1-68
 - command language 1-1
 - perform commands 1-1
- SYSID 1-116
- SYSIN 2-1
- system level 6-39
- system profile Member (SYSPROF) 2-58
- system type 6-39
- SYSTSIN 2-2

T

- table definition 2-46
- table definition field descriptions 2-46
- table entry 2-48
- table entry field descriptions 2-48
- tagged import files, creating D-1
- temporary storage queues, for DataInterchange Utility 5-33—5-35
- terminating environmental services 3-19
- TEST 3-31
- TESTIND field 2-63
- testing
 - translate to application 3-49
 - translate to standard 3-25
- TESTMODE keyword 1-109
- THANDLE 3-20
- timeout (REQDB) A-56, A-61
- timeout (TPPDB) 3-108, A-43, A-50

- TP contacts definition (7P3) 2-32
- TPDATALINE (TPPDB) 3-108, A-43, A-50
- TPID keyword 1-109
- TPNICKN keyword 1-110
- TPNICKNESEND keyword 1-111
- TPTID keyword 1-111
- TRABORT 3-34
- trace, GLB 1-68
- tracking file (FFSTRAK) 2-5
- TRADING PARTNER CAPABILITY DATA EXTRACT
 - command
 - description
 - examples 1-52, 1-53
 - syntax 1-52
- trading partner mapping data element 2-42
- trading partner mapping rules 2-42
- trading partner mapping segment 2-41
- trading partner profile data block (TPPDB)
 - copy books
 - ASSEMBLER C-43
 - COBOL C-9
 - field descriptions A-44—A-52
 - function in communication services 3-107
 - include files
 - C C-31
 - PL/I C-20
 - layout A-42
- TRADING PARTNER PROFILE DATA EXTRACT
 - command
 - description
 - examples 1-51
 - syntax 1-50
- trading partner profile member (TPPROF) 2-19
- trading partner profile member field descriptions 2-22
- trading partner receive usage 2-44
- trading partner send usage 2-43
- trading partner transaction 2-40
- TRANPROC 3-14, 3-20
- TRANRC field 2-63
- transaction
 - acknowledgment image, retrieving 3-105
 - activity report 1-40
 - de-enveloping 3-90
 - details, printing 1-42
 - header, retrieving 3-100
 - image
 - printing 1-43
 - retrieving 3-104
 - layer 3-72
 - receiving 3-117
 - sending 3-109
 - status, requesting 1-29
- TRANSACTION ACTIVITY DATA EXTRACT command
 - description
 - examples 1-54
 - syntax 1-54
- TRANSACTION DATA EXTRACT command
 - description
 - examples 1-60
 - syntax 1-59
- transaction errors B-21
- transaction exit routines
 - description 4-9
 - parameters
 - Assembler definition 4-13
 - C definition 4-12
 - COBOL definition 4-11
 - description 4-10
 - post-translation 4-9
 - pre-translation 4-9
- TRANSACTION keyword 1-111
- transaction response application
 - description 5-29
 - where to specify 5-31
- transaction set header (T) record 2-76
- transaction store query performance techniques G-4
- transaction store, reporting using 1-48
- transactions, supplied by DataInterchange Utility 5-43
- transient data queues, for DataInterchange Utility 5-33—5-35
- TRANSLATE AND ENVELOPE command
 - description 1-11
 - examples 1-12
 - syntax 1-12
- TRANSLATE AND SEND command
 - description 1-15
 - examples 1-15
 - limitations 1-15
 - syntax 1-15
- translate files
 - calls to translator
 - first for transaction 3-61
 - first of session 3-58
 - last for transaction 3-68
 - last of session 3-68
 - other than first or last 3-67
 - description 3-57
- translate to application
 - API
 - description 3-49
 - translate files 3-57
 - translate specific transactions 3-49
 - calls to translator
 - first for transaction 3-52
 - first of session 3-50
 - last for transaction 3-57
 - last of session 3-57
 - other than first or last 3-56
 - de-enveloping transactions 3-46
 - description 3-45
 - illustration 3-48
 - receiving transactions 3-46

translate to application (*continued*)

- special considerations
 - clustered transactions 3-69
 - partial structures 3-68
 - single unit of work 3-68
- testing 3-49

TRANSLATE TO APPLICATION command

- description 1-23
- examples 1-24
- syntax 1-24

translate to standard

- API description 3-27
- calls to translator
 - first for transaction 3-30
 - first of session 3-28
 - last for transaction 3-36
 - last of session 3-39
 - other than first or last 3-36

data modes 3-25

description 3-22

enveloping 3-22

during translation 3-35

TRCB fields 3-38

illustration 3-24

initializing

- FCB 3-28
- SNB 3-28
- TRCB 3-28—3-30, 3-31—3-34
- TRIDB 3-34
- TRODB 3-30

overview 3-14

sending 3-22

special considerations 3-40—3-44

- batches and bundles 3-43
- forced interchanges 3-43
- partial structures 3-41
- raw data 3-40
- recovery scope 3-42

testing 3-25

TRCB fields returned

- after translation 3-37
- first call for transaction 3-35
- interchange header/trailer 3-55
- when interchange written 3-35

