

# **Component Broker for Windows NT and AIX CICS and IMS Application Adaptor Quick Beginnings Release 2.0**

Document Number GC09-2703-03

December 16, 1998

Owners:  
IBM Corporation, <http://www.ibm.com>

Component Broker Home Page, <http://www.software.ibm.com/ad/cb/>



Component Broker for Windows NT and AIX

GC09-2703-03

## **CICS and IMS Application Adaptor Quick Beginnings**

Release 2.0





Component Broker for Windows NT and AIX

GC09-2703-03

## **CICS and IMS Application Adaptor Quick Beginnings**

Release 2.0

**Note**

Before using this information and the product it supports, be sure to read the general information under Appendix E, "Notices" on page 229.

**Third Edition (December 1998)**

This edition applies to Release 2.0 of Component Broker and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 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

<b>About This Book</b> .....	xi
Who Should Read This Book .....	xi
How This Book is Organized .....	xi
Documentation Conventions .....	xii
The Component Broker Documentation .....	xiii
<b>Chapter 1. Introduction</b> .....	1
The Procedural Application Adaptor .....	1
What PAA Does .....	1
How Customers Use PAA .....	2
The Elements of PAA .....	2
How the Parts of PAA Work Together with Component Broker .....	3
Component Broker Flows for a Pessimistic APPC Connection .....	8
Component Broker Flows for an Optimistic APPC Connection .....	9
Developing a PAA Application .....	11
Analyzing the Existing CICS/IMS Transactions .....	12
Designing the Objects .....	13
Creating the PAO .....	13
Parsing the Definitions .....	14
Creating a Mapper .....	15
Creating the Commands and Navigators .....	16
Modifying the PAO CRUD Methods to Call Commands and Navigators .....	18
Unit Testing in the VisualAge for Java Environment .....	19
Importing Into Object Builder .....	19
Push Down Methods .....	20
CICS and IMS Overview .....	20
Definitions .....	22
<b>Chapter 2. Planning the Install</b> .....	25
Packaging .....	25
System Requirements .....	25
Prerequisites .....	25
The Component Broker Package .....	26
Installation Considerations .....	26
Differences Between VisualAge for Java for Component Broker 1.3 and 2.0 .....	27
General Differences .....	27
<b>Chapter 3. Installing the CICS and IMS Application Adaptor on Windows NT</b> .....	31
Installing the Prerequisite Software .....	31
Verifying the Installation of the Component Broker Run Time .....	31
Installing the Communications Server .....	32
Configuring the Communications Server .....	34
Setting Up the Remote Node .....	34
Gathering Your Local Node Configuration Information .....	34
Configuring the Local Node .....	36
Installing the CICS Transaction Gateway .....	38
Configuring the CICS Client Within the Transaction Gateway .....	39
Installing the Run Time .....	40
Pre-Installation .....	40
Installation .....	40

Configuring the CICS and IMS Application Adaptor	41
Starting the Transaction Gateway	42
Reverting to the Original Install State	42
Uninstalling Communications Server for Windows NT	42
Uninstalling the CICS and IMS Application Adaptor	43
<b>Chapter 4. Installing the CICS and IMS Application Adaptor on AIX</b>	<b>45</b>
Installing the Prerequisite Software	45
Installing VisualAge for Java 2.0 for AIX	45
Configuring VisualAge for Java 2.0 for AIX	46
Installing the Communications Server	47
Installing the Software Bundle Definitions	47
Installing the Software	47
Installing the Communications Adaptor Support	48
Configuring the Communications Server	48
Configuring the Node Parameters	49
Defining a Connection	49
Defining the Partner LU	50
Installing and Configuring the CICS Transaction Gateway	50
Installing the CICS and IMS Application Adaptor	51
Configuring the CICS and IMS Application Adaptor	52
Installing the Development Environment for the CICS and IMS Application Adaptor	52
Uninstalling the CICS and IMS Application Adaptor	53
Environment Setup	53
<b>Chapter 5. Developing an IMS-HOD Application</b>	<b>55</b>
The IMS Sample Application	55
Interacting with the IMS IVP	56
PhoneBookEntry Object Model	57
Enterprise Access Builder Procedures	58
Importing Pre-requisite Features into the Workspace	58
Creating a Project/Package under VisualAge for Java	59
Creating the Procedural Adaptor Object and Key	59
PhoneBookPAOKey	61
PhoneBookPAO	61
Importing the tele.mfs File	62
Creating the Record Mapper	62
Creating the SingleLine Record Type and Record Bean	63
Creating the Command Beans	67
Creating the CmdBaseToMenu Command	68
Command Bean Summary	68
Creating the CmdMenuToClear Command	70
Creating the CmdClearToBase Command	70
Creating the CmdFirstToSecondSignon Command	71
Creating the CmdSecondSignonToBase Command	71
Creating the CmdMenuToMenuDisplay Command	72
Creating the CmdMenuToMenuAddUpdt Command	74
Creating the CmdMenuToMenuDel Command	75
Creating Logoff/Logon/Class	76
Creating Navigator Beans	77
Creating the NavigatorRetrieve Navigator	77
Creating the NavigatorAddUpdate Navigator	79
Creating the NavigatorDel Navigator	80
Creating the NavigatorSignon Navigator	81

Using the Navigators	82
Creating and Editing the PBELogonLogoff Method	82
Editing the PhoneBookPAO::retrieve method	83
Editing the PhoneBookPAO::del method	83
Editing the PhoneBookPAO::insert method	84
Editing the PhoneBookPAO::update method	84
Creating an Executable Class	85
Exporting the PBE Package	87
Developing an IMS-HOD Business Object	88
Importing the Bean	88
Defining the PhoneBookEntry Component	89
Creating the Business Object File	89
Defining the Business Object	89
Connecting the Data Object Implementation to the Persistent Object	91
Defining the Managed Object	93
Generating the Code	93
Creating Client and Server DLL Files	93
Defining the Client DLL File	93
Defining the Server DLL File	94
Building the DLL Files	94
Packaging the Application	94
Creating the Application Family	94
Defining the Application	95
Creating the Container Instance	95
Configuring the Managed Object	95
Generating the Applications	96
Building the Application - Client and Server	96
Building the Application - Client and Server	96
Installing the Application	97
Loading the Application onto System Management	97
Configuring the Application with System Management	97
Running the Sample Application	99
<b>Chapter 6. Developing a CICS-HOD Application</b>	<b>101</b>
The CICS Sample Application	101
Interacting with the CICS IVP	102
Enterprise Access Builder Procedures	103
Creating a Project and Package for the Samples	103
Creating the Procedural Adapter Object and Key	104
Creating the MenuCustomer Class	104
Adding Properties to the MenuCustomer Class	104
Creating the MenuCustomerKey Class	105
Adding Properties to the MenuCustomerKey Class	105
Linking the PAO and its Key Class	106
Creating Record Beans and a Record Mapper	106
Creating the DFHDGA Record Type and Record Bean	107
Creating the DFHDGB Record Type and Record Bean	108
Creating the Record Mapper	109
Create the SingleLine Record Type and Record Bean	109
Verifying the Project Contents	110
Creating Command Beans	111
Creating the CmdBaseToMenu Command	111
Command Bean Summary	112
Creating the CmdMenuToBase Command	114

Creating the CmdMenuToListing Command	114
Adding Features to the CmdMenuToListing Command	115
Creating the CmdListingToMenu Command	116
Creating the CmdMenuToListingAddUpdt Command	117
Adding Features to the CmdMenuToListingAddUpdt Command	117
Creating the CmdListingToMenuAddUpdt Command	118
Creating the CmdMenuToMenuDelDebit Command	118
Adding Features to the CmdMenuToMenuDelDebit Command	119
Creating Navigator Beans	120
Creating the NavigatorRetrieve Navigator	120
Creating the NavigatorAddUpdate Navigator	122
Creating the NavigatorDelDebit Navigator	123
Usng the Navigators	125
Editing the MenuCustomer::debit method	125
Editing the MenuCustomer::del method	125
Editing the MenuCustomer::insert method	126
Editing the MenuCustomer::retrieve method	126
Editing the MenuCustomer::update method	127
Unit Testing the EAB Object	127
Run the Unit Test Main Method	130
Exporting the MenuCustomer Package	131
Developing a CICS-HOD Business Object	132
Importing the Bean	132
Defining the Acct Component	133
Creating the Business Object File	133
Creating the Business Object	133
Connecting the Data Object Implementation to the Persistent Object	136
Defining the Managed Object	137
Generating the Code	137
Creating Client and Server DLL Files	137
Defining the Client DLL File	137
Defining the Server DLL File	138
Generating the Makefiles	138
Packaging the Application	138
Creating the Application Family	138
Defining the Application	139
Creating the Container Instance	139
Configuring the Managed Object	139
Generating the Application	140
Building the Application - Client and Server	140
Installing the Application	141
Loading the Application onto System Management	141
Configuring the Application with System Management	141
Running the Sample Application	142
<b>Chapter 7. Developing a CICS-ECI Application</b>	<b>145</b>
The CICS-ECI Sample Application	145
Developing an Application to Retrieve Cash Account Information from a CICS Server	146
Importing Prerequisite Features into the Workspace	147
Creating a Project and Package Under VisualAge for Java	148
Creating the Procedural Adaptor Object and Key	148
BeCashAcctPAOKey	150
BeCashAcctPAO	150
Importing the Customer COBOL File	150

Creating the Record Mapper	151
Creating the BeCashAcctCommand Class	152
Modifying the Procedural Adaptor Object to Call the Commands	154
Modifying the Procedural Adapter Object to connect to the CICS Server	155
Creating an Executable Class	156
Running the Customer Command Application	158
Exporting the BeCashAcct Package	158
Developing a CICS-ECI Business Object	159
Importing the Bean	159
Defining the CashAcct Component	160
Creating the Business Object File	160
Defining the Business Object	160
Connecting the Data Object Implementation to the Persistent Object	162
Defining the Managed Object	163
Generating the Code	163
Creating Client and Server DLL Files	164
Defining the Client DLL File	164
Defining the Server DLL File	164
Generating the Makefiles	165
Packaging the Application	165
Creating the Application Family	165
Defining the Application	165
Creating the Container Instance	165
Configuring the Managed Object	166
Generating the Applications	166
Building the Application - Client and Server	167
Installing the Application	167
Loading the Application onto System Management	167
Configuring the Application with System Management	167
Running the Sample Application	170
<b>Chapter 8. Developing an IMS-APPC Application</b>	<b>171</b>
The IMS Sample Application	171
Enterprise Access Builder Procedures	171
Importing Prerequisite Features into the Workspace	172
Creating a Project/Package under VisualAge for Java	172
Creating the Procedural Adaptor Object and Key	173
APhoneBookPAOKey	174
APhoneBookPAO	174
Importing the PhoneBook COBOL File	175
Creating the Record Mapper	176
Creating the Command Classes	178
Modifying the Procedural Adapter Object to call the Commands	181
Exporting the pbe Package	182
Developing an IMS-APPC Business Object	183
Importing the Bean	184
Defining the PhoneBookRec Component	184
Creating the Business Object File	184
Defining the Business Object	185
Connecting the Data Object Implementation to the Persistent Object	187
Defining the Managed Object	188
Generating the Code	188
Creating Client and Server DLL Files	188
Defining the Client DLL File	188

Defining the Server DLL File . . . . .	189
Building the DLL Files . . . . .	189
Packaging the Application . . . . .	189
Creating the Application Family . . . . .	189
Defining the Application . . . . .	190
Creating the Container Instance . . . . .	190
Configuring the Managed Object . . . . .	190
Generating the Applications . . . . .	191
Building the Application - Client and Server . . . . .	191
Installing the Application . . . . .	192
Loading the Application onto System Management . . . . .	192
Configuring the Application with System Management . . . . .	192
Running the Sample Application . . . . .	194
<b>Chapter 9. Developing a CICS-APPC Application . . . . .</b>	<b>195</b>
The CICS-APPC Sample Application . . . . .	195
Enterprise Access Builder Procedures . . . . .	196
Importing Pre-requisite Features into the Workspace . . . . .	196
Creating a project/package under VisualAge for Java . . . . .	197
Creating the Procedural Adaptor Object and Key . . . . .	197
ABeCashAcctPAOKey . . . . .	199
ABeCashAcctPAO . . . . .	199
Importing the Customer COBOL File . . . . .	199
Creating the Record Mapper . . . . .	200
Creating the ABeCashAcctCommand Class . . . . .	201
Inbound Side of Command . . . . .	203
Outbound Side of Command . . . . .	203
Modifying the Procedural Adaptor Object to call the Commands . . . . .	204
Exporting the ABeCashAcct Package . . . . .	205
Developing a CICS-APPC Business Object . . . . .	205
Importing the Bean . . . . .	206
Defining the ACashAcct Component . . . . .	207
Creating the Business Object File . . . . .	207
Defining the Business Object . . . . .	207
Connecting the Data Object Implementation to the Persistent Object . . . . .	209
Defining the Managed Object . . . . .	210
Generating the Code . . . . .	210
Creating Client and Server DLL Files . . . . .	210
Defining the Client DLL File . . . . .	211
Defining the Server DLL File . . . . .	211
Generating the Makefiles . . . . .	211
Packaging the Application . . . . .	212
Creating the Application Family . . . . .	212
Defining the Application . . . . .	212
Creating the Container Instance . . . . .	212
Configuring the Managed Object . . . . .	213
Generating the Applications . . . . .	213
Building the Application - Client and Server . . . . .	213
Installing the Application . . . . .	214
Loading the Application onto System Management . . . . .	214
Configuring the Application with System Management . . . . .	214
Running the Sample Application . . . . .	217
<b>Appendix A. Installing the IVPs and CICS-ECI Sample . . . . .</b>	<b>219</b>

IVP Install Instructions . . . . .	219
CICS-ECI Sample Install Instructions . . . . .	219
Content of the t3-trans Directory . . . . .	220
<b>Appendix B. Installing the CICS DTP Sample Programs . . . . .</b>	<b>223</b>
Installing on CICS/ESA . . . . .	223
Installing on a CICS Transaction Server (NT or AIX) . . . . .	223
<b>Appendix C. Interchange Files within VisualAge for Java . . . . .</b>	<b>225</b>
<b>Appendix D. IMS Configuration . . . . .</b>	<b>227</b>
<b>Appendix E. Notices . . . . .</b>	<b>229</b>
Trademarks . . . . .	230



---

## About This Book

The *Component Broker for Windows NT and AIX CICS and IMS Application Adaptor Quick Beginnings* provides a brief technical overview of the CICS and IMS application adaptor and provides information on how to:

- Install and configure the CICS and IMS application adaptor portion of Component Broker with its prerequisite software.
- Write your first CICS and IMS applications using application development tools provided on the CBToolkit compact disc.

Do not use either the *Component Broker for Windows NT and AIX CICS and IMS Application Adaptor Quick Beginnings* or the *Component Broker for Windows NT and AIX Quick Beginnings* as a substitute for the Component Broker library. The library provides detailed information beyond the concepts introduced in this book. Before installing Component Broker, read the README file, located in the CBConnector compact disc root directory, for last minute product information. After completing the tasks and exercises in this book, review the rest of the product library.

---

## Who Should Read This Book

The *Component Broker for Windows NT and AIX CICS and IMS Application Adaptor Quick Beginnings* is intended for application programmers who want to:

- Understand the basics of the CICS and IMS application adaptor run-time.
- Plan an initial installation of the CICS and IMS run-time on top of an existing Component Broker install.
- Install the CICS and IMS application adaptor portion of Component Broker.
- Develop basic applications for IMS and CICS backend systems.

---

## How This Book is Organized

Chapter 1, "Introduction" on page 1 provides a introduction to the functions and features of the Component Broker CICS and IMS application adaptor as they apply to this document.

Chapter 2, "Planning the Install" on page 25 describes the required system environment and software prerequisites for installing the CICS and IMS application adaptor run-time environment. This chapter describes which installable units are part of this package.

Chapter 3, "Installing the CICS and IMS Application Adaptor on Windows NT" on page 31 provides the procedures for installing, configuring, and uninstalling the CICS and IMS application adaptor portion of Component Broker and its prerequisite software.

Chapter 5, "Developing an IMS-HOD Application" on page 55 provides an end-to-end sample application using the principles of IMS and Host On-Demand (HOD).

Chapter 6, "Developing a CICS-HOD Application" on page 101 provides an end-to-end sample application using the principles of CICS and Host On-Demand (HOD).

Chapter 7, "Developing a CICS-ECI Application" on page 145 provides an end-to-end sample application using the principles of CICS and its external call interface (ECI).

Chapter 8, “Developing an IMS-APPC Application” on page 171 provides an end-to-end sample application using the principles of IMS and Advanced Program-to-Program Communication (APPC).

Chapter 9, “Developing a CICS-APPC Application” on page 195 provides an end-to-end sample application using the principles of CICS and Advanced Program-to-Program Communication (APPC).

Appendix A, “Installing the IVPs and CICS-ECI Sample” on page 219 provides an instruction for installing the CICS ECI and HOD Transaction Server and the Encina shared file system (SFS) required to run the CICS-ECI sample.

Appendix B, “Installing the CICS DTP Sample Programs” on page 223 provides procedures to install, set up, and configure the two CICS DTP sample programs on a CICS region.

Appendix C, “Interchange Files within VisualAge for Java” on page 225 provides details on exporting the .class and .java files from one version of VisualAge for Java to another version.

---

## Documentation Conventions

The following conventions distinguish different text elements.

plain	Window titles, folder names, icon names, and method names.
monospace	Programming examples, user input at the command line prompt or into an entry field, directory paths, and user output.
<b>bold</b>	Menu choices and menu names, labels for push buttons, check boxes, radio buttons, group-box controls, drop-down list boxes, combo-boxes, notebook tabs, and entry fields.
<i>italics</i>	Programming keywords and variables, titles of documents, and initial use of terms that are in the glossary.

The following short cut conventions are used to abbreviate menu selections and object expansions within tree views of the System Manager User Interface and Object Builder.

- The right arrow when used within a menu shows a series of menu selections. For example, “**File** → **New**” is translated to mean: “On the **File** menu, click **New**”.

The right arrow when used within a tree view shows a series of folder (or object) expansions. For example, “Expand Management Zones → Sample Cell and Work Group Zone → Configuration” is translated to mean:

1. Expand Management Zones.
2. Expand Sample Cell and Work Group Zone.
3. Expand Configuration.

**Note:** An object in a view can be expanded when there is a plus sign (+) beside that object. After an object is expanded, the plus sign is replaced by a minus (-) sign.

- + Expands a tree structure to show more objects. To expand, click the plus sign (+) beside any object. If you double-click the object, a new tree structure is displayed with that object as the root of the tree.
- Collapses a tree structure to review from view its containing objects. To collapse, click the minus sign (-) beside any object.

### left mouse button

Used for all actions in the application except for opening the pop-up menu of an object. For example, if you click with the left mouse button on an object, it is selected. Click with the left mouse button on a menu option to perform that action.

### right mouse button

Opens the pop-up menu of an object that contains a list of actions that can be performed on that object. The list varies depending on the type of object.

WIN

Denotes a section that applies only to the Windows 95 or Windows NT platform. Do not interpret this symbol to denote that an equivalent AIX section exists.

**Note:** The Windows 95 platform only supports the Component Broker Java client.

AIX

Denotes a section that applies only to the AIX platform. Do not interpret this symbol to denote that an equivalent Windows section exists.

---

## The Component Broker Documentation

The following information is part of Component Broker:

- Help information is available from Component Broker product panels.
- The Component Broker online library can be viewed using a frames-compatible Web browser.
- *Component Broker for Windows NT and AIX Quick Beginnings*, G04L-2375 explains how to easily create and verify a starter Component Broker environment. These instructions walk the user through a typical server and client installation. Users can extend this configuration using the information in the *Component Broker for Windows NT and AIX Planning, Performance, and Installation Guide*.
- *Component Broker for Windows NT and AIX Planning, Performance, and Installation Guide*, SC09-2798 provides a comprehensive overview of the Component Broker environment, then guides the user through planning considerations including capacity planning, performance tuning, prerequisites, and migration. It also leads the user through installation options for all Component Broker environments.
- *Component Broker for Windows NT and AIX Oracle Application Adaptor Quick Beginnings*, GC09-2733 provides a brief technical overview of the Oracle application adaptor and guides the user through its installation and configuration. Step-by-step instructions guide the user through creating an initial Oracle application using application development tools included in the CBToolkit package.
- *Component Broker for Windows NT and AIX System Administration Guide*, SC09-2704 provides information about configuring and operating one or more hosts managed by Component Broker. It also provides general information about using the System Manager User Interface.
- *Component Broker for Windows NT and AIX Application Development Tools*, SC09-2705 explains how to create and test Component Broker applications using the tools provided in the CBToolkit with a focus on common development scenarios such as inheritance and team development.
- *Component Broker Programming Guide*, G04L-2376 describes the programming model including business objects, data objects, and information about MOFW, IDL, and C++ CORBA programming.
- *Component Broker Advanced Programming Guide*, SC09-2708 describes the Component Broker implementation for the CORBA Object Services and the Component Broker Object Request Broker (including remote method invocation and the Dynamic Invocation Interface (DII) procedures), Session Service, Cache Service, Notification Service, Interlanguage Object Model (IOM), and work-load management (WLM).
- *Component Broker Programming Reference*, SC09-2810 contains information about the APIs available to Component Broker application developers.
- *Component Broker for Windows NT and AIX Problem Determination Guide*, SC09-2799 explains how to identify and resolve problems within a Component Broker environment using the tools provided with

Component Broker. The book includes information on installation problems, run time errors, debugging of applications, and analysis of log messages.

- *Component Broker Glossary*, SC09-2710 contains terms and definitions relating to Component Broker.
- *OS/390 Component Broker Introduction*, GA22-7324 describes the concepts and facilities of Component Broker and the value it has on the OS/390 platform. The audience is a knowledgeable decision maker or a system programmer.
- *OS/390 Component Broker Planning and Installation*, GA22-7331 describes the planning and installation considerations for Component Broker on OS/390.
- *OS/390 Component Broker System Administration*, GA22-7328 describes system administration tasks and operations tasks, as provided in the system administration user interface for OS/390.
- *OS/390 Component Broker Programming: Assembling Applications*, GA22-7326 provides information for assembling applications using Component Broker on OS/390.
- *OS/390 Component Broker Operations: Messages and Diagnosis*, GA22-7329 provides diagnosis information and describes the messages associated with Component Broker on OS/390.

---

## Chapter 1. Introduction

This document introduces the Procedural Application Adaptor within Component Broker. PAA support on Component Broker enables application developers to access and extend access to existing procedural resource managers, such as CICS and IMS. Once created, these objects can be called by Component Broker Business Objects to access and manipulate resources on these third tier systems. This book assumes that one is familiar with Component Broker.

---

### The Procedural Application Adaptor

The Procedural Application Adaptor (PAA) of Component Broker enables Component Broker applications to access procedural resources, such as CICS or IMS. Component Broker ships the CICS/IMS Application Adapter that is based on its Procedural Application Adapter infrastructure. It consists of both a development environment within Component Broker, as well as a runtime environment. The runtime environment integrates the Component Broker services like Security, RAS, and Transaction capability with the various technologies to communicate with these procedural systems like Communications Server, CICS Client, and Host On-Demand. The development environment involves using portions of VisualAge for Java, with its Enterprise Access Beans support and generating beans that can be understood by Object Builder.

---

### What PAA Does

The majority of customer data today exists in legacy data stores that have been around for years. To leverage this data, customers access and manipulate the data through existing transaction programs. This allows customers to extend and to add business value to their existing set of transactions using Component Broker without disrupting their existing work flow. In respect to the reuse of existing transactions, there are many reasons why this is more important than accessing the data directly.

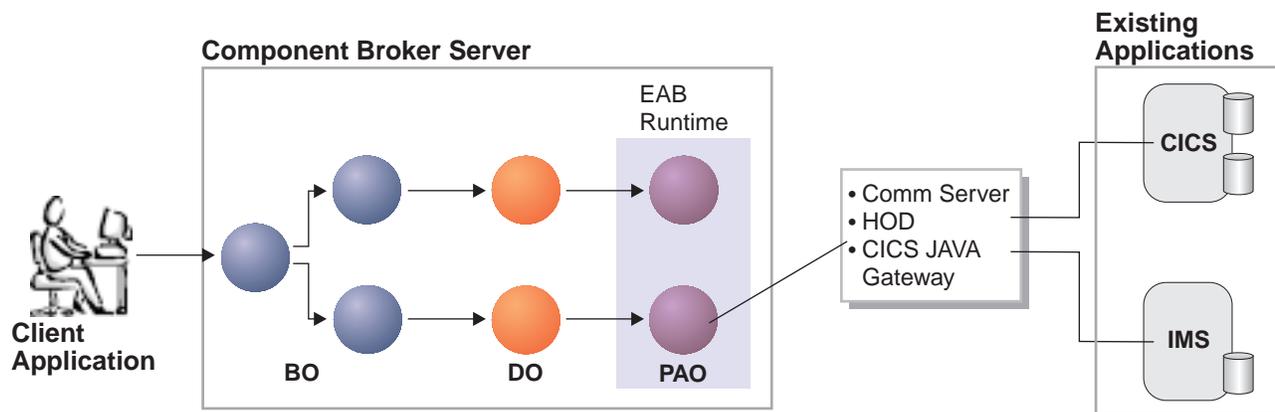
- The existing applications implement business functions and rules. For example, even a simple debit transaction may write trace records, fulfill regulatory reporting requirements, and ensure that daily and weekly withdrawal limits are not exceeded.
- The transactions maintain data integrity between the existing databases, most of which are non-relational and cannot rely on RDB integrity functions.
- The database record formats are complex and evolve over time. Due to the time evolved nature of the record format, the application's implementation may be the only record of what the database schema actually is. For example, for CICS File Control, the file record is often a variant record that is mapped to a canonical form by the CICS transactions.
- The existing applications are carefully crafted to support efficient resource usage and integrity, and cannot, or should not coexist with more complex OO applications that directly access the databases.

All of the functions of the TP programs could be re-implemented in Component Broker business objects, but this is not practical. Moreover, the existing applications are needed to support non-reengineered business processes and for other application models like mission critical, extremely high volume OLTP, for which distributed objects are not yet suitable.

---

## How Customers Use PAA

Customers, in using the Procedural Application Adaptor, create objects that wrap their legacy data. Access to the legacy data is obtained by invoking existing transactions to retrieve the information. The objects contain attributes that map to the bits of underlying data. Since the object represents the state of the object in the legacy data store, its lifecycle is controlled by the standard data access operations create, retrieve, update and delete (CRUD). Component Broker, and its Application Adaptor framework, drive these object instances and call the CRUD methods at the appropriate times. For example, if a client application is attempting to find an object that is not currently instanced in this server, it will create a new object of the appropriate type and issue the retrieve method on it to retrieve all the state attributes that are required. The tools that are provided in Component Broker for PAA support aid in mapping these object's attributes to fields on a request to the underlying procedural system (as fields in a remote procedural call (RPC) style request or fields on a screen). These requests flow through some form of client communication mechanism to the third tier server and are executed with the appropriate data being returned (if any). The figure below represents the target usage scenario for Component Broker.



---

## The Elements of PAA

### Client Application

The client application simply provides the presentation layer for the application. It calls Component Broker Business objects to obtain and manipulate data. It may be a Java, C++, or Active-X client.

### Component Broker Server

The Component Broker server manages Component Broker Applications by instantiating the customer's objects as well as providing the necessary services to manage the accessing of the data and its underlying resources.

### Component Broker Application

Customer specified implementation of Business Objects and its underlying business logic.

### Procedural Adaptor Object (PAO)

Procedural Adaptor Object (PAO) is the cache element that contains the state of an object that is backed by the Procedural Application Adaptor. PAOs inherit the four data access methods (create, retrieve, update, and delete). Customers will use the VisualAge for Java tools (specifically, the Enterprise Access Builder (EAB) support) to build Commands and Navigations that will interact with the legacy system.

### PAA Runtime Library

Integrates the PAA support into Component Broker. It also provides the necessary mapping and usage of underlying technologies that communicate with the tier3 server, such as Communication Server, Host On-Demand, or the Transaction Gateway.

### Host On-Demand

A member of the eNetwork software family that is a Java-based solution that incorporates industry-standard Telnet 3270 (TN3270) protocols. Component Broker ships with a subset of Host On-Demand. This subset provides a Java-based TN3270 client.

### Transaction Gateway/CICS Client

Java/C++ client for a CICS server. It uses two proprietary protocols, ECI and EPI, to issue transactions in CICS. ECI is an RPC-like interface into CICS. EPI is a screen interface into CICS. The CICS Client and Transaction Gateway are requisites for Component Broker, but are shipped with Component Broker.

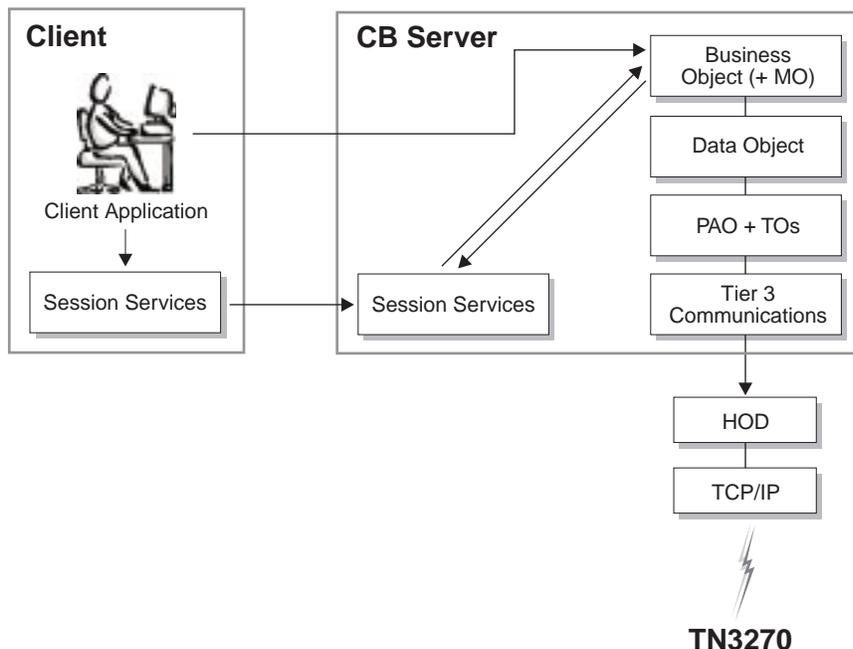
### Communications Server

Communications Server provides the LU 6.2 connection between Component Broker and the CICS/IMS server. Communications Server is a prerequisite for Component Broker.

---

## How the Parts of PAA Work Together with Component Broker

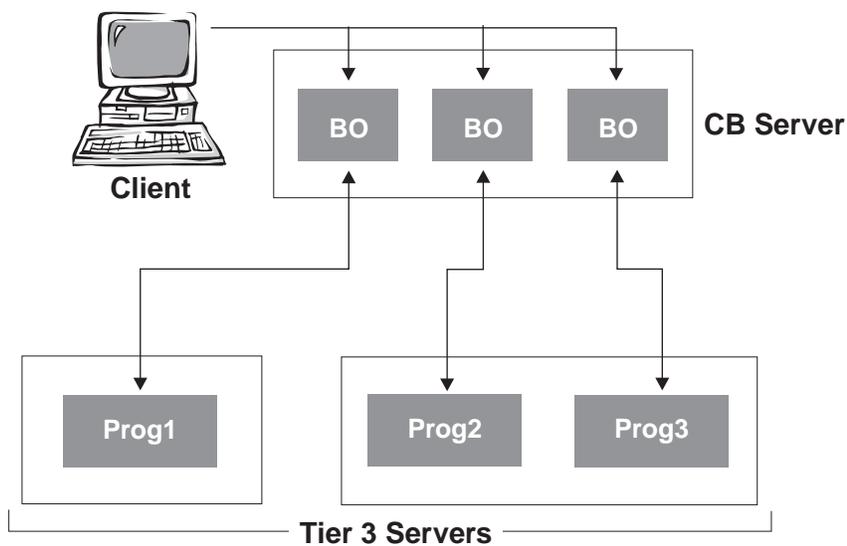
As PAA helps customers access legacy resources, there are three main technologies used within Component Broker to access the underlying resources. The first technology is IBM's Host On-Demand (HOD). HOD gives Component Broker the ability to simulate a user sitting in front of a TN3270 terminal. The simulated user will access the necessary screens to manipulate the data in the tier3 system. In looking at a PAA backed application that uses HOD (from a client perspective), it looks like any other Component Broker-based client application. There is one small difference. PAA client applications will register with Session Services to begin a session to start manipulating data.



Once registered with session services, context flows from the client to the server to inform any objects on the Component Broker server that the client is participating within a session. For managed objects that are configured in a sessional container, when initially activated in the server, they will register themselves as a resource with session services. This resource that is registered will get called back when the client

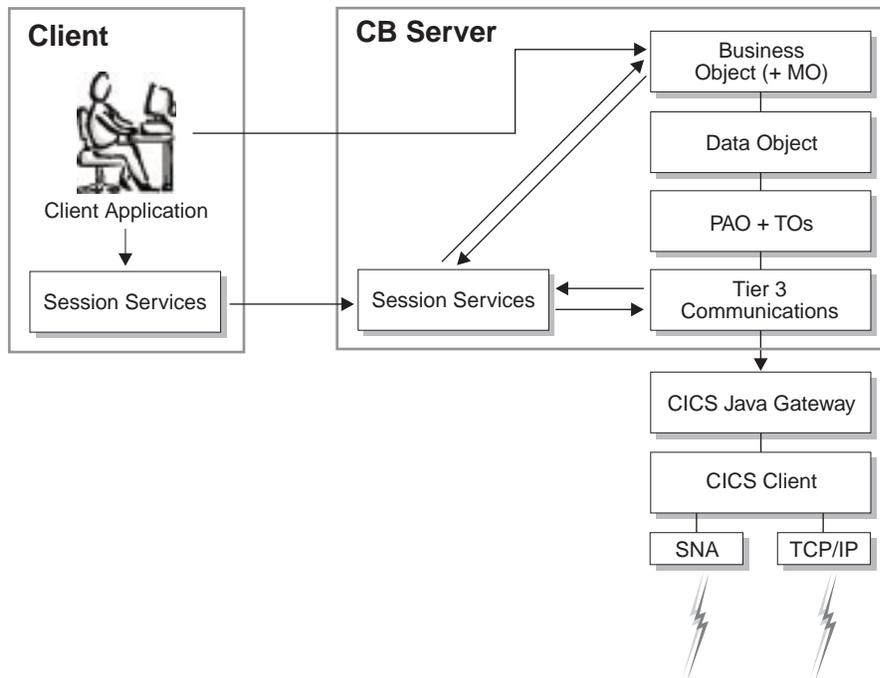
terminates the session. If the client terminates the session requesting a checkpoint, an update is driven on the managed objects to drive whatever updates are needed to the tier3 system.

As managed objects are activated within the system, a retrieve is driven on the objects to retrieve their state from the underlying tier3 system. To retrieve the state, a customer generated navigation is called to traverse the necessary screens to retrieve the state. A request by the Navigation requires a HOD connection to the tier3 system. Upon request for a connection to the tier3 system, if one is not available, one will be created and associated with a session. A resource will be registered with session services to terminate the connection when the session has ended. All further requests for a connection during this session will reuse this connection. As the navigation is executed by the PAA runtime, screens are constructed, passed to the tier3 communications code, that will pass them to HOD. HOD, using its TN3270 based Java client code, executes the screens on the TN3270 daemon to which it is attached. This TN3270 daemon may be a TCP/IP TN3270 daemon started by MVS, a CICS telnet daemon started on a CICS server, or a CICS telnet daemon sitting in front of the CICS Universal Client.



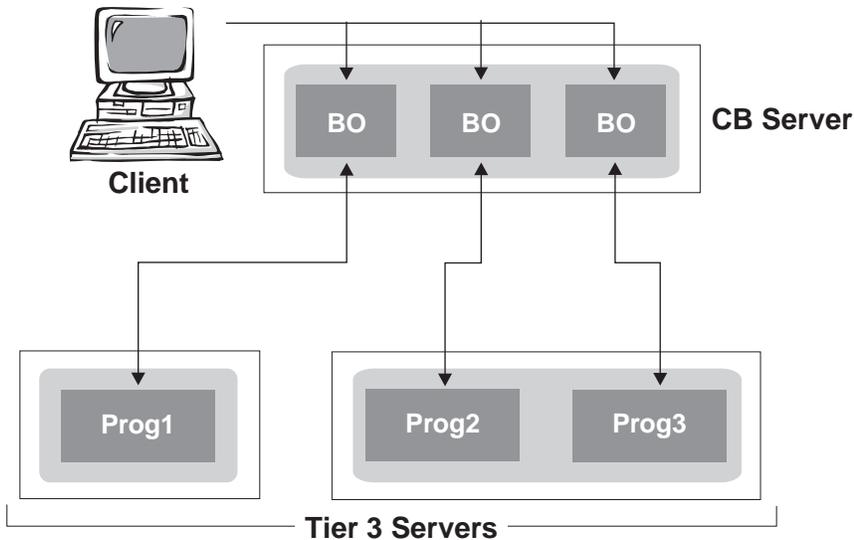
As requests are made on the Component Broker objects and the framework drives the CRUD methods, these individual CRUD requests complete. Once executed, there is no way to roll back what has occurred. This can be viewed as a sync-level 0 transaction. In the figure above, as each Business Object is created, it required a retrieve on each of the tier3 programs. When the client ends the session with a checkpoint, the framework drives updates to the underlying data store. The framework continues to drive updates to all the resources even if one of the resources should fail.

The second technology is the Transaction Gateway and the Universal CICS client and usage of an ECI request. This technology gives Component Broker the ability to issue RPC-like requests to a CICS server to execute existing CICS transactions that manipulate the data in the tier3 system. In looking at a PAA-backed application that uses ECI, it looks like any other Component Broker-based client application (like the HOD technology). PAA applications will register with Session Services to begin a session to start manipulating data.



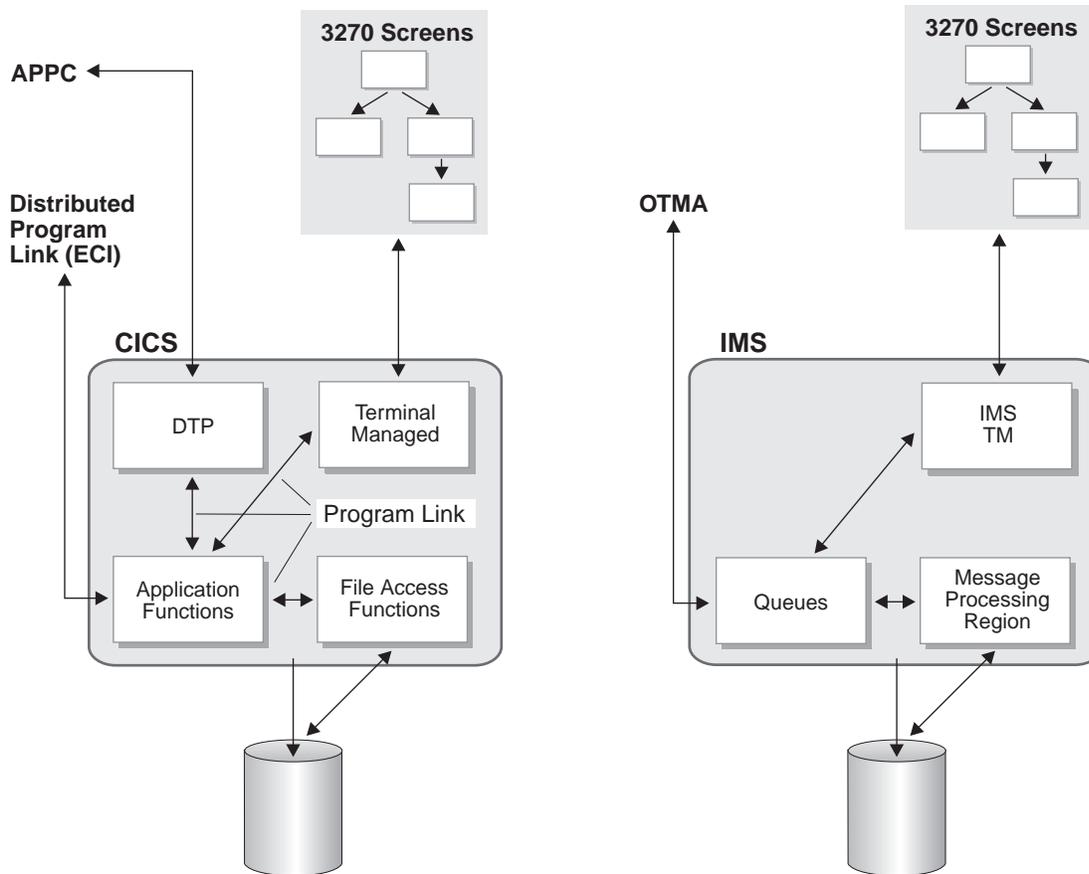
Like HOD, once registered with session services, context will flow from the client to the server to inform any objects on the Component Broker server that the client is participating within a session. For managed objects that are configured in a sessional container, when initially activated in the server, they will register themselves as a resource with session services. This resource that is registered will get called back when the client terminates the session. If the client terminates the session requesting a checkpoint, an update is driven on the managed object to drive whatever updates are needed to the tier3 system.

As managed objects are activated within the system, a retrieve is driven on the objects to retrieve their state from the underlying tier3 system. To retrieve the state, an EAB Command or Navigation is called to build up and issue the necessary RPC-like request(s) to the tier3 system. A request by the command or navigation will require an ECI connection to the tier3 system. Upon request for a connection to the tier3 system, if one is not available, one will be created and associated with a session. A resource will be registered with session services to terminate the connection when the session has ended. All further requests for a connection during this session will reuse this connection. As the command or navigation is run, buffers of data are constructed representing the ECI request and wrapped in a Java ECI call to the Transaction Gateway. These Java ECI requests are then converted to ECI requests and passed to the CICS Universal Client. The CICS Universal client will then issue the ECI request to its configured CICS server.



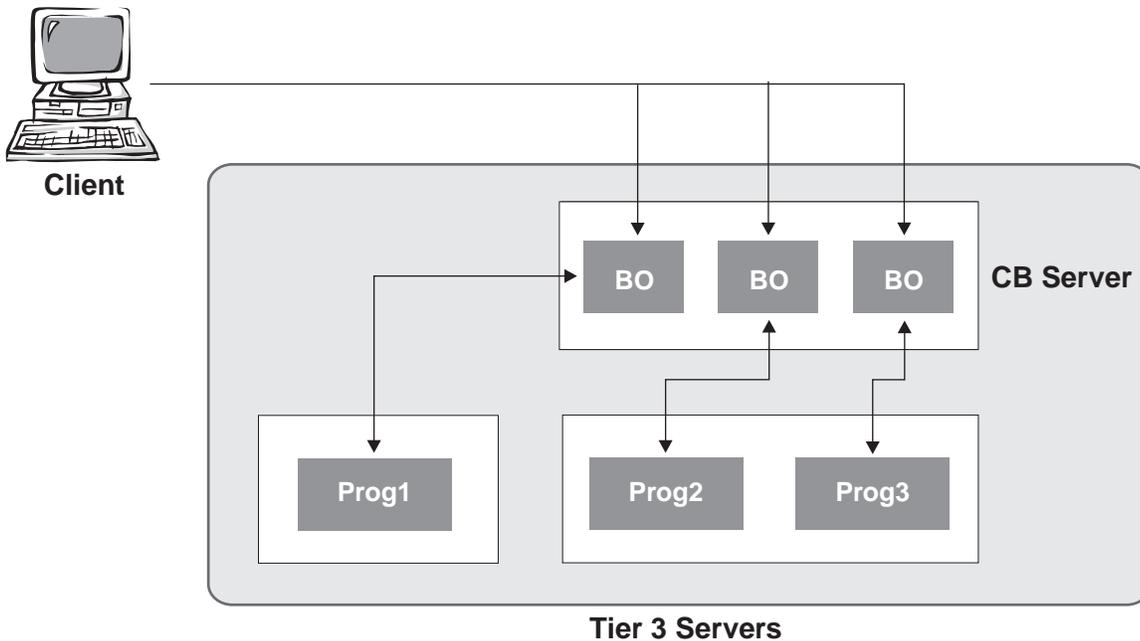
As requests are made on the Component Broker objects and the framework drives the CRUD methods, these individual CRUD requests complete. With ECI, each connection to a tier3 system is associated with a logical unit of work within the session. This can be viewed as a sync-level 1 transaction. If two objects are configured in the same container to the same CICS region, they will use the same connection within the session. As the framework drives updates, the updates are executed. If one of the updates within the session fails, Component Broker has the ability to roll back that logical unit of work in the session. In the example above, two of the business objects are configured in the same container. You will see that the requests to run the Prog2 and Prog3 transactions will execute under a single logical unit of work, that can be rolled back. If all objects participating in the session are configured in the same container on the same server, this would exhibit transactional characteristics of atomicity.