TRANSLATE TO STANDARD command

- description 1-6
- examples 1-7
- syntax 1-7

translation

- return codes B-4—B-27
- sample programs
 - receiving D-22
 - sending D-12

translation services

- database tables 3-20
- description 3-20

translation services (*continued*)

- major functions 3-21
- minor functions 3-21
- overview
- translate to application 3-45—3-69
- translate to standard 3-22
- translating files 3-57—3-68

translator control block (TRCB)

copy books

- ASSEMBLER C-39
- COBOL C-5

field descriptions A-10—A-28

include files

- C C-27
- PL/I C-16

initializing for

- data extract 3-102
- de-envelope API 3-87
- envelope API 3-74
- translate to application API 3-58
- translate to standard API 3-50

layout A-6

translator input data block (TRIDB)

description A-29

field descriptions A-31

initializing for

- data extract 3-102
- de-envelope API 3-90
- envelope API 3-76
- translate to application API 3-60
- translate to standard API 3-51

layout A-29

translator output data block (TRODB)

description A-31

field descriptions A-33

initializing for

- data extract 3-102
- de-envelope API 3-90
- envelope API 3-76
- translate to application API 3-60
- translate to standard API 3-51

layout A-31

TRANSSRV 3-16

TRANXRC field 2-63

TRCB return fields

application

- de-envelope API 3-92
- envelope API 3-77

functional acknowledgment 3-92

group header/trailer

- de-envelope API 3-96
- envelope API 3-79

interchange header/trailer

- de-envelope API 3-92
- envelope API 3-78

interchange written

- close and queue interchange API 3-85

- TRCB return fields (*continued*)
 - interchange written (*continued*)
 - de-envelope API
 - envelope API 3-80
 - transaction attributes
 - de-envelope API 3-91
 - envelope API 3-77
 - transaction header/trailer
 - de-envelope API 3-96
 - envelope API 3-80
- TREL field 2-67
- TRERLVL keyword 1-111
- TRNSTAT 3-41
- TRXACCEPT 3-34
- TRXCTLNO keyword 1-112
- TRXDATE keyword 1-112
- TRXLIFE keyword 3-89
- TRXSTAT keyword 1-113
- TRXTIME keyword 1-114
- TS queues, processing multiple 5-4
- TSKEY 3-21, 3-23
- TSKEYU 3-21, 3-23
- TTYTYPE field 2-65
- TVER field 2-67

U

- UCS profile member (U) 2-56
- unit of work
 - DataInterchange Utility and application 5-10
 - description 5-7
 - ensuring for DataInterchange Utility 5-10
- UNPURGE command
 - description 1-31
 - examples 1-32
 - syntax 1-30
- UNTDI profile member (T) 2-55
- UPDATE STATISTICS command
 - description 1-38
 - examples 1-38
 - syntax 1-38
- UPDATE STATUS command
 - description 1-29
 - examples 1-29
 - syntax 1-29
- updating
 - network status 1-29
 - statistics 1-38
- updating status
 - alternate key formats
 - account number/user ID 3-130
 - description 3-130
 - interchange qualifier/ID 3-130
 - API 3-126
 - data block 3-131
 - description 3-126

- updating status (*continued*)
 - envelope key formats
 - account number/user ID 3-129
 - description 3-128
 - interchange qualifier/ID 3-129
 - trading partner nickname 3-128
 - transaction handle 3-129
 - return codes 3-128
- USAGETID field 2-15
- user exit, performance monitor 5-44
- user field in continuous receive profile
- user program information profile member (ADAMCTL) 2-52
- USERID keyword 1-115
- USERPGM keyword 1-115
- USERSYNC field 5-22
- UTILCB 3-18
- utility control information 4-13, 4-33
- utility JCL for DB2 2-2
- UTILRC field 2-63

V

- validating command language input 1-4
- VANICICS network program example D-122
- variables
- VERIFY keyword 1-115
- Version, level 3-97

W

- warning conditions, return codes B-19
- work file (FFSWORK) 1-2, 2-8

X

- X12 profile member (X) 2-56
- XPANDED field 2-64

Z

- Z record
 - description 2-71
 - format 2-72

Communicating Your Comments to IBM

DataInterchange
Programmer's Reference
Version 3 Release 1
Publication No. SB34-2001-04

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
United States & Canada: 813+878-5475
- If you prefer to send comments electronically, use this network ID:
IBM Mail Exchange: USIB2PH9 at IBMMAIL

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

DataInterchange
Programmer's Reference
Version 3 Release 1
Publication No. SB34-2001-04

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Global Services
Department RD10
PO BOX 20143
TAMPA FL 33633-0872



Fold and Tape

Please do not staple

Fold and Tape



File Number: MVS-79

Program Number: 5655-B29 and 5655-B30

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SB34-2001-04