The third technology is Advanced Program to Program Communication (APPC). This technology gives Component Broker the ability to issue RPC-like requests to a CICS/IMS server to execute existing CICS/IMS transactions that manipulate the data in the tier3 system in a transactional fashion. In looking at a PAA backed application that uses APPC, it looks like any other Component Broker-based client application that uses Component Broker transaction services. PAA APPC-based applications will register with Transaction Services to begin a transaction before Component Broker objects in a transactional container are manipulated.



Once registered with transaction services, context will flow from the client to the server to inform any objects on the Component Broker server that the client is participating within a transaction. For managed objects that are configured in a transactional container, when initially activated in the server, they will register themselves as a resource with transaction services. This resource that is registered will get called back when the client commits/rollbacks the transaction. If the client commits the transaction, an update driven on the managed object to drive whatever updates are needed to the tier3 system.

As managed objects are activated within the system, a retrieve is driven on the objects to retrieve their state from the underlying tier3 system. To retrieve the state, an EAB Command or Navigation is called to build up and issue the necessary RPC-like request(s) to the tier3 system. A request by the command or navigation will require an APPCC connection to the tier3 system. Upon request for a connection to the tier3 system, if one is not available, one will be created and associated with a transaction. A resource will be registered with transaction services to terminate the connection when the transaction has ended. All further requests for a connection during this transaction will reuse this connection. As the command or navigation is run, buffers of data are constructed representing the APPCC request and wrapped in an LU 6.2 request to the Communication Server. These LU6.2 requests are then sent to the partner LU 6.2 system (CICS/IMS) to be executed on the tier3 side. Output from the request comes back as an LU 6.2 response from the tier3 side.



As requests are made on the Component Broker objects and the framework drives the CRUD methods, the individual requests complete. With APPC, each connection to a tier3 system is associated with the same logical unit of work within the transaction. If three objects are configured in transactional containers, they will all have connections that are tied into the same logical unit of work. As the framework drives updates, the updates are executed. If one of the updates within the transaction fails, Component Broker can roll back the entire distributed transaction (due to the ability to do two-phase commit processing across all of the participating tier3 resources). This is because APPC has a sync-level 2 capability to act not only as a communication mechanism, but also as a transactional resource manager.

The APPC support that we have added has two varieties: optimistic and pessimistic. The pessimistic variety creates a sync-level 2 connection with its tier3 partner from the beginning. This sync-level 2 connection usually requires the tier3 system to register resources with a sync. point manager and hold them for the duration of the transaction. The optimistic variety is targeted to hold resources for a shorter period of time as it will create a sync-level 0 connection to the tier3 system initially. When the updates need to be driven to the tier3 system, a sync-level 2 connection will be established, initial states retrieved and compared (to ensure changes haven't taken place), and the updates will be driven under the sync-level 1 2 connection. As you can see, this has the added overhead of some extra flows to retrieve the data another time.

---

## Component Broker Flows for a Pessimistic APPC Connection

The following steps represent the typical flows for a pessimistic APPC connection throughout Component Broker for a simple retrieve, modify, and update scenario.

1. The client will drive a `findByPrimaryKey` to the Component Broker server to find an given object by its associated key.
2. The Component Broker server will look to see if the object already exists on the server under a given transaction. If not, it attempts to instantiate the object and invoke `internalizeByPrimaryKey` on the DO (that will set its state on the object's PAO).
3. The PAA framework drives `retrieve` on the DO delegating down to the PAO for it to gather its state from the tier3 system.

4. The PAO drives a retrieve command (constructed using the EAB tools) to drive an RPC-like request via an APPC connection to the backend system.
5. The tier3 communications code checks to see if a connection exists to the tier3 system associated with the current transactional context. If so, it will reuse the connection.
6. If not, it will create a sync-level 2 APPC connection to the partner LU 6.2 system and register itself as a resource with transaction services to be notified during commit/rollback processing.
7. Once connected, the EAB runtime library issues a send of the data on that connection, and waits for the results to be received back from the tier3 system.
8. The state has now been retrieved, the object fully instantiated, has its reference returned back to the client for manipulation.
9. The client manipulates the object (modifying a few of its non-key attributes).
10. The client then issues commit on its transaction.
11. The commit flows to transaction services to its associated coordinator to drive commit processing.
12. The managed object was registered with transaction services as a synchronization object to be called prior to commit processing. The `before_completion` method is called on the MO to drive any updates on the managed object to its underlying datastore.
13. Updates are driven on the data object (and delegated to its corresponding PAO object).
14. The PAO drives an update command (constructed using the EAB tools) to drive another RPC-like request via the existing APPC connection to the back end system. This results in another set of send/receive flows across the connection.
15. After the updates have been driven to the underlying datastore, a prepare is now called on all resource objects that are registered with transaction services.
16. The OTS/APPC code starts a prepare on all of the currently allocated APPC conversations involved with this transaction. If the updates and the prepare were successful, it would vote to commit the changes to the underlying datastore.
17. If all of the resources vote commit, commit processing is invoked on all the transactional resource objects.
18. OTS/APPC drives commit on the conversation. After the conversation is committed, it is deallocated.
19. All registered synchronization objects have `after_completion` called on them for cleanup processing.
20. Transaction services performs its necessary cleanup (as the transaction completed successfully).
21. Control is returned to the client to continue.

---

## Component Broker Flows for an Optimistic APPC Connection

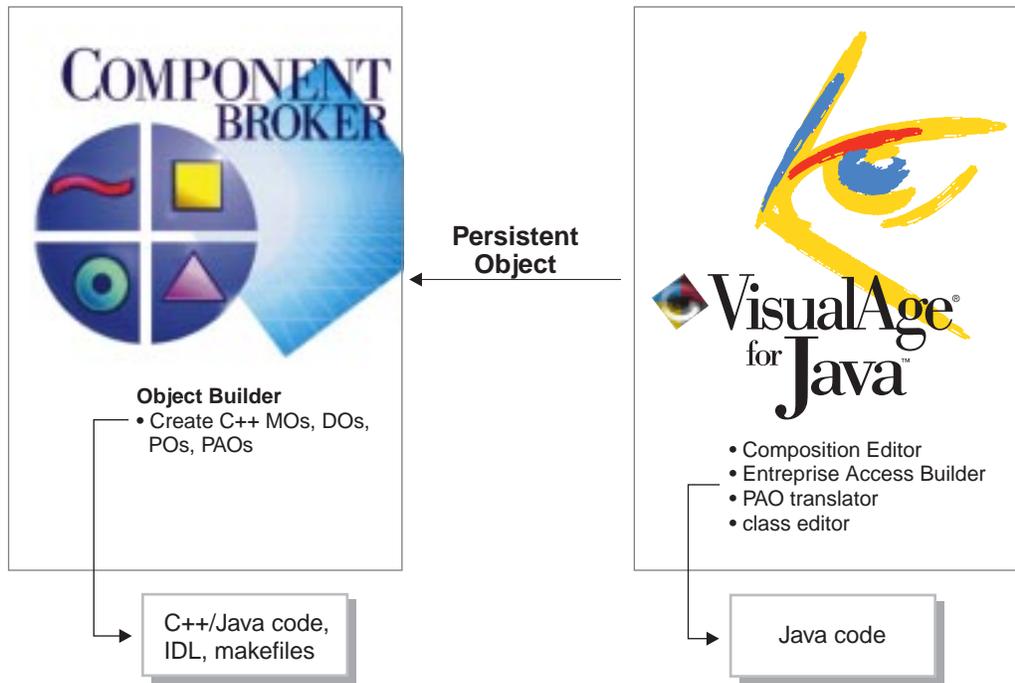
The following steps represent the typical flows for an optimistic APPC connection throughout Component Broker for a simple retrieve, modify, and update scenario. The changes for optimistic versus pessimistic have an asterisk (\*) next to them to distinguish that different processing is happening.

1. The client drives a `findByPrimaryKey` to the Component Broker server to find a given object by its associated key.
2. The Component Broker server looks to see if the object already exists on the server under a given transaction. If not, it will attempt to instantiate the object and invoke `internalizeByPrimaryKey` on the DO (that will set its state on the object's PAO).
3. The PAA framework drives retrieve on the DO delegating down to the PAO for it to gather its state from the tier3 system.

4. The PAO drives a retrieve command (constructed using the EAB tools) to drive an RPC-like request via an APPC connection to the back end system.
5. The tier3 communications code checks to see if a connection exists to the tier3 system associated with the current transactional context. If so, it will reuse the connection.
6. \* If not, it will create a sync-level 0 APPC connection to the partner LU 6.2 system and register itself as a resource with transaction services to be notified during commit/rollback processing.
7. Once connected, the EAB runtime library issues a send of the data on that connection, and waits for the results to be received back from the tier3 system.
8. \* A copy of the received state is saved from the retrieve and associated with the PAO.
9. The state has now been retrieved, the object fully instantiated, has its reference returned back to the client for manipulation.
10. The client manipulates the object (modifying a few of its non-key attributes).
11. The client then issues commit on its transaction.
12. The commit flows to transaction services to its associated coordinator to drive commit processing.
13. The managed object was registered with transaction services as a synchronization object to be called prior to commit processing. The before\_completion method is called on the MO to drive any updates on the managed object to its underlying datastore.
14. Updates are driven on the data object (and delegated to its corresponding PAO object).
15. \* As update is issued to the PAO, it will recognize that it is configured for optimistic support. Therefore, another retrieve is issued on the PAO to gather its state again from the underlying datastore.
16. \* This second retrieve will deallocate the sync-level 0 conversation and allocate a sync-level 2 conversation. It will then flow a second retrieve to the backend system.
17. \* The results from this retrieve are compared with the original results (under the sync-level 0 operation). If different, a rollback is thrown back to transaction services. Otherwise...
18. The PAO will drive an update command (constructed using the EAB tools) to drive another RPC-like request via the existing APPC connection to the backend system. This results in another set of send/receive flows across the connection.
19. After the updates have been driven to the underlying datastore, a prepare is now called on all resource objects that are registered with transaction services.
20. The OTS/APPC code starts a prepare on all of the currently allocated APPC conversations involved with this transaction. If the updates and the prepare were successful, it would vote to commit the changes to the underlying datastore.
21. If all of the resources vote commit, commit processing would be invoked on all the transactional resource objects.
22. OTS/APPC would drive commit on the conversation. After the conversation is committed, it is deallocated.
23. All registered synchronization objects would have after\_completion called on them for cleanup processing.
24. Transaction services performs its necessary cleanup (as the transaction completed successfully).
25. Control is returned to the client to continue.

## Developing a PAA Application

PAA Applications are developed within both the VisualAge for Java and Object Builder environments. Essentially, the VAJ tools are used to develop the requests to the tier3 system. Once the requests are created, the artifacts created are imported into Object Builder and connected with the rest of the Component Broker environment.



To develop a PAA application, complete the following steps:

- Analyze the existing CICS/IMS transactions
- Design the objects that will be created in object space and how the attributes on the objects relate to the fields on a screen/buffer passed to/from the CICS/IMS system
- Use the Application Development tools within VisualAge for Java to:
  - Create your PAO
  - Parse the COBOL definitions or BMS/MFS screen mappings
  - Create a mapper to map from the PAO object to the EAB buffer or vice versa
  - Develop commands and navigators to create, retrieve, update, and delete the information
  - Modify the PAO CRUD methods to call the associated commands/navigators
  - Unit test navigations/commands to ensure they access the data correctly
  - Use Object Builder to connect with Component Broker Business objects
  - Deploy as any other Component Broker-based application

## Analyzing the Existing CICS/IMS Transactions

Accessing existing transactions on these tier3 systems requires some sort of client technology (client to the tier3 server). That technology may be screen based (using screen scraping techniques) or RPC based (using programmatic client access technology) like APPC or ECI in order to access the legacy transactions. With either mechanism, the semantics of the transaction must be fully understood to successfully wrap these transactions within Component Broker. Understanding the transactions may involve using them to discover the various ways they can be navigated and how they react to various inputs. It may also involve looking at the underlying source code to fully understand what the transaction is doing. The more that you understand about the existing transactions, the easier it will be to develop the wrappers to map them to objects within Component Broker.

If the existing transactions were screen based, part of the analysis phase would be to understand what navigations through what set of screens retrieved the information that you needed to understand. For example, if you were attempting to get all of the information about a person through an existing CICS screen based application, you would have to perhaps issue the initial menu transaction, enter some information in a menu (such as the person's name), and be presented with a screen with the details about that person. Once finished, you may have to clear the screen to get back to the original state to prepare to enter another transaction.



If the existing transactions were RPC based, part of the analysis phase would be to understand what the data areas look like that must be created in order to issue the transaction. For example, if you were attempting to issue an ECI or APPC request to CICS, that data area layout could be defined in a COBOL source file. You would have to get the source file to understand what the fields are that must be filled out when the transaction is issued as well as what fields get filled in on the return from the transaction. In the following COBOL source file, the WS-COMMAREA-BUFFER identifies the format of the data that is required by the CICS COBOL transaction as it is called via ECI or APPC from another application.

```

*
IDENTIFICATION DIVISION.
PROGRAM-ID. BECASHAC.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-COMMAREA-BUFFER.
03 COMM-REQUEST-TYPE PIC 9(2).
03 COMM-RETURN-VALUE-1 PIC X(8).
03 COMM-RETURN-VALUE-2 PIC X(8).
03 COMM-TRACE PIC X.
03 COMM-TOTAL-RECORDS PIC X(4).
03 COMM-RES-TYPE PIC X(2).
03 COMM-ACCOUNTID PIC X(8).
03 COMM-BALANCE PIC 9(8).
03 COMM-TYPE PIC X.
03 COMM-TYPE PIC X.
LINKAGE SECTION.
01 DFHCOMMAREA PIC X(1242).
PROCEDURE DIVISION.

```

---

## Designing the Objects

Once the existing transactions are analyzed, objects must be created within Component Broker that hold the state from the transactions that are executed on the tier3 system. The PAO will have a set of key attributes that reflect its identity in the tier3 system. It also has a set of other non-key attributes corresponding to the state of the object. For example, in looking at the COMMAREA defined in the preceding COBOL program, the fields COMM-ACCOUNTID and COMM-RES-TYPE are the two fields that represent the key to identify an account in a CICS server. The fields COMM-BALANCE, COMM-TYPE, and COMM-UTILITIES represent three other fields that are non-key but information nonetheless about the account.

---

## Creating the PAO

Using VisualAge for Java, a new class must be created that inherits from EntityProceduralAdapterObject. This class introduces four abstract methods that need implementations filled in. The methods (insert, retrieve, update, and delete) correspond to the four data access mechanisms that are driven by the Component Broker framework when activating or passivating objects in the Component Broker server. As attributes (or properties) are defined for a class within VisualAge for Java, getter and setter methods are created for the attributes. For example, in the PAO shown in the following figure, you can see what methods will eventually exist. Note that there are two methods missing (the setter methods for the key attributes). These methods were removed since they will be obtained via another object that will be tied tightly with this PAO (the PAOKey object).

```

+ @ BeCashAcctPAO
- BeCashAcctPAO()
- del()
- getAccount_ID()
- getBalance()
- getRes_type()
- getType()
- getUtilities()
- insert()
- retrieve()
- setBalance(String)
- setType(String)
- setUtilities(String)
- toString()
- update()

```

The PAOKey object inherits from BusinessObjectKey. A key is used to locate its target object, an EntityProceduralAdapterObject, in the current object space. The name of the key class created must be the name of the target PAO with a "Key" appended at the end. In the case above, it will look like the description shown in the following figure.

```

+ @ BeCashAcctPAOKey
- BeCashAcctPAOKey()
- addPropertyChangeListener(PropertyChangeListener)*
- firePropertyChange(String, Object, Object)
- getAccount_ID()
- getPropertyChange()
- getPropertyValues()
- getRes_type()
- removePropertyChangeListener(PropertyChangeListener)*
- setAccount_ID(String)
- setRes_type(String)

```

---

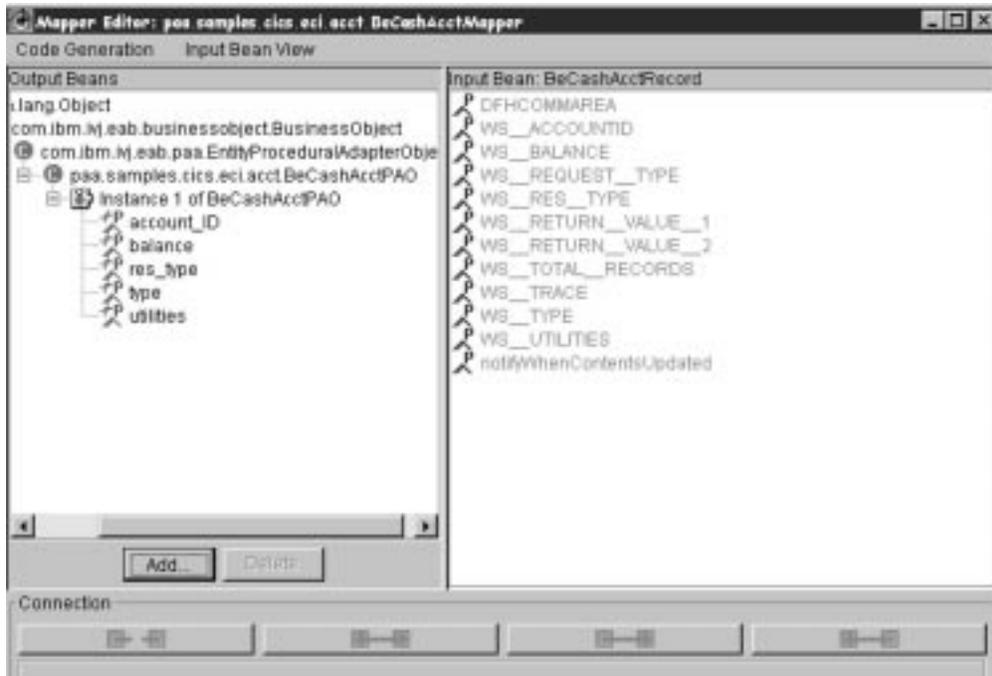
## Parsing the Definitions

A Record, in VisualAge for Java, is a logical collection of application data elements. These data elements are related by application level semantics that is stored and retrieved as a unit. For example, data elements in the ECI case would correspond to fields in a COMMAREA. All of the fields, such as the COMMAREA, are retrieved as a unit. After you retrieve a record, you can then access the individual data elements directly. In the ECI example above, you can access the individual fields. These records will define the structure of what the input or output to a Command is within Enterprise Access Builder. To create a Record Type that is based on the structure defined by a COBOL program, the **Create COBOL Record Type** tool is used to parse the .ccp file and generate a record type. Within this tool, a user specifies the name and location of the .ccp file and the data area/COMMAREA desired within the .ccp file, and the tool generates a dynamic record type bean describing the layout of the data area. This record type can be edited to manipulate the fields if desired. When finished, another tool is run to generate Record Beans that are used by the Commands that interface with the CICS/IMS system.

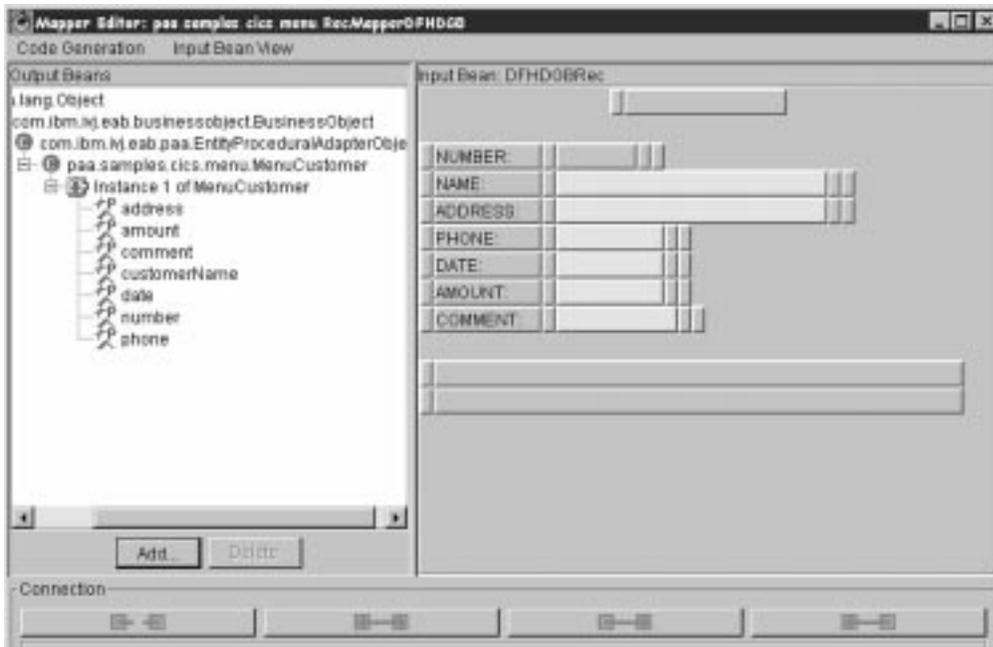
If the definitions that defined the input and output fields to the commands were BMS or MFS maps, then a different set of tools are used to create the record types. The tools are called **Create BMS Record Type...** and **Create MFS Record Type...**, respectively. The record types generated would contain information that would describe the layout of the input/output screens.

## Creating a Mapper

Once you have a PAO that requires data from a record bean (or vice versa), a mapping is needed to permit the exchange of data between the record data contained in a command and a PAO (or set of PAOs). The Mapper Editor is used to generate a mapper object to do this mapping for you. In the mapper editor, you specify two entities; a record bean to map to/from, and a PAO to map to/from. Once both have been specified, you can visually connect the properties in the record bean to properties in the PAO (and which way the data should flow between the two). In the following example, the appropriate fields in the PAO have been mapped to the corresponding fields in the record bean.



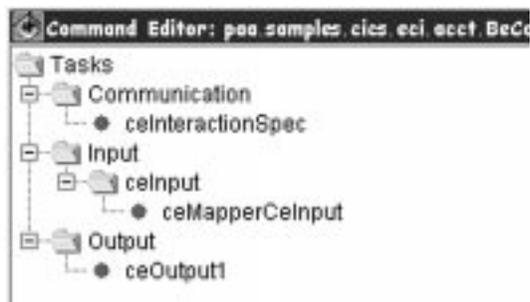
In the case of a mapping between a record bean defining a screen layout, the mapper can be configured to present the screen layout that is defined by the record beans. The following example demonstrates such a configuration.



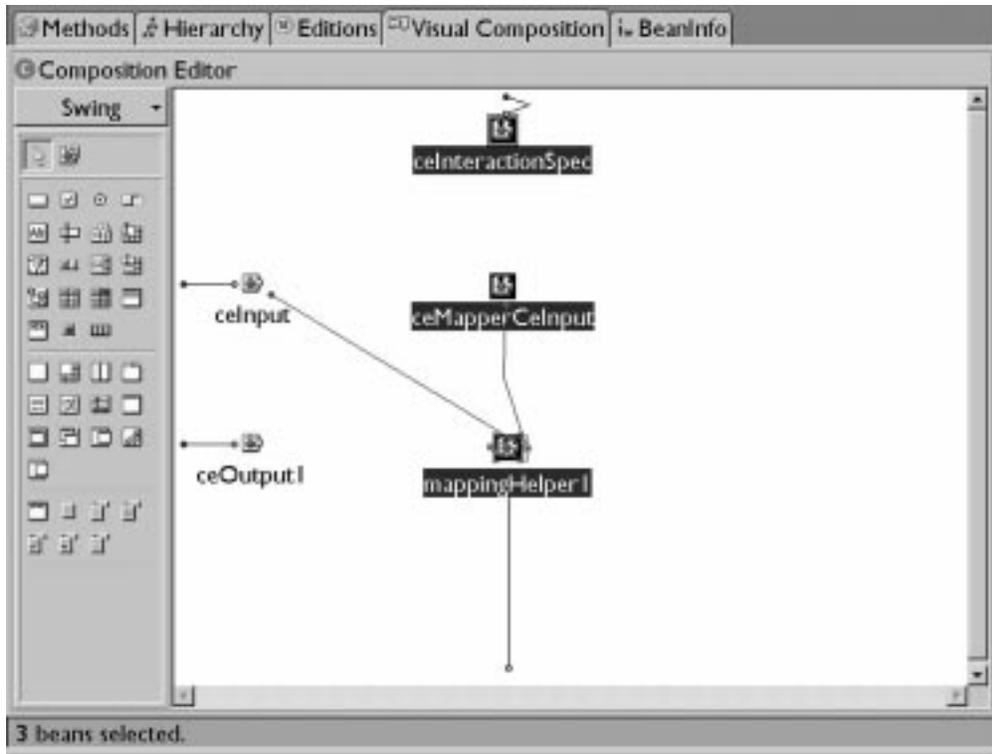
## Creating the Commands and Navigators

An Enterprise Access Builder Command wraps a single interaction with a host system. Upon execution, an EAB Command takes its input data and sends the data via a connector to a host system. It then returns, as its output, the data returned by the host system. There are two ways to construct a command: Visual Composition Editor or the Command Editor.

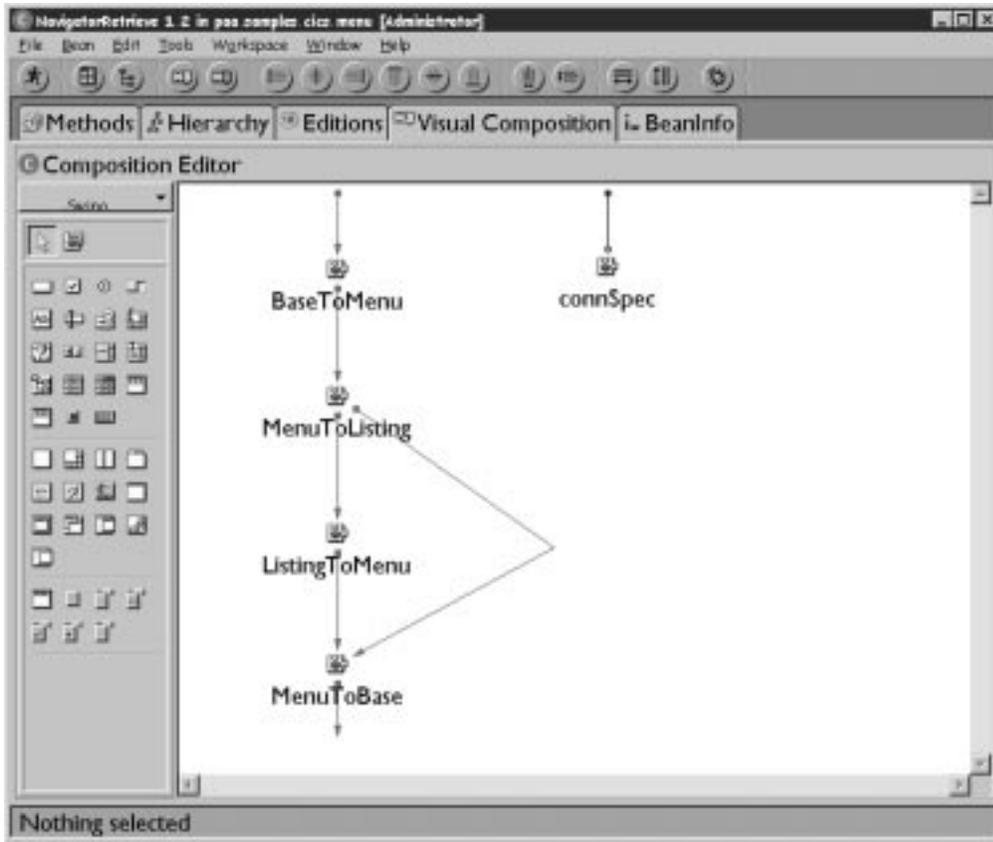
When constructing a command, the input data, output data, and connection information must be defined to the command. Using the Command Editor, each piece of the command can be accessed in a task approach (see the following figure).



Using the Visual Composition Editor, each part needed for the Command can be placed in the workspace (see the following figure).



If multiple interactions with the backend system are necessary, such as a set of screens needing to be traversed or multiple RPC requests to a backend system, a Navigator can be built up to script the interactions with the backend system. From the outside, a Navigator looks like a command. It consists of commands and other navigators strung together to form a more complex interaction with a host system. When you execute a Navigator, it takes input and provides it to the commands and navigators that it is composed of. Each command in the Navigator is executed in the order specified. After the final interaction, the output of the individual commands and navigators can be made available as output of the navigator. For example, for a set of screen interactions, the following sample would represent a Navigator.



## Modifying the PAO CRUD Methods to Call Commands and Navigators

Now that Commands or Navigators have been created to interact with the backend system, they should be tied together with the CRUD methods that the Component Broker Application Adaptor framework will call. Therefore, each method should be modified to call the appropriate command while passing the appropriate information to interact with the backend system. In the following example, a command is created, the key information set on it, and the execute command is called to execute the interaction with the backend CICS system. Afterwards, output information is queried to determine whether or not the command had executed properly.

```
public void insert() throws com.ibm.ipaa.IDataKeyAlreadyExistsException {
    BeCashAcctInCommand bec = new BeCashAcctInCommand();
    bec.setCeInputWS__REQUEST__TYPE(new Java.lang.String('01'));
    bec.setCeInputWS__ACCOUNTID(this.getAccount_ID());
    bec.setCeInputWS__RES__TYPE(this.getRes_type());
    bec.execute();
    if (bec.getCeOutput1WS__RETURN__VALUE__1().equals('00000014'))
        throw new com.ibm.ipaa.IDataKeyAlreadyExistsException();
}
```

---

## Unit Testing in the VisualAge for Java Environment

Once the object is developed, VisualAge for Java provides a stand alone testing environment in order to validate that the commands and objects that were created all work properly. Essentially, to test, another class with a main() function must be created that will drive the new commands and navigations. A sample of such a test is shown below. In the test, a PAO object is created and the method retrieve is called to retrieve the PAOs information from the underlying CICS data store.

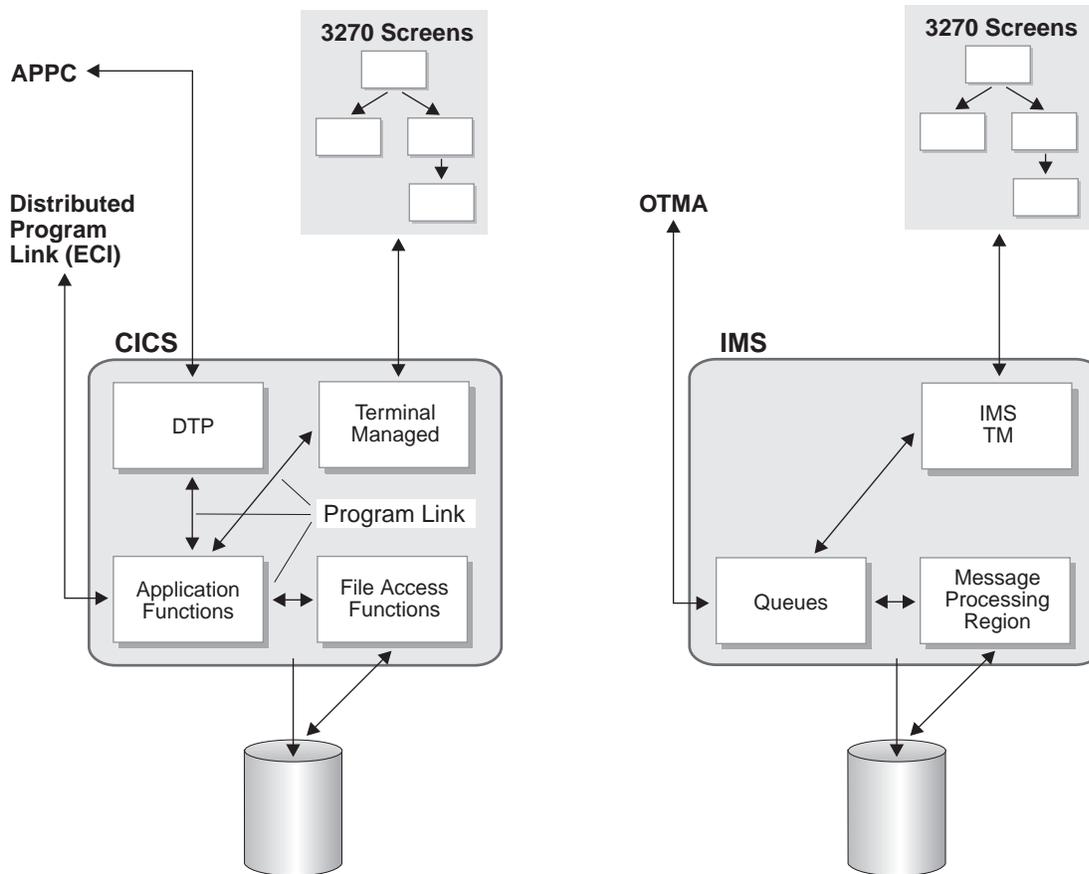
```
BeCashAcctPAOKey key = new BeCashAcctPAOKey();
key.setAccount_ID('87654321');
key.setRes_type('01');
BeCashAcctPAO bec = (BeCashAcctPAO) key.getTarget(true);
// Retrieve Cash Account information
try
{
    ((BeCashAcctPAO)key.getTarget(true)).retrieve();
}
catch (Exception e)
{
    System.out.println('Exception from Retrieve ' + e.toString());
}
System.out.println('\n\n' + bec.toString());
```

---

## Importing Into Object Builder

Once the PAO object has been created, you are now ready to import this into Object Builder to connect with the rest of your Component Broker objects. Object Builder allows the user to import a Procedural Adaptor Object (PAO). A PA schema is created when a PAO bean is imported from VisualAge for Java. This schema has an associated persistent reference at the time of its creation. The PAO importer will introspect on the PAO bean for the methods and attributes that it supports. The attributes will be exposed as getters/setters, the framework methods exposed, and any other methods exposed as pushdown methods. After the PAO has been imported, the user would then use Object Builder to connect attributes, framework methods, and pushdown methods on a DO Implementation to the PO. After the PAO has been connected, the structure may look like the following figure.





CICS and IMS applications are logically composed of the following parts:

- terminal management and screen services manage the connection state with the 3270 terminals, and convert between the BMS and MFS data and the input/output buffers expected by the application logic (in IMS, IMS TM implements these functions; in CICS, these functions are performed by a Terminal Owning Region).
- application logic implements the COBOL, C, etcetera, business functions implemented by the application programmer.

In IMS, Message Processing Regions contain the implementation of the end-user applications. IMS-TM and MPRs communicate through shared messages with the following high level flows:

1. IMS TM receives an incoming 3270 screen.
2. IMS TM uses MFS to extract the relevant fields from the 3270 data stream, formats a message and places the message on a queue.
3. An MPR removes the message, passes the data business functions, and accesses and updates DL/1 or DB2 data.
4. A response message is placed on an outbound queue, and the transaction commits. The write to the queue is part of the transaction commit scope.
5. IMS TM reads the response message, updates or formats a new 3270 data stream using MFS, and sends the stream to the terminal.

The CICS processing is similar. The incoming 3270 data stream maps to a first transaction program based on the transaction ID. This programs uses BMS to map and extract the fields from the data stream, and formats a COMMAREA. The first program performs a (Distributed) Program Link to pass the COMMAREA

to the end-users business functions that run as a transactional subroutine. The transaction logic executes, the databases are read and updated, and the DPL returns to the first TP. This program maps the COMMAREA back into a 3270 data stream using BMS, performs a Commit that includes the DPL updates in the commit scope, and sends the stream to the terminal. A CICS region that contains application logic is called an Application Owning Region (AOR). A CICS region may be both a Terminal Owning Region (TOR) and an AOR.

---

## Definitions

Component Broker, a middleware product, integrates a number of diverse products under one umbrella to present a consistent interface to it. As such, there is an explosion of the number of terms, concepts, and acronyms that you should be familiar with. Following are brief descriptions of some of the terms:

**CICS** CICS (Customer Information Control System) is IBM's general-purpose online transaction processing (OLTP) software. It is a powerful application server that runs on a range of operating systems. CICS seamlessly integrates all the basic software services required by OLTP applications. Typical OLTP applications include: retail distributed systems, finance, order entry and processing, payroll, ATM, and airline reservation systems.

**IMS** IMS (Information Management System) consists of two pieces: IMS Database Manager (IMS DB) and IMS Transaction Manager (IMS TM) that run under the MVS operating system. IMS DB is a database system. IMS TM is a data communication mechanism. It provides high-volume, high-performance, high-capacity, low-cost transaction processing from both IMS DB and DB2 databases. IMS TM uses input and output message queues. It schedules messages by associating programs with the transactions that they are to process.

### COMMAREA

A COMMAREA (communications area) is a CICS area that is used by CICS applications to pass data between tasks that communicate with a given terminal. The area can also be used by CICS applications to pass data between programs in a task.

### MFS maps

Message Format Service (MFS) maps are a high-level 3270 screen definition language. MFS maps are used by the MFS service within IMS to provide interfaces that enable IMS applications to set, read, and update fields in a 3270 data stream. The language allows application programs to deal with simple logical messages instead of device-dependent data, thus simplifying the application development process.

### BMS maps

Basic Mapping Support (BMS) maps are a high-level 3270 screen definition language. BMS services provide interfaces for CICS application programs that enable CICS applications to set, read, and update fields in a 3270 data stream. BMS maps tell BMS how to format field data for display.

**HOD** Host On-Demand. A member of the eNetwork software family that is a Java-based solution that incorporates industry-standard Telnet 3270 (TN3270) protocols.

**ECI** External Call Interface. A facility that allows a non-CICS program to run a CICS program. Data is exchanged in a COMMAREA as for normal CICS interprogram communication.

### APPC and LU 6.2

APPC (Advanced program-to-program communication) is an implementation of the SNA LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs. The part of an application that initiates or responds to APPC communications is a transaction program (TP). It is part of a program that handles transactions (or exchanges of data) with another program. The communication between two transaction programs is called a conversation.

**Transaction**

Transactions have a variety of meanings. In the context of IMS or CICS, a transaction is a set of input data that triggers the execution of a specific process or job. In the context of Component Broker, a transaction is an atomic unit of work: either all the actions in a transaction are committed, or none at all. Within Component Broker, a transaction is coordinated by Object Transaction Services (OTS). In the context of APPC, a transaction is an exchange of data between two transaction programs.

**Session** Used within Component Broker to manage resources within the context of a unit-of-activity scope. It is similar to the notion of transactions but allows the scope to be aligned with an application rather than individual transactions.

**Pushdown Method**

A method whose business logic is in the 'procedural' call on the underlying legacy application.

**Enterprise Access Builder**

Enterprise Access Builder (EAB) is part of the VisualAge for Java Enterprise Toolkit that enables development of Java code that is targeted to access legacy systems.



---

## Chapter 2. Planning the Install

This chapter provides information you may find useful in planning your CICS and IMS application adaptor install. This chapter contains the following information:

- “Packaging”
- “System Requirements”
- “Prerequisites”
- “The Component Broker Package” on page 26
- “Installation Considerations” on page 26
- “Differences Between VisualAge for Java for Component Broker 1.3 and 2.0” on page 27

---

### Packaging

The CICS and IMS Application Adaptor is packaged as follows:

- CICS and IMS Application Adaptor compact disc
- *Component Broker for Windows NT and AIX CICS and IMS Application Adaptor Quick Beginnings*

---

### System Requirements

The system requirements for installing Component Broker CICS and IMS application adaptor support is the same as for installing the base Component Broker packages. For additional details, see the *Component Broker for Windows NT and AIX Quick Beginnings*.

---

### Prerequisites

WIN The following products are prerequisites for the Component Broker CICS and IMS application adaptor for Windows NT:

- IBM Component Broker Connector for Windows NT, Version 2.0 and all associated prerequisites  
**Note:** As a minimum, the Component Broker server must be installed.
- IBM eNetwork Communications Server Version 5.0 for Windows NT, for APPC support.
- IBM VisualAge for Java Enterprise Edition for Windows, Version 2.0, for development only, packaged with the Component Broker Toolkit package.
- IBM CICS Transaction Gateway for Windows NT, Version 3.0. This is installed automatically when IBM VisualAge for Java Enterprise Edition for Windows (for development only) is installed.
- JavaSoft Java Development Kit (JDK) 1.1.6 or above.

#### Notes:

1. The CICS and IMS application adaptor requires a special version of IBM Host On-Demand. This version is installed during the CICS and IMS application adaptor install and is called `somhod20.jar`. If Host On-Demand was previously installed, ensure that the JAR file associated with the version shipped as part of Component Broker is in the CLASSPATH before any earlier versions. The earlier version is the `hac120.jar` file.
2. The CICS and IMS application adaptor capabilities of Object Builder require that the Component Broker CICS and IMS application adaptor be installed in the same directory as the Component Broker

run time and Object Builder (that is, in x:\CBroker, where x: is the drive on which you install the product).

3. If you are using CICS on a computer running Windows NT, you must use Transaction Server for Windows NT 4.0.1 or later.
4. If you are using CICS on a mainframe computer, you must be using CICS/ESA 3.2.1 or later.
5. If you are using IMS on a mainframe computer, you must use IMS 6.1 or later.

**AIX** The following products are prerequisites for the Component Broker CICS and IMS application adaptor for AIX:

1. IBM Component Broker Connector for AIX, Version 2.0 and all associated prerequisites.  
**Note:** As a minimum, the Component Broker server must be installed.
2. IBM eNetwork Communications Server, Version 5.0 for AIX, for APPC support.
3. IBM Visual Age for Java Enterprise Edition for AIX, Version 2.0, for development only, packaged with the Component Broker Toolkit package.
4. IBM CICS Transaction Gateway for AIX, Version 3.01. This is installed automatically when IBM VisualAge for Java Enterprise Edition for AIX (for development only) is installed.
5. JavaSoft Java Development Kit (JDK) 1.1.6 or above.

---

## The Component Broker Package

The CICS and IMS application adaptor run time are on the CICS and IMS Application Adaptor compact disc. Install this software on your system if you want to access a CICS or IMS server.

---

## Installation Considerations

The following restrictions apply to the Component Broker CICS and IMS application adaptor.

- The CICS and IMS application adaptor does not support the Component Broker cache service. However, caching is supported by EAB, but non-caching is not supported. Also, the following cache restrictions apply:
  - The Component Broker for Windows NT and AIX does not support the caching of data in the managed object or business object.
- The CICS and IMS application adaptor does not support the query service. Also, the following query restrictions apply:
  - Iteration is not supported; therefore, iterable home is not available.
  - The base home is the only home supported.
- The CICS and IMS application adaptor does not support secondary or mutable keys. Also, the following key restrictions apply:
  - Changes made to a backend datastore may not be propagated to an underlying session.
  - A key component, a part of an object reference, must contain only the primary key attribute.
- If the CICS and IMS application adaptor is to access tier-3 systems that require a user ID and password, the Component Broker server must be a secure server.

- To develop Procedural Adaptor Object beans in VisualAge for Java, you must have first installed the Component Broker server. The CB installation adds features to your /ide/features directory which you import into your workspace as directed in the sample applications.

## Differences Between VisualAge for Java for Component Broker 1.3 and 2.0

For Release 2.0 of Component Broker, you are required to use IBM Visual Age for Java Release 2.0 to develop your Procedural Adapter Object beans. From R1.3 to 2.0, there were significant changes in the CICS and IMS support in Visual Age for Java. This section provides an overview of those differences.

### General Differences

- The process of importing PAO beans into Object Builder has not changed significantly. However, you must specify which kind of connection the bean uses.
- The PAO bean builder has been renamed from CICON (CICS and IMS Connection) to EAB (Enterprise Access Builder).
- Transaction Objects are analogous to Navigators.
- In R1.3, TransRecords contained record information internally as methods. In 2.0, record information is externalized as beans, which can be accessed by other beans such as the mapper bean.
- In R1.3, the TransRecord in Transaction Objects was represented by nodes, and interactions with the backend system were represented by arrows. In R2.0, the interactions with the backend system are called Commands, and are represented by the nodes in the Navigator. Arrows indicate the order of execution of Commands. Mapping is done to and from the input and output Records of the Commands and the PAO.
- In R1.3, Transaction Objects were used for signing on to a system. In R2.0, Navigators perform this task.
- Transition parameters are now represented by Records. Previously, you could define constants using the Transitions tab. Now you define constants by modifying the properties of a bean (usually a Record).
- Most of the information specified in the Advanced tab of the properties sheet can now be specified in the Record Editor. For example, combining, splitting, adding or removing of fields, specifying new lengths, and specifying constant values can be done in the Record Editor.
- Where you previously generated a TOM method, you can now invoke the execute Command.

**PAO Beans:** The SuperClass CBProceduralAdapterObject has been replaced with EntityProceduralAdapterObject. This class is defined in a similar manner (attributes properties and method properties for pushdowns) and still has insert, delete, update, and retrieve methods. The defining of the key has changed (see below).

**Key (Applies to all scenarios):** Previously, you defined a key by adding the term "#key# " in the short description of the PAO's attribute properties. Now you define a Key class (which is itself a bean) that derives from BusinessObjectKey.

**Records and Record Mappers (new for 2.0):** In R2.0, Records contain the information about the fields in the HOD screen, or in the COBOL COMMAREAs. Record Mappers map these fields to the properties of the PAO. Record Mappers replace the use of Edit Connection Screen that was previously accessed under the Connections tab. Any mappings you did for R1.3 must be remapped using the Record Mapper tool, which stores the mapping information in classes.

In R1.3, Host On Demand mapping information was captured in the form of classes. When defining the Transaction Object (TO), transaction records contained information about the fields, and how to map them to the PAO (do not confuse the 1.3 transaction records, which were not classes themselves, with what are called Records in R2.0). This information was duplicated if similar transaction records were required in several TO's.

In R1.3, ECI and APPC beans were used to capture information about the fields in the COMMAREA, but the transaction records in the TO contained the information on how these mapped to the PAO.

In R1.3, there was no conversion available between the type of the field in the BMS/MFS/CCP and the type of the PAO property; you needed to use the Conversion Manager for the appropriate TO. In R2.0, if an appropriate conversion is not available, you need to modify the method that maps from the BMS/MFS/CCP Record to the PAO bean in the Record Mapper, and provide your own conversion. You also need to modify the method that maps in the reverse direction (from the PAO bean to the BMS/MFS/CCP Record).

**Logon Class:** The logon class is not required. You can go directly to the RuntimeContext and set the userid and password on the LogonInfo returned by `JavaRuntimeContext.getLogonInfo()`.

**Transaction Objects versus Commands and Navigators in Host On Demand:** In general, the transaction objects have been replaced with Commands and Navigators. To implement a `PAO::CRUD` in R1.3, you defined a TO. The TO defined the screen flow for the HOD session. In the TO, you defined the inputs and outputs and stages using transrecords, and the sequence in which these stages would be visited. For example, you might have defined a TO for insert; the insert TO would have transaction records defining all possible screens that a user could encounter expect as a Record was inserted in the CICS/IMS backend. You connected the Records according to how the screens flowed, and defined at each transaction record what input was expected to the screen, and what output was retrieved from the screen. You defined a method on the TO (for example, `createCustomer`, `updateCustomer`), which would handle errors. The `PAO::CRUD` instantiated the TO, then called this method.

To implement `PAO::CRUD` in R2.0, you define a Command or Navigator. A Navigator contains one or more Commands and is itself a Command (which can become part of yet another Navigator). The Navigator defines the transitional flow between Commands. For each Command there are inputs and outputs which are usually defined as Records (single line entry for one Command, Menu/DFHDGA for another, and so on). Each Navigator represents a specific flow, for example insert or remove. Each Command contains information such as the input and outputs of the screen that they came from, or are going to, as well as an interaction spec which can contain the function key, as well as how to map the fields on the screen with the properties of the PAO.

Navigators, which are full fledged beans, string these Commands into a sequence that CRUD the objects in the backend. The `PAO::CRUD` methods directly call the `execute()` method on the Command or Navigators. The interaction spec of the Commands defines the function key to press in order to navigate to the next screen (in R1.3, this was defined in the Transitions Tab of the transaction record). Where TOs had nodes, which represented screens in R1.3, and arrows, which represented interactions with the backend, Navigators have nodes which represent interactions with the backend, and arrows represent the order in which these interactions typically occur.

**Transaction Objects vs Commands and Navigators in ECI and APPC:** In R1.3, at least one TO was required for each `PAO::CRUD` method, and these TOs could have one or more transaction record. The TOs were similar, differing in the transaction record (the request type could be different, as could whether or not they collect data from, or emit data to, the business object). Also, the error checking differed between CRUD methods.

In R2.0, a minimum of one Command/Navigator is required for each of the PAO::CRUD methods. The Command has the COMMAREA Record as input and output and uses the Mapper Class to map the fields to the PAO properties.

**Communication Spec replaced by Connection Spec:** In the R1.3 unit test environment, the communication spec was initialized in the PAO constructor. In R2.0, the connection spec is externalized as a feature of the Commands and Navigators that are called from the CRUD methods.

**Additional Information:** This overview of the differences between Releases 1.3 and 2.0, is a summary for the purposes of planning your migration. Complete information can be found in the VisualAge for Java 2.0 Enterprise documentation, as well as the Component Broker Application Development Tools Guide.



---

## Chapter 3. Installing the CICS and IMS Application Adaptor on Windows NT

This chapter contains the following procedures for installing the CICS and IMS application adaptor for Component Broker.

- “Installing the Prerequisite Software”
- “Installing the Run Time” on page 40
- “Uninstalling the CICS and IMS Application Adaptor” on page 43

See the “Installation and Configuration” section of the *Late Breaking News* provided with Component Broker for important setup information.

---

### Installing the Prerequisite Software

This section contains information about and procedures for installing and configuring the software prerequisites, including:

- “Verifying the Installation of the Component Broker Run Time”
- “Installing the Communications Server” on page 32
- “Installing the CICS Transaction Gateway” on page 38

**Note:** If the CICS and IMS application adaptor is to access tier-3 systems that require a user ID and password, the Component Broker server must be a secure server.

### Verifying the Installation of the Component Broker Run Time

The CICS and IMS application adaptor requires the Component Broker server.

To verify that the Component Broker server is installed:

1. From the Windows NT **Start** menu, select **Run**. The Run dialog is displayed.
2. In this dialog, enter  
`regedit`

The Registry Editor window is displayed.

3. Expand HKEY\_LOCAL\_MACHINE → SOFTWARE → IBM → Component Broker, and select Packages.
4. Ensure that the server is installed.

If the server is not installed, the CICS and IMS application adaptor cannot be installed. For details on installing the Component Broker run time, see the Chapter on Installing IBM Component Broker for Windows NT in the *Component Broker for Windows NT and AIX Planning, Performance, and Installation Guide*.

### Important

Before using the CICS and IMS application adaptor:

- Ensure that CLASSPATH contains: jdk1.1.6\lib\classes.zip
- Ensure that INCLUDE contains: jdk1.1.6\include
- Ensure that LIB contains: jdk1.1.6\lib

If you update the CLASSPATH system variable, restart your system.

## Installing the Communications Server

Communications Server should be installed on an Intel-based computer running either Windows NT Server 3.51 or 4.0. To install the Communications Server, you must have a Windows NT administrator user ID with local authority.

Before beginning to install Communications Server:

- Close other application programs that you are running.
- If you have any version of Communications Server for Windows NT already running, stop it before starting to install.
- Communications Server should be installed prior to installing any version of the Personal Communications product (including the entry-level emulation program shipped with Communications Server). If Personal Communications is already installed on your server, remove it prior to installing Communications Server.
- Your machine must be running in VGA mode to install Adobe Acrobat. If you are running in another mode, change to VGA mode before starting to install.

Perform the following steps to install Communications Server for Windows NT:

1. Insert the Communications Server for Windows NT CD-ROM into the CD-ROM drive and follow the steps in the interface provided.

#### Notes:

- a. Windows NT Version 3.51 does not automatically launch the interface. If you are installing on a machine running this version, you must access the CD-ROM drive manually and run SETUP.EXE to launch the installation interface.
  - b. Use any editor to read the README.TXT file for the latest product notes. Online help is available throughout the installation procedure by clicking the Help button.
2. Click the Setup icon to begin the installation. When the Welcome to IBM Communications Server window displays, click the **Next** button to continue.
  3. In the Choose Destination Directory window:
    - a. The default installation directory is C:\IBMCS. To change the drive or directory, click the **Browse** button to open the Choose Directory window in which you can specify a different location.
    - b. In the Choose Directory window:
      - 1) Type a directory name with eight or fewer characters.
      - 2) Click the **OK** button to continue.
      - 3) If the specified directory does not exist, Setup displays a window asking if you want the directory to be created. Click the **Yes** button.

- c. The Choose Destination Location window is displayed again and lists the directory you specified. Click the **Next** button to continue to the Select Program Folder window...
  4. In the Select Program Folder window:
    - a. Accept the default folder or specify a different folder in which to install the Communications Server icons.
    - b. Click the **Next** button to continue.
  5. In this window:
    - a. Type at least one existing user ID to be added initially to the group. This establishes the IBMCSADMIN group, which allows authorized users to remotely configure and administer Communications Server.
    - b. Click the **Next** button to continue.

**Note:** You can add additional user IDs later using the Windows NT User Manager.
  6. In the Number of Concurrent Licenses window:
    - a. Type the number of concurrent user licenses that you have purchased.
    - b. Click the **Next** button to continue.
  7. In the Start Copying Files window, verify that all information is correct.
    - Click the **Back** button to review or change any information you previously entered.
    - Click the **Next** button to begin copying the Communications Server files onto your system.
  8. A horizontal Progress Bar displays indicating the progress of the file transfer. Once the Progress Bar appears, do not stop the installation procedure. After all the product files have been copied, the Installing NT Services window appears.

**Note:** During the copying procedure, there are vertical progress bars to the left of the window that further enable you to watch the installation progress. The progress bar on the far left monitors how much data remains in each file as it is being transferred; the progress bar in the middle shows the percentage of the installation files that have been copied; the progress bar on the right shows how much disk space remains for you to use during the installation process.
  9. At the end of the installation, a dialog box asks if you would like to install the IEEE 802.2 interface for the Local Area Network (LAN) using the IBM LLC2 protocol interface.
    - If you plan to use Communications Server over a LAN or if you are not sure, click the **Yes** button. The install program launches a network control window with instructions for configuring IBM LLC2 to operate over your LAN adapters.
    - If you do not plan to use Communications Server over a LAN, click the **No** button.
  10. Restart your computer.
- Note:** If you want to install the online documentation on an additional machine, such as a publications server, you can install it later by performing a drag-and-drop (using Windows Explorer) of the documentation files to the desired path. Once you have created icons in the appropriate folder, you will need to associate these files with the Adobe Acrobat reader. See the README.TXT file for more information.

## Configuring the Communications Server

Before you can start configuring your node to run Component Broker, you need to perform the following steps:

1. Have the remote node set up to run the backend CICS/IMS application with LU 6.2.
2. Gather all the information for your local node configuration.

For additional information about configuring the Communications Server, see the “Configure Communications Server” section of the *Component Broker for Windows NT and AIX System Administration Guide*.

Throughout this document, the SNA node where the Communications Server is to be configured to run Component Broker PAA/LU 6.2 will be referred to as the **CB node** or **local node**, whereas the node where the CICS/IMS backend application is run will be called **backend node** or **remote node**. The remote node may be a host machine located thousands of miles away, or a workstation sitting in the next office.

These steps are discussed in the sections following.

### Setting Up the Remote Node

If the backend application resides on a host machine, you need to request that the host admin set up the application and VTAM for LU 6.2.

If the backend node is a workstation for CICS applications, the node must have both IBM Communications Server and Encina PPC Gateway installed and configured to front the CICS system. As part of the Communications Server configuration for that node, the node must be designated as a network node.

In both cases, you should obtain the following pieces of information from the backend node admin:

- The TP names for the transaction programs that you intend to run on the backend node. Your mainframe counterpart may call these “tran codes.”
- The fully-qualified LU name to which the above TP names are associated. Note that this is a specific LU name, not the Control Point (CP) name for the remote node. Some may refer to this as the “APPL name” or “APPLID.”
- The mode names required by the remote transaction programs.

As an example, for the IMS Phone Book IVP set up at a Santa Teresa Lab host, the following information is obtained:

- The TP name is IVTNO
- The fully-qualified LU name is USIBMSTY.STY7IM16
- The mode name is L62MDE01

### Gathering Your Local Node Configuration Information

In this step, you need to gather information about the local node and its adjacent node.

The local node needs a unique CP name to identify itself to the network, and Communications Server uses the CP name for the node ID. Because of potential conflicts, you should not make up and implement a CP name yourself, so you would normally request it from a coordinator who has a more global view of the network. This coordinator could be the administrator of your gateway host, another administrator, or you could use a system utility.

If you are assigned an XID, you can use it (see “Configuring the Local Node” on page 36); otherwise do not request one. An XID consists of a 3-digit block number (for an NT node, this number is always 05D) followed by a 5-digit PU number. With this format, you can easily identify your XID, if one has been assigned to you.

**Adjacent node:** Node **B** becomes node **A**'s adjacent node if node **A** can specify a direct link to node **B**. **A** specifies a direct link to **B** by using **B**'s “hardware address” as the destination address in its own configuration. A direct link, as the name suggests, requires no apparent routing or name resolution and is the easiest kind of connection to set up and test.

**Hardware Addresses:** For workstations, the hardware address is indeed a hardware address, being a unique number burned into the token-ring card on the workstation. To find this number on an NT workstation, for example, on the **Start** menu, click **Programs** → **Administrative Tools** → **Windows NT Diagnostics**. On the dialog select the **Network** page, then click the **Transports** button to display the number.

For mainframe machines, any hardware addresses you obtain can, for convenience, be considered to be hardware addresses, although in reality there are often underlying mappings involved. Since a host may have several token-ring controllers attached, it is not uncommon for a single host to have multiple hardware addresses. For example, systems running VTAM normally use 12-digit numbers starting with “4000” as addresses. You can obtain host addresses from the responsible host administrator.

**Adjacent Nodes:** Every node in a network must have at least one adjacent node. If the adjacent node is the targeted backend node, then all is well. However, if the adjacent node is not the final destination, then it must be a gateway of some sort that can locate the final destination for the local node.

A node can be your adjacent node if you are able to use its hardware address in your connection specification. This is true in the following circumstances:

1. The node is a workstation running Communications Server and it is on the same LAN as your local node. One criterion for determining if the nodes are on the same LAN is whether the nodes in question can reach one another using NetBIOS. Another is whether they are physically attached to the same token-ring network.
2. The node is a host running VTAM and it belongs to the same net ID as your node.

The first circumstance is a possibility if one of the following is true:

- Your backend CICS applications reside on a workstation on your LAN that runs Communications Server and PPC Gateway.
- You do not have an existing gateway available to you and are setting up an APPN network node to serve as your gateway. This is a very unlikely scenario and its coverage is beyond the scope of this document.

The second circumstance is a possibility if one of the following is true:

- Your backend CICS/IMS applications reside on a host having the same net ID as yours.
- You are using a host gateway to reach the backend node.

The last two scenarios involving host VTAMs are the most common situation. Your VTAM administrator should be able to provide you with the network addresses for the relevant VTAM. It is quite possible that the same administrator is also responsible for allocating the CP name for your local node.

## Configuring the Local Node

To configure IBM Communications Server on the local node, you complete the steps in this section and verify these steps by starting the node operations.

You can have several different configurations for the local node, each recorded in an .acg file. This way you can experiment with several sets of different parameters if you so desire.

To start the configuration program, on the **Start** menu, click Programs → IBM Communications Server → SNA Node Configuration. This opens a new window. As you move the cursor across the menu bar items (for example, **File**, **Scenarios**, and so on), the corresponding menu automatically pops up. Click **File** → **New**. This automatically opens the **Scenarios** menu.

Select CPI-C, APPC or 5250 Emulation.... This scenario includes about ten groups of parameters to configure, and some groups are optional. All these groups are listed in the list window in the upper-left corner, starting with **Configure Node**. You need to configure only the following groups:

- Node
- Devices
- Connections

To configure your local node, perform the following steps:

1. Select the **Configure Node** line in the list window, then click the **New...** button. The Define the Node dialog opens.
  - a. On the **Basic** page:
    - 1) In the fields under **Fully qualified CP name**, enter the net ID and CP name for your local node.
    - 2) In the field under **CP alias** enter any name you like. This alias will be displayed prominently when you start the node operations. If you are experimenting with several configurations, it is a good idea to select a name that is indicative of this specific configuration, for example, trial1.
    - 3) If you have been given an XID, enter it under **Local Node ID**, otherwise leave the field at default. Notice that the XID you are given has two sections, of which the block ID should be 05D, a designation for all NT machines. If this is not so, the validity of the XID is questionable.
    - 4) Under **Node Type**, select End Node.
    - 5) Click the **Advanced** tab
  - b. On the Advanced page:
    - 1) Verify that the two option boxes under **Registration of LU resources** and the one under **Discovery Support** are checked.
    - 2) Click **OK**.
2. Highlight the **Configure Devices** line in the list window and click **New...** The Define a LAN Device dialog opens.
  - a. On the **Basic** page, you should be able to see LAN0\_04 for **Port name**, a number corresponding to your network adapter, such as 0 (representing IBM Shared RAM Token-Ring Adapter Driver), for **Adapter number**, and 04 for **Local SAP**. Do not change any fields.

**Note:** If any fields on this page are blank, Communications Server was probably not installed correctly. You must exit configuration and reinstall Communications Server.
  - b. Click **OK**.

3. Highlight the **Configure Connections** line in the list window, then click the **New...** button. The Define a LAN Connection dialog opens.
  - a. On the **Basic** page:
    - 1) Type a name in the **Link station name** field or accept the default. This name represents the connection you are currently defining.
    - 2) Verify that the **Device name** field contains the LAN0\_04 port name.
    - 3) Type the hardware address for the adjacent node that you obtained earlier into the **Destination address** field.
    - 4) Click the **Advanced** tab.
  - b. On the Advanced page:
    - 1) Verify that **APPN support** is checked. You should also check **Activate link at start**, because otherwise you will have to activate the connection manually after starting the node operations.
    - 2) If you intend to define multiple VTAM connections and have received different XIDs for those connections, enter the one for the connection being defined into the fields under **Local Node ID**, otherwise accept the defaults.
    - 3) Click the **Security** tab.
  - c. On the **Security** page:
    - 1) Optional: If you know the CP name for the adjacent node, enter the information into the fields under **Adjacent CP name**.
    - 2) For Adjacent CP type, Verify that APPN Node is displayed in the **Adjacent CP type** field.
 

**Note:** This configuration assumes that the adjacent node is either an APPN network node or a VTAM node that supports APPN.
    - 3) Click **OK**.
4. From the menu bar, click **File** → **Save As...** In the dialog that opens, specify the name of a file in which to save the configuration. You should use the node alias as the file base name (that is, the file extension should be .acg). For example, you could specify a file name like trial1.
5. From the menu bar, click **File** → **Exit** to exit the SNA Node Configuration application.
6. From the **Start** menu, click **Programs** → **IBM Communications Server** → **SNA Node Operations**. The Communications Server Node Operations window opens. Identify the tool bar buttons by moving the cursor slowly across the buttons to see the hover annotation for each button. Locate the **Start**, **Stop**, **Node**, and **Connections** buttons.

In the Communications Server Node Operations window:

- a. Click **Start**. A file dialog opens. Enter the name of the file in which you previously saved your configuration, for example, trial1.acg.
- b. The client area of the window initially displays the node data. Click **Connections** to display the connections data.

If both the **State** and **Sub-state** columns show *Active*, your configuration is working. If you see *Pending...* in these columns, click **Node**, then **Connections** again. You can also right-click the link name LINK0000 and select **Start**.

If the columns still display *Pending...*, but the adjacent node is up and running, there could be a problem with your configuration that must be corrected before continuing.

- c. The **Destination Address** column displays the hardware address you entered earlier. The information displayed in the **Adjacent CP Name** column is important and should be recorded for later reference.
- d. Scroll the client area horizontally. The **Adjacent CP Type** displays LEN/EN, where the LEN stands for “low-entry networking node,” the lowest form of all nodes. This label is APPN's designation of a host node running VTAM, even if the VTAM version supports APPN, and does not represent a problem.
- e. Click **Stop** and close the window to complete the configuration.

**Note:** You should consider backing up your configuration files periodically in case a problem occurs with your configuration. By default, your configuration files (.acg files) are located in the \private subdirectory of your Communications Server installation directory.

## Installing the CICS Transaction Gateway

**Note:** If Visual Age for Java 2.0 has already been installed, or will be installed on this machine, skip this section and proceed to the Configuring the CICS Client section. Visual Age for Java installs the Transaction Gateway by default. Installing the CICS Transaction Gateway in combination with Visual Age for Java may result in two copies of the Gateway being installed with unfavorable results.

The CICS and IMS application adaptor uses the CICS client for its External Call Interface (ECI) support. Should that support be required, the CICS client can be installed as follows:

- On the computer used for the Component Broker server. In this case, the CICS and IMS application adaptor loads the Transaction Gateway DLL directly.
- On a different computer. In this case, the Transaction Gateway daemon, when started on its computer, must listen at a specific port number. See “Starting the Transaction Gateway” on page 42 for information about starting the gateway.

Perform the following procedure to install the CICS client.

1. If you have a previous version of the CICS Transaction Gateway on your system, uninstall it and remove \java\JGate\classes from your CLASSPATH user environment variable, and remove \java\JGate\bin\nt from your PATH user environment variable.
2. Insert the CICS and IMS Application Adaptor compact disc into the CD-ROM drive.
3. Display the contents of the compact disc.
4. Double-click the CICSCLI folder to display its contents.
5. Double-click the WinNT folder to display its contents.
6. Double-click on ctgnt.exe to start the install procedure. This package contains an executable version of the Install program for the CICS Transaction Gateway Version 3.0.
7. At the next window, you are asked if you want to unpack the contents. Click the **Finish** button to continue the installation and the installShield is displayed.
8. In this window, click the **Next** button to install the CICS Transaction Gateway on your system.
9. Click the **Yes** button to accept the terms of the license agreement and continue.
10. If the only CICS Client version 2.x is installed, it will detect it and prompt you to uninstall it. If so, click the **Next** button to have it automatically uninstalled.
11. For the uninstall, a window is displayed listing the installed components. Click the **Select all** button to select all features, and then click the **Delete** button.

12. A destination location will be selected. You can change the location and then click the **Next** button to continue.
13. Accept the typical install and then click the **Next** button to continue.
14. Accept the default program folder and then click the **Next** button to continue.
15. Accept the update path and install and service, and then click the **Next** button to continue. The code will be installed.
16. A window is displayed asking if TCP62 is desired. Either click the **Next** button to continue, or check the **Yes** box, and then click the **Next** button to continue.
17. Click the **Finish** button.

## Configuring the CICS Client Within the Transaction Gateway

The configuration of a CICS client is dependent on the type of CICS server. To determine your CICS client configuration information, review the CICSCLI.INI file in the <cics\_install\_directory>\BIN directory. You must modify this file before using the CICS client. If you create a copy and modify it, you need to set the CICSCLI environment variable to point to the new file.

The client configuration file specifies:

- The name of your CICS server
- The type of communication protocol to use
- Other parameters relating to the communication between the CICS clients and server

The configuration file is split into the following sections:

- One for the client
- Multiple sections for each server defined
- Multiple sections for each driver defined.

The client section starts with the keyword "Client=\*" and is followed by a series of variable-value pairs that specify the client configuration. Add the following line under the Client= stanza in the CICSCLI.INI file: DceCellDirectory = N

The server section starts with the keyword "Server=xyz," where xyz is one of the following:

- CICSTCP** For TCP/IP communication
- CICSNETB** For NetBIOS communication
- CICSSNA** For SNA communication
- CICST62** For TCP6.2 communication

Each section documents the specific information that is required for each protocol. The xyz is an arbitrary name that you want to use for the client to refer to that specific CICS server. Some people name xyz to be the CICS region name to which they are connecting.

You need to modify the server section of the configuration file. For information on how to correctly specify the server configuration, review the code comments within the configuration file, or see the *CICS Client Administration Guide*.

For example, If you are using a CICS server with a TCP/IP listening port configured, modify the *NetName* parameter to be the IP address or hostname of the CICS server, and modify the *Port* parameter to be the listening port number (or zero (0) to use the default of 1435).

## Notes:

1. There is a difference between the listening port that the CICS Server uses for connections through the CICS Client, and the TN3270 listening port defined for TN3270 clients (such as IBM eNetwork Host On-Demand).
2. All the lines in the server section of the configuration file, except the ones following the Server=CICSTCP, are commented out. TCP/IP protocol is the default. To specify a different protocol, comment out the TCP/IP information and uncomment the desired protocol. You must change the CICSTCP to specify your CICS server name. Choose CICSNETB, CICSSNA, or CICST62 as appropriate.
3. Add the following line under the Client = stanza in the CICSCLI.INI file: DceCellDirectory = N ; Do not check for DCE on the system.

---

## Installing the Run Time

This section discusses the installation of the CICS and IMS application adaptor run time.

### Pre-Installation

Before installing the CICS and IMS application adaptor, disable any antiviral programs, unless you are using IBM AntiVirus 3.0. Failure to do so could result in an incomplete installation. After completing the CICS and IMS application adaptor installation, you can reactivate your antiviral programs.

If you have not rebooted your system since installing the Component Broker run time, reboot your system before installing CICS and IMS application adaptor run time.

### Installation

Perform the following steps to install the CICS and IMS application adapter option:

1. Ensure that the jar files ctgclient.jar and ctgserver.jar are in your classpath.
2. Insert the CICS and IMS Application Adaptor compact disc into your CD-ROM drive. If your system autoloads the compact disc, you can skip to step 5.
3. Display the contents of the compact disc. Change the directory to winNT.
4. Start the installation by clicking the Setup icon. Three Setup icons are displayed in the contents list. To install the CICS and IMS application adapter, you must click the Setup icon that looks like a computer. The Welcome window is displayed.
5. On the Welcome window, click the **Next** button. A message is displayed indicating that Setup is searching for Component Broker packages on your system. An informational message states that Component Broker packages were found and lists the directory in which they were found.
6. In the Verify Configuration Setting window, verify that the items to be installed are correct.
  - Click the **Back** button to review or change any information you previously entered.
  - Click the **Next** button to start the installation.
7. On completion of the installation, a window displays stating that the installation has completed. When you exit this window, your computer automatically restarts. Optionally, in this window you can select check boxes to:
  - Automatically launch the configuration tool when the computer restarts.
  - Automatically display the readme file when the computer restarts

Click the **Finish** button to close the installation program and to restart your computer.

## Configuring the CICS and IMS Application Adaptor

Perform the following steps to configure your CICS and IMS application adaptor installation:

1. If your computer starts the configuration tool automatically when you restart your computer after performing the installation, skip to the next step. Otherwise, start the configuration tool from the Windows NT **Start** menu and select **Programs** → **IBM Component Broker** → **Component Broker Configuration tool**. The Configure New Install window opens.
2. In the Configure New Install window:
  - a. Select the **Yes** radio button to configure your additional Component Broker installation.
  - b. Click the **Next** button to continue.
3. In the Verify Configuration Setting window, verify that the items to be installed are correct.
  - Click the **Back** button to review or change any information you previously entered.
  - Click the **Next** button to start the configuration.
4. The Configuration Status window displays information about the items being configured.
5. An informational window is displayed stating when the configuration is completed. Click the **OK** button to close the window.
6. Exit the Component Broker Configuration tool from the Configuration Status window by clicking either the **Close** button or the **Cancel** button.

### Important

Follow the instructions in the Chapter on “Installing the Development Environment” in the Component Broker for Windows NT and AIX *Component Broker for Windows NT and AIX Planning, Performance, and Installation Guide* for details on installing the required development software for the CICS and IMS application adaptor.

For CICS and IMS development, you must install the CICS and IMS Application Adaptor SDK and in addition to the other development software required by your configuration (determined through the planning of your Component Broker network).

To work with the CICS and IMS samples detailed in this book, you must install the Samples. The CICS and IMS Application Adaptor compact disc contains only the run-time environment. All Component Broker development software is contained on the CBToolkit compact disc.

After installing the run time and development portions for the CICS and IMS application adaptor, you can begin developing your own CICS and IMS application adaptor applications. This document contains samples in the following chapters, which you can walk-through to learn about developing applications that access CICS and IMS backend systems.

- Chapter 5, “Developing an IMS-HOD Application” on page 55 contains a sample of an IMS-HOD application
- Chapter 6, “Developing a CICS-HOD Application” on page 101 contains a sample of a CICS-HOD application
- Chapter 7, “Developing a CICS-ECI Application” on page 145 contains a sample of a CICS-ECI application

---

## Starting the Transaction Gateway

To use the networked Transaction Gateway from a Component Broker server, the Transaction Gateway daemon must be started. Use the following instructions to start the Transaction Gateway:

1. Change your directory as follows:

```
cics_install_directory\java\JGate\bin\nt
```

2. Start the gateway. Enter:

```
JGATE -port=nnnn
```

Where *nnnn* represents the port number on which you decide to listen.

The Transaction Gateway is now ready to be used.

---

## Reverting to the Original Install State

If you revert to the original installed state for Component Broker, you also need to perform the steps in this section to revert to the original state for your CICS and IMS application adaptor.

If the System Manager is stand-alone and after you have reverted back to your original Component Broker installed state:

1. Load the sompaa.ddl file into System Manager.
2. Copy the ipaa services into the sample configuration.

The system is now reset to its original install state.

When the System Manager is not stand-alone (there is a remote System Management Agent), the host containing the System Manager can be reset using the previous procedure, but any host containing a System Management Agent must be uninstalled and reinstalled.

---

## Uninstalling Communications Server for Windows NT

To uninstall Communications Server, perform the following steps:

- For Windows NT 3.51:
  1. Double-click the Uninstall icon in the IBM Communications Server folder.
  2. Follow the procedures provided.
- For Windows NT 4.0:
  1. Double-click the Control Panel icon.
  2. Double-click the Add/Remove Programs icon.
  3. Double-click your Communications Server folder in the provided list.
  4. Click the **Remove** button.
- If you have the IBM LLC2 protocol interface installed, a help panel helps you uninstall the LLC2 protocol.

---

## Uninstalling the CICS and IMS Application Adaptor

There are two ways to uninstall CICS and IMS application adaptor.

- From the Windows NT **Start** menu, select **IBM Component Broker CICS and IMS Application Adaptor for Windows NT → Uninstall**.
- From the Control Panel, select Add/Remove Programs → IBM Component Broker CICS and IMS Application Adaptor for Windows NT.

The uninstall program confirms your intentions before the uninstall. When the uninstall completes, restart your system.



---

## Chapter 4. Installing the CICS and IMS Application Adaptor on AIX

This chapter contains the following procedures for installing the CICS and IMS application adaptor for Component Broker on AIX.

- “Installing the Prerequisite Software”
- “Uninstalling the CICS and IMS Application Adaptor” on page 53

See the Installation and Configuration section of the *Late Breaking News* provided with Component Broker for important setup information.

---

### Installing the Prerequisite Software

This section contains information about and procedures for installing and configuring the software prerequisites, including:

- “Installing and Configuring the CICS Transaction Gateway” on page 50
- “Installing the Communications Server” on page 47

**Note:** If the CICS and IMS application adaptor is to access tier-3 systems that require a user ID and password, the Component Broker server must be a secure server.

### Installing VisualAge for Java 2.0 for AIX

To install VisualAge for Java, perform the following procedure:

1. Log in as root (if you are not already logged in)..
2. Insert the VisualAge for Java compact disc into the CD-ROM drive.
3. At a shell prompt, enter: `smitty install_latest`. The Install and Update from LATEST Available Software screen is displayed.
4. On this screen:
  - a. In the INPUT device/directory for software field, press the **F4** key to display a list of available devices from which you can select the appropriate input device (`/dev/cd0`) and press the **Enter** key. The screen is redisplayed with additional fields.
  - b. In the SOFTWARE to install field, press the **F4** key to display a list of the software packages contained on the compact disc.
  - c. It is necessary to install some support software before installing VisualAge for Java. In the SOFTWARE to install list, scroll down and select:  
`IMNSearch.rte`  
`xIC.rte`
  - d. Press the **Enter** key. The Install and Update from LATEST Available Software screen is redisplayed, but the software packages selected are in the SOFTWARE to install field.
  - e. Ensure that the **AUTOMATICALLY install requisite software?** and **EXTEND file systems if space needed?** fields are yes.
  - f. Press the **Enter** key. The ARE YOU SURE? screen is displayed.
  - g. On this screen, press the **Enter** key to continue.

h. When the install has completed, it should show a command status of OK in the upper left corner of the screen. Press the **F3** key to return to the Install and Update from LATEST Available Software screen.

i. In the SOFTWARE to install field, press the **F4** key to display a list of the software packages contained on the compact disc.

j. In the SOFTWARE to install list, scroll down and select the following:

- IBMJava.builder
- IBMJava.corba
- IBMJava.dab
- IBMJava.hpj
- IBMJava.ide
- IBMJava.lic
- IBMJava.remote-debugger
- IBMJava.remote-jdebugger
- IBMJava.repository
- IBMJava.toolextensions
- IBM\_Connectors.CICS
- IBM\_Connectors.ENCINA

**Note:** You can optionally install the documentation.

k. Press the **Enter**. The Install and Update from LATEST Available Software screen is redisplayed, but the software packages selected are in the SOFTWARE to install field.

l. Ensure that the **AUTOMATICALLY install requisite software?** and **EXTEND file systems if space needed?** fields are yes.

m. Press the **Enter** key. The ARE YOU SURE? screen is displayed.

n. On this screen, press the **Enter** key to continue.

5. When the install has completed, it should show a command status of OK in the upper left corner of the screen. Press the **F10** key to exit.

## Configuring VisualAge for Java 2.0 for AIX

**Note:** VisualAge for Java (VAJ) 2.0 for AIX contains a defect that prevents the software from starting properly. As a temporary workaround, put the following line in the file .Xdefaults in the user's home directory on the AIX machine where the X server runs (notice that VAJ itself may be running on another machine). If the file does not exist, create it.

```
IBMJava.font: -adobe-utopia-regular-r-normal--0-0-75-75-p-0-iso8859-1
```

The configuration mainly consists of creating a standalone work environment for the user. You will go through the following steps the first time you start VAJ.

1. Log in to the AIX machine where VAJ is installed using your user ID (CB userID).
2. From an AIX command prompt, enter VAJIDE.
3. The program will prompt you to specify a directory in which to store your work environment. Select a directory that has sufficient disk space required by the environment. The program will show the space requirement.

4. You will be asked to choose between standalone and team environment. Choose standalone. It may take some time before the next prompt appears.
5. In this dialog, select the workspace owner. In the list box, only Administrator is listed. Select it and click the **OK** button. It may take some time before the next prompt appears.
6. In the next dialog, enter a network name. You may enter <userid>, and click the **OK** button.
7. It may take some time to load all of the features into the repository. A message window will display the progress. The workbench window will display when the installation is completed.

You should now have a directory named IBMJava in your home directory, for example /home/<userid> and a directory that you specified in step 3 that contains the VAJ repository and other information associated with <userid>.

## Installing the Communications Server

This section describes the steps necessary to install Communications Manager. The three main steps in the process are as follows:

1. Installing the software bundle definitions
2. Installing the software
3. Installing the support for your communications adaptor

### Installing the Software Bundle Definitions

1. If you are not logged in, log in as root.
2. Insert the Communications Server compact disc into your CD-ROM drive.
3. From the shell prompt, type: `smitty easy_install_bundle`. The Install Software Bundle (Easy Install ) window is displayed.
4. In this window:
  - a. Press the **F4** key to receive a prompt for the input device.
  - b. Select the CD-ROM drive containing the Communications Server compact disc and press the **Enter** key. The Select Fileset Bundle window is displayed.
  - c. Select Media-Defined and press the Enter key to continue.
  - d. Press the **Enter** key again to display the ARE YOU SURE? window.
  - e. Press the **Enter** key to begin the install.
  - f. Press the **F10** key to exit SMIT and return to the shell prompt.

### Installing the Software

1. If you are not logged in, log in as root.
2. Insert the Communications Server compact disc into your CD-ROM drive.
3. From the shell prompt, type: `smitty easy_install_bundle`. The Install Software Bundle (Easy Install ) window is displayed.
4. In this window:
  - a. Press the **F4** key to receive a prompt for the input device.
  - b. Select the CD-ROM drive containing the Communications Server compact disc and press the **Enter** key. The Select Fileset Bundle window is displayed.

- c. Select **Communications** and press the **Enter** key to continue.
- d. Press the **Enter** key again to display the ARE YOU SURE? window.
- e. Press the **Enter** key to begin the install.
- f. Press the **F10** key to exit SMIT and return to the shell prompt.

## Installing the Communications Adaptor Support

Since Communications Server supports several communications adaptors, you must install the support for the adaptor that you are currently using. For example, you may be communicating via a Token-Ring adaptor for your Component Broker applications. The communications support for that adaptor must be installed separately from the general communications support.

To install the software support for your communications adaptor, perform the following steps:

1. If you are not logged in, log in as root.
2. Insert the Communications Server compact disc into your CD-ROM drive.
3. From the shell prompt, type: `smitty install_latest`. The Install and Update from LATEST Available Software window is displayed.
4. In this window:
  - a. Press the **F4** key to receive a prompt for the input device.
  - b. Select the CD-ROM drive containing the Communications Server compact disc and press the **Enter** key. The Install and Update from LATEST Available Software window is displayed again.
  - c. In this window:
    - 1) Press the **F4** key on the SOFTWARE to install field to display a list of the software on the compact disc.
    - 2) Under the bos.dlc section, select your communications adaptor by positioning the cursor to the left of the appropriate adaptor and pressing the **F7** key. For example, to install Token-Ring support, select **Token-Ring Data Link Control**.
    - 3) Under the sna.instdlc section, select your communications adaptor by placing the cursor to the left of the appropriate adaptor and pressing the **F7** key. For example, to install Token-Ring support, select **Communications Server for AIX Token-Ring DLC Inclusion Fileset**.
    - 4) Press the **Enter** key to continue.
  - d. Press the **Enter** key to continue again and the ARE YOU SURE? window is displayed.
  - e. Press the **Enter** key to begin the install.
  - f. Press the **F10** key to exit SMIT and return to the shell prompt.

## Configuring the Communications Server

This guide assumes that either (1) your target CICS or IMS transactions run on a host machine that is directly accessible from your node, or (2) your node reaches those transactions through a host gateway to which your node has direct link. As far as configuring your node for APPC communications is concerned, the host systems in both cases can be treated the same. In the following text, the term host refers to such a host running VTAM (the mainframe version of an SNA stack).

Before you proceed, obtain the following information from your network system administrator:

1. The fully-qualified CP name for your AIX node (it looks like "MYNETID.MYCPNAME")

2. The MAC address for the host VTAM. This is normally a 12-digit number starting with "4000"

For additional information about configuring the Communications Server, see the "Configure Communications Server" section of the *Component Broker for Windows NT and AIX System Administration Guide*.

A Communications Server utility, `xsnaadmin`, can be used to configure and run Communications Server. From an aixterm window, type `xsnaadmin`.

## Configuring the Node Parameters

At the far-right end of the toolbar, there should be an icon, with legend "<hostname> Unconfigured" inside.

1. Login as root.
2. From the menubar select **Services** → **Configure node parameters...** A dialog will pop up. In the dialog...
3. For APPN support, select **Network node**.
4. In the pane entitled SNA addressing, enter your node's fully-qualified CP name in the two fields labeled **Control point name**. For Control point alias, you may keep the default, which should be your hostname, or enter a name that you choose.
5. Leave everything else at default and click the **OK** button.

Your node is defined, but it remains inactive now, as its icon will show.

## Defining a Connection

1. From the menubar select **Selection** → **New...**
2. In this dialog, the **Port using** radio button should already be selected.
3. From the pull-down selection list following Port using, select the network adaptor type on your AIX node. The default is Token ring card.
4. Click **OK**. The dialog closes while another dialog pops up.

You now have a port defined. In the client area of the `xsnaadmin` window, you should see a top pane entitled **Connectivity and dependent LUs**. Under the title there should be an item named, for example, TRSAPO, currently inactive. Select this item if it is not already selected.

5. From the menubar select **Selection** → **New...**
6. In this dialog, the **Link station to port TRSAPO** radio button should already be selected.
7. Click **OK**. The dialog closes while another dialog pops up. Make the following entries in the dialog:
  - a. For Activation, select On node startup from the pull-down selection list.
  - b. Ensure that for **Remote node type**, **Discover** is selected, and for **Remote node role**, Host is selected.
  - c. In the field labeled **MAC address**, enter the host MAC address you obtained from your system admin.
  - d. Click **OK**.

## Defining the Partner LU

1. Select the node icon at the far right side of the toolbar with the legend <hostname> Inactive inside. Click the **Start** button at the left end of the toolbar. You should see the legend inside the node icon change to "<hostname> Active", and in the top pane of the client area, you should see the item TRSAPO and the item under it, TRL0, show their status as Active also.
2. On the lower portion of the client area, you should see a pane entitled Remote system. The top line in the pane should show the fully-qualified CP name for the host VTAM that you configured in the "Defining a Connection" section, (you only entered the MAC address, and SNA determined the CP name for you).

**Note:** If your remote system name is not listed on the Remote systems pane, you can add it by performing the following procedure, but first, you must know the name of your gateway host.

- a. Click on the **Remote systems** title.
- b. From the toolbar, click the **Add** button.
- c. Click the **Define remote node** radio button.
- d. At the **Remote node** window, enter the host network name.
- e. Click the **OK** button to close the dialog box.
- f. You will receive an informational message that the LU has been created. Click the **OK** button to close the dialog box. Your host VTAM name should now appear in the list of Remote systems.

You can now continue with the following steps.

- a. Highlight this CP name
- b. From the menubar, select **Selection** → **new**
- c. In the dialog that pops up, ensure that the radio button Define partner LU on node MYNETID.MYGTWY is selected. Click the **OK** button.
- d. On the next dialog, enter the fully-qualified partner LU name in the field labeled **Partner LU name**, and ensure MYNETID.MYGTWY appears in the field following the push button Location.
- e. Click the **OK** button. This completes the Communications Server setup.

## Installing and Configuring the CICS Transaction Gateway

This software comes in the form of a compressed archive file. Locate the file ctg301a.tar.Z on one of the CDROMs that come with the Component Broker package. Follow these steps to install and configure the software:

1. If you have a previous version of the CICS Transaction Gateway on your system, uninstall it and remove `\java\JGate\classes` from your CLASSPATH user environment variable, and remove `\java\JGate\bin\nt` from your PATH user environment variable.
2. Login to the target machine as root.
3. Create a temporary directory as working space. There should be at least 25MB free space. Copy ctg301a.tar.z from the CDROM to the working directory, naming it ctg301a.tar.Z
4. Uncompress the file ctg301a.tar.Z and extract the archived files to the `/usr/lpp/ctg` directory by entering the following two commands in an aixterm window:

```
uncompress ctg301a.tar
tar -xvf ctg301a.tar
```

5. Set up all the necessary links by changing to the directory `/usr/lpp/ctg` and issue the following command:

```
mkcicscli
```

6. Now change to the directory `/usr/lpp/ctg/bin` and edit the file `CICSCLI.INI`. Before editing the file, you can optionally save the original file by using another name such as `CICSCLI.INI.ORIG`. You may need to change the permissions for `CICSCLI.INI` by entering `chmod 777 CICSCLI.INI`. This step is not mandatory and is useful only if you have a default CICS server that you regularly connect to. The server section of the file looks like the following:

```
Server = myserver ; Arbitrary name for the server
Description = TCP/IP Server ; Arbitrary description for the server
Protocol = TCPIP ; Matches with a Driver section below
NetName = myhost.ibm.com ; The server's TCP/IP address
Port = 0 ; Use the default TCP/IP CICS port
```

Enter a name for the Server. Specify the fully qualified server host name for the NetName. If the System Administrator of your CICS server uses a nonzero port, enter the port number for Port.

7. Add the following line under the Client = stanza in the `CICSCLI.INI` file:

```
DceCellDirectory = N; Do not check for DCE on the system
```

8. Ensure that the path `/usr/lpp/ctg/bin` is included in both environment variables `LD_LIBRARY_PATH` and `LIBPATH` after this package and the Component Broker are both installed. If not, append it to these variables as appropriate (for example, through definitions in your `.profile` or `.kshrc` file).
9. To test your installation, type `cicsterm` in an aixterm window. You should have started a 3270 session with the CICS server that you specified in `CICSCLI.INI`.

**Note:** The first time an UNSECURE CB server on AIX tries to initiate a connection to a CICS region, a userid/password is required by default to make the connection. Therefore, on the AIX machine where the CICS client is installed, run the following command:

```
/usr/lpp/ctg/bin/cicscli /C=server /U=userid /P=password
```

where `server` = name of the CICS server as defined in the `CICSCLI.INI` file, and `userid` and `password` are the userid/password combination that will be used in accessing the CICS server.

---

## Installing the CICS and IMS Application Adaptor

1. Insert the CD for the Application Adaptor for CICS and IMS into the CD-ROM drive.
2. Login as root and mount the CD-ROM file system as follows:

```
mount /cdrom
```

3. Type `smitty install_latest`

The Install and Update from the LATEST Available Software window is displayed.

4. From this window:

- a. In the **INPUT device/directory for software** field, type `/cdrom/aix` and C. This window is redisplayed with additional fields.
- b. In the **SOFTWARE to install** field, press the **F4** key to display a list of software packages contained on the CD.
- c. In the **SOFTWARE to install** list, scroll down to the entry containing: `CBPAA.TYPICAL ALL`
- d. Press the **F7** key to select the entire package.

- e. Press the **Enter** key. The Install and Update from LATEST Available Software window is redisplayed, but the software package selected is in the **SOFTWARE to install** field.
- f. Ensure that the **AUTOMATICALLY install requisite software?** and **EXTEND file systems if space needed?** fields are **yes**.
- g. Press the **Enter** key. The ARE YOU SURE? window is displayed.
- h. From the ARE YOU SURE? window, press the **Enter** key to start the installation.

The installation process begins. After the files are copied, the command status, as indicated in the upper left corner of the screen, is OK. When you receive this OK status, all files were copied and the system is ready to be configured. Press the **F10** key to exit smit. The files are installed in the /user/lpp/CBPAA directory.

**Note:** If any errors occurred, view the \$HOME/smit.log file. If the cause of the error is not clear, save a copy of the smit.log file and report the problem to your IBM representative.

---

## Configuring the CICS and IMS Application Adaptor

Login as root. At the command prompt, type `smit ty apps`

1. Select **Configure CBPAA (CICS and IMS Application Adaptor)** and press the **Enter** key.
2. Select **Y** to configure Application Adaptor for CICS and IMS. If the field is not set to **Y**, use the **Tab** key to change the value to **Y**.
3. Press the **Enter** key to begin configuration.

The configuration is completed when the command status, as indicated in the upper left corner of the screen, is OK.

4. Press the **F10** key to exit smit.
5. Log out as root

---

## Installing the Development Environment for the CICS and IMS Application Adaptor

1. Insert the CBToolkit CD into the CD-ROM drive.
2. Login as root
3. Mount the CD-ROM file system as follows:
 

```
mount /cdrom
```
4. At the command prompt, type `smit ty install_latest`. The Install and Update from the LATEST Available Software window is displayed. From this window:
  - a. In the **INPUT device/directory for software** field, press the **F4** key to display a list of available devices from which you can select the appropriate input device (/dev/cd) and press the **Enter** key. This window is redisplayed with additional fields.
  - b. In the **SOFTWARE to install** field, press the **F4** key to display a list of software packages contained on the CD.
  - c. In the SOFTWARE to install list, scroll down to the entry 2.0.0.0 for CBToolkit.T00LKIT (CICS\_IMS\_ApplicationAdaptorSDK) and press the **F7** key to select this package.
  - d. Press the **Enter** key. The Install and Update from LATEST Available Software window is redisplayed, but the software package selected is in the SOFTWARE to install field.

- e. Ensure that the **AUTOMATICALLY install requisite software?** and **EXTEND file systems if space needed?** fields are **yes**.
- f. Press the **Enter** key. The ARE YOU SURE? window is displayed.
- g. Press the **Enter** key to continue.

The installation process begins. After the files are copied, the command status, as indicated in the upper left corner of the screen, is OK. When you receive this OK status, all files were copied and the system is ready. Press the **F10** key to exit smit. The files are installed in the /user/lpp/CBToolkit directory.

**Note:** If any errors occurred, view the \$HOME/smit.log file. If the cause of the error is not clear, save a copy of the smit.log file and report the problem to your IBM representative.

- h. Log out as root

---

## Uninstalling the CICS and IMS Application Adaptor

Login as root. At the command prompt, type `smit remove`. The Remove Installed Software window is displayed.

1. In the **Software name** field, type CBPAA to remove the runtime and development environment for the CICS and IMS Application Adaptor.
2. In the **PREVIEW only?** field, if the value is not **n**, use the **Tab** key to change the value to **n**.
3. In the **REMOVE dependent software** field, if the value is not **y**, use the **Tab** key to change the value to **y**.

**Note:** CBToolkit.TOOLKIT.CICS\_IMSApplicationAdaptorSDK is also removed.

4. Press the **Enter** key to remove the CICS and IMS Application Adaptor from your system.

The uninstallation process begins. After the files are removed, the command status, as indicated in the upper left corner of the screen, is OK. When you receive this OK status, all files were removed. Press the **F10** key to exit smit.

5. Log out as root.

---

## Environment Setup

Use the same user name to run Component Broker on AIX to run Component Broker CICS and IMS Application Adaptor on AIX. Follow the steps in the Configuring Your Component Broker User ID section in the Installing Component Broker for AIX chapter in *Component Broker for Windows NT and AIX Quick Beginnings* to properly configure the environment.



---

## Chapter 5. Developing an IMS-HOD Application

This chapter provides information for building a sample Component Broker application with an IMS backend. This information, for the most part, can be pertinent for building a CICS-backend Component Broker application.

This chapter contains the following information.

- “The IMS Sample Application”
- “Enterprise Access Builder Procedures” on page 58
- “Developing an IMS-HOD Business Object” on page 88

**Note:** To walk through this sample, the following software and Component Broker software must be installed on your system:

- The Component Broker samples
- The CICS and IMS Application Adaptor SDK
- IBM VisualAge Java with EAB

### Important Information

Before walking through this sample, please refer to the *Late Breaking News* provided with Component Broker before performing the exercise in this chapter. This document provides the latest information regarding the CICS and IMS application adaptor samples, which may differ from the instructions for this sample application.

---

## The IMS Sample Application

The IMS-HOD sample application is based on an IMS Installation Verification Procedure (IVP). The IVP is a mock phone book database, where each entry in the phone book contains the following fields:

- Last name
- First name
- Phone number extension
- Internal zip code

This sample application works on an IMS database and permits adding, inquiring, updating, and deleting of phone book entry records through the ADD, DISPLAY, UPDATE, and DELETE transactions.

Although this sample application is not a full-blown IMS application, it captures the essence of an application involving multiple 3270 panel navigation and delivering some amount of business function. This sample application can be extended and customized to explore different IMS-HOD application issues.

**WIN** The sample that you build in this section is included with the product and can be built by following the steps in the HTML file in:

`CBroker\samples\InstallVerification\PAA\readme.htm`

**AIX** The sample that you build in this section is included with the product and can be built by following the steps in the HTML file in:

`$HOME/samples/InstallVerification/PAA/readme.htm`

### Important Information

The IMS sample provided in this chapter uses a TCP/IP-based emulation to communicate with the IMS applications running on mainframe machines. For the 3270 emulator to communicate with the mainframe, the following requirements must exist:

- A TCP/IP link must exist between your Windows NT node and the host machine where your IMS application is running
- A TN3270 listener must be defined to the host machine

Contact your systems administrator for additional details about installing and configuring these requirements.

## Interacting with the IMS IVP

Using the IMS IVP involves navigating a sequence of 3270 panels. For this sample, the application name is "APPL8". After logging in to the IVP, one of the transaction paths can be started. The following sequence completes a full-cycle for the ADD process code.

1. Type:

```
/FOR IVTCC
```

the transaction is started, and the next panel is displayed.

2. On this panel, the following entry fields are displayed:

- **PROCESS CODE**
- **LAST NAME**
- **FIRST NAME**
- **EXTENSION NUMBER**
- **INTERNAL ZIP CODE**

a. Type ADD in the **PROCESS CODE** field.

b. Complete the other fields as appropriate.

```
*****                                     IMSTERM1
*      IMS INSTALLATION VERIFICATION PROCEDURE      *
*****

                                TRANSACTION TYPE : CONVERSATIONAL
                                DATE              : 12/10/97

PROCESS CODE (*1) :  add
LAST NAME        :  smith
FIRST NAME       :  john
EXTENSION NUMBER :  8-555-1234
INTERNAL ZIP CODE :  sm9876

                                (*1) PROCESS CODE
                                ADD
                                DELETE
                                UPDATE
                                DISPLAY
                                TADD
                                END

                                SEGMENT# :
```

c. Press the **Enter** key.

This displays a new panel.

- This panel contains a success or failure message in the lower-left corner of the screen. If the message reads:

ENTRY WAS ADDED

the entry was successful. If the message reads otherwise, the entry was not added.

```

*****
*      IMS INSTALLATION VERIFICATION PROCEDURE      *
*****
                                                    IMSTERM1
TRANSACTION TYPE : CONVERSATIONAL
DATE              : 12/10/97

PROCESS CODE (*1) : ADD
LAST NAME         : SMITH
FIRST NAME        : JOHN
EXTENSION NUMBER  : 8-555-1234
INTERNAL ZIP CODE : MS9876

(*1) PROCESS CODE
ADD
DELETE
UPDATE
DISPLAY
TADD
END

ENTRY WAS ADDED                               SEGMENT# : 0001

```

- Click the **Clear** button. This displays a blank screen.

**Note:** Depending on which emulator you are using and its keyboard mapping, generally either the **Esc** key or the **Pause** key on your keyboard will clear items.

- Type:

/EXIT

This displays a screen with a certain message.

- Enter /sign off. This signs you off.

- Click the **Clear** button. The IVP is ready to start the transaction again.

NOTE: The transaction is the /FOR IVPCC command , and the ADD, DISPLAY, UPDATE, and DELETE process codes belong to this transaction.

All transaction paths follow the same sequence for a full cycle, except that on the second panel a different process code (ADD, DISPLAY, UPDATE, or DELETE) is typed into the **PROCESS CODE** field.

**Note:** Depending on the process code specified, certain entry fields can remain empty. For example, with the DELETE and DISPLAY process codes, only the **LAST NAME** field needs to be specified.

## PhoneBookEntry Object Model

Based on the IMS IVP, the following Component Broker business object interface is defined.

```

Interface PhoneBookEntry {
    attribute string lastName;
    attribute string firstName;
    attribute string extNumber;
    attribute string internalZip;
    void showAll (out string lname, out string fname,
                 out string ext, out string zip);
}

```

The attributes in the interface are:

**lastName**

This is the last name of the person to whom the phone book entry belongs. It is the key attribute used to uniquely identify a phone book entry instance.

**firstName**

The first name of the person.

**extNumber**

The phone number extension for the person.

**internalZip**

The mail stop or internal address for the person. This code is used by the company for internal mail delivery

The method in this interface is:

**showAll**

This method reads all the attributes and passes them out in its parameters.

**Note:** One of the critical concepts in the Component Broker Programming Model is object identity or key. A key uniquely identifies an instance of a class.

---

## Enterprise Access Builder Procedures

An overview of the steps is given below:

1. "Creating a Project/Package under VisualAge for Java" on page 59
2. "Creating the Procedural Adaptor Object and Key" on page 59
3. "Importing the tele.mfs File" on page 62
4. "Creating the Record Mapper" on page 62
5. "Creating the SingleLine Record Type and Record Bean" on page 63
6. "Creating the Command Beans" on page 67
7. "Creating Navigator Beans" on page 77

**WIN** If you are using VisualAge for Java on Windows 95 or Windows NT, from the **Start** menu, select **Programs** → **IBM VisualAge for Java for Windows** → **IBM VisualAge for Java**.

**AIX** If you are using Visual Age for Java on AIX, type `vajide` on the command line.

If the VisualAge Quick Start dialog appears, select **Go to the Workbench** and click **OK**. The IDE appears.

From the Window pulldown, select **Options**. Select **Design Time** and un-check **Inherit BeanInfo** of bean superclass. Click **OK**.

## Importing Pre-requisite Features into the Workspace

1. Select **File** → **Quick Start**.
2. Select **Features** in the left pane and **Add Feature** in the right pane.
3. Click **OK**.
4. Select the following:

- IBM Procedural Application Adapter 1.0
- CICS Connector 3.0
- IBM Component Broker Host On Demand 1.0
- IBM Component Broker Connectors 1.0
- IBM Enterprise Access Builder Library 2.0
- IBM Component Broker PAA Samples for IMS 1.0

Click **OK**.

You may ignore the following expected errors this introduces in the following packages:

- com.ibm.ivj.communications
- com.ibm.ivj.trace
- com.ibm.eNetwork.ECL
- com.ibm.eNetwork.ncod.services.RAS

**Note:** If you do not see all of these features listed, they have been previously installed. To confirm, perform the following steps:

- a. Select **File** → **Quick Start**.
- b. Select **Features - Delete Feature** and see which features are already loaded (then Cancel).

## Creating a Project/Package under VisualAge for Java

1. From the list of projects, select **IBM Component Broker PAA Samples for IMS**.
2. Open the pop-up menu of **IBM Component Broker PAA Samples for IMS**, and select **Add Package**. This creates a package for the project.
3. Type `paa.mysamples.ims.hod.pbe` for the new package, and click the **Finish** button.

**Note:** To open the pop-up menu, right-click on the denoted item if you are using the default mouse configuration. You do not have to select the item (left-click to highlight) before opening its pop-up menu (right-click to open). You can select the item and open its pop-up menu using a single right-click.

## Creating the Procedural Adaptor Object and Key

The procedural adaptor object inherits from `com.ibm.ivj.eab.paa.EntityProceduralAdapterObject`, which serves as a base implementation for all procedural adaptor objects. As a subclass of `EntityProceduralAdapterObject`, the procedural adaptor object contains the CRUD methods (create (or insert), retrieve, update, and delete). However, these methods are all empty-bodied. You must define their implementation for your procedural adaptor object.

The attributes defined in the `PhoneBookEntry` interface are essential. Thus, the procedural adaptor object, as the adaptor that connects the Component Broker data object to the backend system, should contain the properties that correspond to these attributes.

1. From the VisualAge for Java desktop under the `CBSamples` project, select **paa.sample.ims.appc.pbe**.
2. Open the pop-up menu for **paa.sample.ims.appc.pbe**, and select **Add** → **Class**.
3. In this dialog:
  - a. Type `PhoneBookPA0` in the **Class name** field.
  - b. Click the **Browse** button to select the Superclass.
    - 1) Browse for and select **EntityProceduralAdapterObject** as your Superclass.

- 2) Click the **OK** button to close the dialog.
4. Click the **Finish** button.

Add the properties for the PhoneBookPAO interface.

1. Select the PhoneBookPAO interface.
2. Open the pop-up menu for **PhoneBookPAO**, and select **Open**, which opens the Object Editor notebook.
3. In this notebook:
  - a. Select the **BeanInfo** tab.
  - b. From the menu bar, select **Features** → **New Property Feature**, which opens the → New Property Feature.
  - c. In this wizard, type the name of the new property in the **Property name** field. For simplicity, use the same name as used in the PhoneBookRec interface.

For example, use:

```
lastName  
firstName  
extNumber  
internalZip
```

for the properties as defined in the pbe.cpp file. Each of these properties must be defined individually. For this step (first time) type `lastName`. For each subsequent time, type `firstName`, `extNumber`, `internalZip`, respectively.

- 1) For all properties, select **java.lang.String** from the pull down menu of the **Property type** field.
- 2) Accept the other defaults and click the **Next** button.
- 3) Click the **Finish** button.
- d. Close the Object Editor window.

Now the Key for this PAO object needs to be created.

1. From the VisualAge for Java desktop under the CBSamples project, select **paa.sample.ims.appc.pbe**.
2. Open the pop-up menu for **paa.sample.ims.appc.pbe**, and select **Add Class**.
3. In this dialog:
  - a. Type `PhoneBookPAOKey` in the **Class name** field.
  - b. Click the **Browse** button to select the Superclass.
    - 1) Browse for and select **BusinessObjectKey** as your Superclass.
    - 2) Click the **OK** button to close the dialog.
  - c. Click the **Finish** button.

Add the properties for the PhoneBookPAOKey interface.

1. Select the **PhoneBookPAOKey** interface.
2. Open the pop-up menu for **PhoneBookPAOKey**, and select **Open**, which opens the Object Editor notebook.
3. In this notebook:
  - a. Select the **BeanInfo** tab.

- b. From the menu bar, select **Features** → **New Property Feature**, which opens the wizard - New Property Feature.
- c. In this wizard:
  - 1) Type the name of the new property in the **Property name** field. For example, use *lastName* for the property that is going to be the key attribute. You can select **java.lang.String** for the type of the property.
  - 2) Accept the other defaults and click the **Next** button.
  - 3) Click the **Finish** button.
- d. Close the Object Editor window.

Modify the PhoneBookPAOKey and PhoneBookPAO to tie the PAO and key class together

### PhoneBookPAOKey

1. Select and expand the **PhoneBookPAOKey** class.
2. Highlight the `getPropertyValues()` method. This method is used by the Enterprise Access Builder (EAB) run time to calculate a value to key into the CICON cache. It needs to be modified to specifically return just the key values.
3. In the source pane, return an array of Objects that make up the key by invoking the methods that get the key properties. For example:
 

```
return new Object[] { this.getLastName() };
```
4. Save the changes to the modified PAO Key class by pressing **Ctrl-S**.
5. Highlight the `setLastName(String)` method. This method is used to set the last name. It must be modified to trim the `oldValue` and `lastName` fields.
6. In the source pane, add two lines, `oldValue.trim()` and `lastName.trim()`. Your code should look like the following:
 

```
{
    String oldValue = fieldLastName;
    lastName.trim();
    fieldLastName = lastName;
    fireProertyChange ("lastName", oldValue, lastName);
}
```
7. Save these changes to the modified PAO Key class.

### PhoneBookPAO

1. Select and expand the PhoneBookPAO class.
2. Modify the getter for the key property value `getLastName()` by getting the key class associated with this PAO and returning that value. For example:
 

```
PhoneBookPAOKey key = (PhoneBookPAOKey) this.getKey();
return key.getLastName();
```
3. Save the changes to the modified PAO by pressing **Ctrl-S**.

## Importing the tele.mfs File

1. Select the package that you have created
2. Open the pop-up menu for the package you are working under, and select **Tools - Records - Create MFS Record Type...**, a wizard window will appear.
3. In this wizard window:
  - a. In the **Class Name** field, type PhoneBookInfo.
  - b. In the **MFS File** field, browse through the files to locate the tele.mfs file. It should be located in one of the following:
    - WIN** (CBroker)\samples\InstallVerification\PAA\Backend\PhoneBook\ and select **Open**.
    - AIX** \$HOME/samples/InstallVerification/PAA/Backend/PhoneBook/ and select **Open**.
  - c. In the Format Name type IVTCCF.
  - d. In the Device Page Name type IVTCCF.
  - e. In the Device Type select 3270,2.
  - f. Check that the Project and Package names are correct.
  - g. Select **Finish**.
4. Select the **PhoneBookInfo** class
5. Open the pop-up menu for the **PhoneBookInfo** class, and select **Tools - Records - Generate Records...**, the Generate Records wizard will appear.
6. In this wizard window:
  - a. In the **Class Name** field, type PhoneBookRecord.
  - b. Select the **Beans** radio button to generate the records as beans.
  - c. Select the **Direct** radio button to access the record fields directly.
  - d. Select the **Dynamic Records** radio button to generate the records as dynamic records.
  - e. Click **Finish** when this is complete. Three new classes will appear in your package:
    - PhoneBookRecord
    - PhoneBookRecordBeanInfo
    - PhoneBookRecordType

## Creating the Record Mapper

1. Select the sample package again that you have created and expand it.
2. Select class **PhoneBookRecord**.
3. Open the pop-up menu for the PhoneBookRecord class, and select **Tools - Mapper Editor...**, a Mapper wizard will appear.
4. In this window:
  - a. From the Code Generation pulldown, select **Set Target mapper**. A window containing three fields will appear.
  - b. In this window:
    - 1) Enter the project and package name of this sample in the first two fields.

- 2) In the **Class** field, enter PhoneBookRecordMapper.
- 3) Click **OK** when this is complete.
- c. Select **Change Input bean** from the Code Generation pulldown. A window containing one field will appear.
- d. In this window:
  - 1) Select **Browse**. Next type PhoneBookRecord to select the class (corresponding to the sample package) and click **OK**.
  - 2) Click **OK** once more to select the Input Bean class. The following message will appear:  
All of your connections will be lost. Do you wish to proceed?  
Click **Yes**.
  - 3) You will now see a list of fields available from PhoneBookRecord.
- e. Click the **Add** button located at the bottom left of the wizard window and type PhoneBookPAO in the **Pattern** field.
- f. Select the **PhoneBookPAO** class corresponding to the package you are currently using.
- g. Click **OK** when this is complete. You will now see a directory named java.lang.Object in the Output Beans side of the window.
- h. Expand this directory until the first instance of PhoneBookPAO is visible (should be able to see extNumber, firstName, lastname, etc.)
- i. Select the **extNumber** field of the PhoneBookPAO object. Move the cursor to the right hand side of the screen and select **EXT**. At the bottom of the screen, click the **<—> input/output** button to connect the two fields. Note that the PhoneBookPAO **extNumber** field should be connected to **EXT** on the input side. Repeat this step to form connections between the rest of the **PhoneBookPAO** fields (firstName **<—>** NAME2, lastName **<—>** NAME1, internalZip **<—>** ZIP). Select **Apply** and **OK** when this is complete. A new class called PhoneBookRecordMapper will appear in the current package.

## Creating the SingleLine Record Type and Record Bean

The IMS application base state is just a blank screen. The SingleLineRecord class is needed to interact with the blank screen.

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class....**
2. In this dialog:
  - a. Type the Class Name: SingleLineRecordType
  - b. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.record.terminal.FixedLengthTerminalRecordType as described in the Select class instructions.
  - c. Click the **Finish** button. This creates the SingleLineRecordType class.
3. Right-click on the SingleLineRecordType class, and select **Tools** → **Records** → **Edit Record Type....** This opens the Java Record Editor.
4. Right-click on the SingleLineRecordType record, and select Create New Field As Child.... This opens the Create a Field wizard.
  - a. Select **Simple** and click the **Next** → button.
  - b. Type the Field Name: Value\_attByte

- c. Select Field Type: com.ibm.ivj.eab.record.terminal.FixedLengthTerminalAttributeType
  - d. Click the **Finish** button. This creates the **Value\_attByte** field.
5. Change the Read Only property of the **Value\_attByte** field to True by clicking on its value.
  6. Right-click on the **Value\_attByte** field, and select Create New Field As Sibling.... This opens the Create a Field wizard.
    - a. Select Simple and click the **Next >** button.
    - b. Type the Field Name: Value
    - c. Select Field Type: com.ibm.ivj.eab.record.terminal.FixedLengthTerminalFieldType
    - d. Click the **Finish** button. This creates the field.

7. Verify that the fields look like the following:

```
SingleLineRecordType
  Value_attByte
  Value
```

8. Click on the **Value** field. Update the type size to 79 by clicking on the 1 to change it to 79, and click **Enter**.
9. Click the **Done** button. This closes the Java Record Editor.

**Note:** If a dialog pops up asking to save your changes, click **yes**.

10. Right-click on **SingleLineRecordType**, and select **Tools** → **Records** → **Generate Records....**
11. In the **Class Name:** field, type SingleLineRecord.
12. Leave the other fields as default and click the **Finish** button. This creates the classes SingleLineRecord and SingleLineRecordBeanInfo.

The IMS application first signon state is a screen with user ID, password, and application name fields. The FirstSignonScreenRecordType class is needed to interact with this screen.

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class....**
2. In this dialog:
  - a. Type the Class Name: FirstSignonScreenRecordType
  - b. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.record.terminal.FixedLengthTerminalRecordType as described in the Select class instructions.
  - c. Click the **Finish** button. This creates the FirstSignonScreenRecordType class.
3. Right-click on the FirstSignonScreenRecordType class, and select **Tools** → **Records** → **Edit Record Type....** This opens the Java Record Editor.
4. Right-click on the FirstSignonScreenRecordType record and select **Create New Field As Child....** This opens the Create a Field wizard.
  - a. Select Padding and click the **Next >** button.
  - b. Type 10 for the Padding: value.
  - c. Click the **Finish** button. This creates the **\_IVJ\_PADFIELD\_0\_** field.
5. Click the **Done** button. This closes the Java Record Editor.
 

**Note:** If a dialog pops up asking you to save your changes, click **yes**.
6. Right-click on the plus to open the FirstSignonScreenRecordType class to edit the constructor method.

7. Replace the try code with the following code:

```
try {
    int[] arraySize = null;

    addField(new Field(new FixedLengthTerminalAttributeType((byte)4), "_0001_attByte", true));
    addField(new Field(new FixedLengthTerminalFieldType(18, false, 0, (byte)0), "_0001", null,
        "Enter Your Userid:", true));

    addField(new Field(new FixedLengthTerminalAttributeType((byte)-112),
        "USERID_attByte", true));
    addField(new Field(new FixedLengthTerminalFieldType(10, false, 1, (byte)32), "USERID", null,
        null, false));

    addPadding(50);

    addField(new Field(new FixedLengthTerminalAttributeType((byte)4), "_0002_attByte", true));
    addField(new Field(new FixedLengthTerminalFieldType(9, false, 0, (byte)0), "_0002", null,
        "Password:", true));

    addField(new Field(new FixedLengthTerminalAttributeType((byte)-112),
        "PASSWORD_attByte", true));
    addField(new Field(new FixedLengthTerminalFieldType(10, false, 1, (byte)32), "PASSWORD",
        null, null, false));

    addPadding(58);

    addField(new Field(new FixedLengthTerminalAttributeType((byte)4), "_0003_attByte", true));
    addField(new Field(new FixedLengthTerminalFieldType(13, false, 0, (byte)0), "_0003", null,
        "Application: ", true));

    addField(new Field(new FixedLengthTerminalAttributeType((byte)-112),
        "APPLICATION_attByte", true));
    addField(new Field(new FixedLengthTerminalFieldType(10, false, 1, (byte)32), "APPLICATION",
        null, null, false));
}
catch (Exception e) {
    throw new RecordException(e.getMessage());
}
```

8. Save the changes to the modified FirstSignonScreenRecordType class by pressing **Ctrl-S**.

9. Right-click on FirstSignonScreenRecordType, and select **Tools** → **Records** → **Generate Records...**

10. In the **Class Name:** field, type FirstSignonScreenRecord.

11. Leave the other fields as default and click the **Finish** button. This creates the classes FirstSignonScreenRecord and FirstSignonScreenRecordBeanInfo.

The IMS application second signon state is a screen with **user ID** and **password** fields. The SecondSignonScreenRecordType class is needed to interact with this screen.

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class...**

2. In this dialog:

a. Type the Class Name: SecondSignonScreenRecordType

b. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.record.terminal.FixedLengthTerminalRecordType as described in the Select class instructions.

- c. Click the **Finish** button. This creates the `SecondSignonScreenRecordType` class.
3. Right-click on the `SecondSignonScreenRecordType` class, and select **Tools** → **Records** → **Edit Record Type...** This opens the Java Record Editor.
4. Right-click on the `SecondSignonScreenRecordType` record, and select **Create New Field As Child...** This opens the Create a Field wizard.
  - a. Select **Padding** and click the **Next** → button.
  - b. Type 10 for the Padding: value
  - c. Click the **Finish** button. This creates the `_IVJ_PADFIELD_0_` field.
5. Click the **Done** button. This closes the Java Record Editor.
 

**Note:** If a dialog pops up asking you to save your changes, click yes.
6. Right-click on the **+** to open the `SecondSignonScreenRecordType` class to edit the constructor method.
7. Replace the try code with the following code:

```
try {

    int[] arraySize = null;

    addPadding(480);

    FixedLengthTerminalAttributeType _0001_attByteType = new FixedLengthTerminalAttributeType();
    _0001_attByteType.setExpectedValue((byte)4);
    addField(new Field(_0001_attByteType, "_0001_attByte",true));

    FixedLengthTerminalFieldType _0001Type = new FixedLengthTerminalFieldType();
    _0001Type.setPaddingPolicy(0);
    _0001Type.setSize(7);
    _0001Type.setPaddingByte((byte)0);
    addField(new Field(_0001Type, "_0001",null,new java.lang.String("USERID:"),true));

    FixedLengthTerminalAttributeType USERID_attByteType = new FixedLengthTerminalAttributeType();
    USERID_attByteType.setExpectedValue((byte)-112);
    addField(new Field(USERID_attByteType, "USERID_attByte",true));

    FixedLengthTerminalFieldType USERIDType = new FixedLengthTerminalFieldType();
    USERIDType.setSize(8);
    addField(new Field(USERIDType, "USERID"));

    addPadding(143);

    FixedLengthTerminalAttributeType _0002_attByteType = new FixedLengthTerminalAttributeType();
    _0002_attByteType.setExpectedValue((byte)4);
    addField(new Field(_0002_attByteType, "_0002_attByte",true));

    FixedLengthTerminalFieldType _0002Type = new FixedLengthTerminalFieldType();
    _0002Type.setPaddingPolicy(0);
    _0002Type.setSize(9);
    _0002Type.setPaddingByte((byte)0);
    addField(new Field(_0002Type, "_0002",null,new java.lang.String("PASSWORD:"),true));

    FixedLengthTerminalAttributeType PASSWORD_attByteType = new FixedLengthTerminalAttributeType();
    PASSWORD_attByteType.setExpectedValue((byte)-112);
    addField(new Field(PASSWORD_attByteType, "PASSWORD_attByte",true));
```

```

FixedLengthTerminalFieldType PASSWORDType = new FixedLengthTerminalFieldType();
PASSWORDType.setSize(8);
addField(new Field(PASSWORDType, "PASSWORD"));

addPadding(141);

FixedLengthTerminalAttributeType _0003_attByteType = new FixedLengthTerminalAttributeType();
_0003_attByteType.setExpectedValue((byte)4);
addField(new Field(_0003_attByteType, "_0003_attByte",true));

FixedLengthTerminalFieldType _0003Type = new FixedLengthTerminalFieldType();
_0003Type.setPaddingPolicy(0);
_0003Type.setSize(16);
_0003Type.setPaddingByte((byte)0);
addField(new Field(_0003Type, "_0003",null,new java.lang.String("USER DESCRIPTOR:"),true));

FixedLengthTerminalAttributeType DESCRIPTOR_attByteType =
    new FixedLengthTerminalAttributeType();
DESCRIPTOR_attByteType.setExpectedValue((byte)-112);
addField(new Field(DESCRIPTOR_attByteType, "DESCRIPTOR_attByte",true));

FixedLengthTerminalFieldType DESCRIPTORType = new FixedLengthTerminalFieldType();
DESCRIPTORType.setSize(8);
addField(new Field(DESCRIPTORType, "DESCRIPTOR"));

}
catch (Exception e) {
    throw new RecordException(e.getMessage());
}

```

8. Save the changes to the modified SecondSignonScreenRecordType class by pressing **Ctrl-S**.
9. Right-click on SecondSignonScreenRecordType, and select **Tools** → **Records** → **Generate Records....**
10. In the **Class Name:** field, type SecondSignonScreenRecord.
11. Leave the other fields as default and click the **Finish** button. This creates the classes SecondSignonScreenRecord and SecondSignonScreenRecordBeanInfo.

## Creating the Command Beans

In this section, you will create six Commands used to interact with the IMS application. Commands dictate the data that gets passed to and from the backend system in a single interaction. Commands use Record beans to define the layout of the input and output data. After you create the commands, you will create Navigators that encapsulate sequences of Commands to perform functions for the PAO. The following table summarizes all the Navigators and Commands used by each of the PAO methods.

<i>Table 1. Navigators and Commands used by PAO methods (IMS-HOD)</i>		
<b>PAO Method Name</b>	<b>Navigator Used</b>	<b>Commands Used</b>
retrieve	NavigatorRetrieve	CmdBaseToMenu CmdMenuToMenuDisplayCmdMenuToClear CmdClearToBase
insert	NavigatorAddUpdate	CmdBaseToMenu CmdMenuToMenuAddUpdt CmdMenuToClear CmdClearToBase
update		
del	NavigatorDel	CmdBaseToMenu CmdMenuToMenuDel CmdMenuToClear CmdClearToBase

## Creating the CmdBaseToMenu Command

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class....**
2. Type the Class Name: CmdBaseToMenu
3. Click the **Browse** button to set the Superclass to CommunicationCommand in package com.ibm.ivj.eab.command as described in the Select class instructions.
4. Click the **Finish** button. This creates the CmdBaseToMenu class.
5. Right-click on CmdBaseToMenu, and select **Tools** → **Command Editor....** This opens the Command Editor window which looks like the following:

```
Tasks
  Communication
  Input
  Output
```

6. Right-click on Communication, select Add InteractionSpec and select the class com.ibm.connector.hod.HODInteractionSpec as described in the Select class instructions. This creates a bean called ceInteractionSpec.
7. Right-click on ceInteractionSpec, and select Properties. This opens the Properties window.
  - a. Click on the **name property** and type #ENTER
  - b. Click the **OK** button to close the properties window.
8. Right-click on Input, select Add IByteBuffer Bean, and select the class paa.mysamples.cics.menu.SingleLineRecord as described in the Select class instructions. This creates an input record bean called ceInput.
9. Right-click on ceInput, and select Properties. This opens the Properties window.
  - a. Scroll down to the Value property, click on it, and type /FOR IVTCC
  - b. Click the **OK** button to close the properties window.
10. Right-click on Output, select Add IByteBuffer Bean, and select the class paa.mysamples.ims.hod.pbe.PhoneBookRecord as described in the Select class instructions. This creates an output record bean called ceOutput1.
11. Click the **OK** button to close the Command Editor.

## Command Bean Summary

The following table summarizes the properties of all the Command Beans used in this sample. You created the CmdBaseToMenu in the previous section.

<i>Table 2. Command Bean summary (IMS-HOD)</i>		
<b>Command Name</b>	<b>Properties</b>	<b>Values</b>
CmdBaseToMenu	ceInteractionSpec	Class: HODInteractionSpec Name: #ENTER
	ceInput	Class: SingleLineRecord Value: /FOR IVTCC
	ceOutput1	Class: PhoneBookRecord
CmdMenuToClear	ceInteractionSpec	Class: HODInteractionSpec Name: #CLEAR
	ceInput	Class: PhoneBookRecord
	ceOutput1	Class: SingleLineRecord
CmdClearToBase	ceInteractionSpec	Class: HODInteractionSpec Name: #ENTER
	ceInput	Class: SingleLineRecord Value: /EXIT
	ceOutput1	Class: SingleLineRecord
CmdMenuToMenuDisplay	ceInteractionSpec	Class: HODInteractionSpec Name: #ENTER
	ceInput	Class: PhoneBookRecord Property Features: CMD,NAME1
	ceOutput1	Class: PhoneBookRecord Add Mapper: PhoneBookRecordMapper Property Features: MSG
CmdMenuToMenuAddUpdt	ceInteractionSpec	Class: HODInteractionSpec Name: #ENTER
	ceInput	Class: PhoneBookRecord Add Mapper: PhoneBookRecordMapper
	ceOutput1	Class: PhoneBookRecord Property Feature: MSG
CmdMenuToMenuDel	ceInteractionSpec	Class: HODInteractionSpec Name: #ENTER
	ceInput	Class: PhoneBookRecord Property Features: CMD, NAME1
	ceOutput1	Class: PhoneBookRecord Property Feature: MSG
CmdFirstToSecondSignon	ceInteractionSpec	Class: HODInteractionSpec Name: #ENTER
	ceInput	Class: FirstSignonScreenRecord
	ceOutput1	Class: SecondSignonScreenRecord
CmdSecondSignonToBase	ceInteractionSpec	Class: HODInteractionSpec Name: #ENTER
	ceInput	Class: SecondSignonScreenRecord
	ceOutput1	Class: SingleLineRecord

## Creating the CmdMenuToClear Command

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class....**
2. Type the Class Name: CmdMenuToClear
3. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.command.CommunicationCommand as described in the Select class instructions. This creates the CmdMenuToClear class.
4. Right-click on CmdMenuToClear, and select **Tools** → **Command Editor....** This opens the Command Editor.
5. Right-click on Communication, select Add InteractionSpec and select the class com.ibm.connector.hod.HODInteractionSpec as described in the Select class instructions. This creates a bean called ceInteractionSpec.
6. Right-click on ceInteractionSpec, and select Properties. This opens the Properties window.
  - a. Click on the **name property** and type #CLEAR
  - b. Click the **OK** button to close the properties window.
7. Right-click on Input, select Add IByteBuffer Bean, and select the class paa.mysamples.ims.hod.PhoneBookRecord as described in the Select class instructions. This creates an input record bean called ceInput.
8. Right-click on Output, select Add IByteBuffer Bean, and select the class paa.mysamples.ims.hod.SingleLineRecord as described in the Select class instructions. This creates an output record bean called ceOutput1.
9. Click the **OK** button to close the Command Editor.

## Creating the CmdClearToBase Command

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class....**
2. Type the Class Name: CmdClearToBase
3. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.command.CommunicationCommand as described in the Select class instructions.
4. Click the **Finish** button. This creates the CmdClearToBase class.
5. Right-click on CmdClearToBase, and select Tools → Command Editor.... This opens the Command Editor window.
6. Right-click on Communication, select Add InteractionSpec and select the class com.ibm.connector.hod.HODInteractionSpec as described in the Select class instructions. This creates a bean called ceInteractionSpec.
7. Right-click on ceInteractionSpec, and select Properties. This opens the Properties window.
  - a. Click on the **name property** and type #ENTER
  - b. Click the **OK** button to close the properties window.
8. Right-click on Input, select Add IByteBuffer Bean, and select the class paa.mysamples.cics.menu.SingleLineRecord as described in the Select class instructions. This creates an input record bean called ceInput
9. Right-click on ceInput, and select Properties. This opens the Properties window.
  - a. Scroll down to the Value property, click on it, and type /EXIT
  - b. Click the **OK** button to close the properties window.

10. Right-click on Output, select Add IByteBuffer Bean, and select the class `paa.mysamples.ims.hod.pbe.SingleLineRecord` as described in the Select class instructions. This creates an output record bean called `ceOutput1`.
11. Click the **OK** button to close the Command Editor.

## Creating the `CmdFirstToSecondSignon` Command

1. Right-click on `paa.mysamples.ims.hod.pbe`, and select **Add** → **Class....**
2. Type the Class Name: `CmdFirstToSecondDisplay`
3. Click the **Browse** button to set the Superclass to `com.ibm.ivj.eab.command.CommunicationCommand` as described in the Select class instructions.
4. Click the **Finish** button to create the `CmdFirstToSecondSignon` class.
5. From the menu for the class, select **Open To** → **BeanInfo**.
6. In this dialog:
  - a. Select Features → Generate BeanInfo.
  - b. Select Features → Add Available Features.
  - c. Select items class through output, both `execute()` items, and both `executionSuccessful` items.
7. Right-click on Communication, select Add InteractionSpec and select the class `com.ibm.connector.hod.HODInteractionSpec` as described in the Select class instructions. This creates a bean called `ceInteractionSpec`.
8. Right-click on `ceInteractionSpec`, and select Properties. This opens the Properties window.
  - a. Click on the **name property** and type `#ENTER`
  - b. Click the **OK** button to close the properties window.
  - c. Right-click on Input, select Add IByteBuffer Bean, and select the class `paa.mysamples.ims.hod.pbe.FirstSignonScreenRecord` as described in the Select class instructions. This creates an input record bean called `ceInput`.
  - d. Right click on `ceInput` and select Promote Bean Feature.
  - e. Click on the **Property** radio button and select APPLICATION.
  - f. Click on the `>>` button.
  - g. Repeat for USERID and PASSWORD.
  - h. Click the **OK** button.
  - i. Right-click on Output, select Add IByteBuffer Bean, and select the class `paa.mysamples.ims.hod.pbe.SecondSignonScreenRecord` as described in the Select class instructions. This creates an input record bean called `ceOutput1`.
  - j. Click the **OK** button to close the Command Editor.

## Creating the `CmdSecondSignonToBase` Command

1. Right-click on `paa.mysamples.ims.hod.pbe`, and select **Add** → **Class....**
2. Type the Class Name: `CmdSecondSignonToBase`
3. Click the **Browse** button to set the Superclass to `com.ibm.ivj.eab.command.CommunicationCommand` as described in the Select class instructions.
4. Click the **Finish** button to create the `SecondSignonToBase` class.

5. From the menu for the class, select **Open To** → **BeanInfo**.
6. In this dialog:
  - a. Select Features → Generate BeanInfo.
  - b. Select Features → Add Available Features.
  - c. Select items class through output, both execute() items, and both executionSuccessful items.
7. Right-click on CmdSecondSignonToBase, and select Tools → Command Editor.... This opens the Command Editor window.
8. Right-click on Communication, select Add InteractionSpec and select the class com.ibm.connector.hod.hodInteractionSpec as described in the Select class instructions. This creates a bean called ceInteractionSpec.
  - a. Click on the **name property** and type #ENTER
  - b. Click the **OK** button to close the properties window.
  - c. Right-click on Input, select Add IByteBuffer Bean, and select the class paa.mysamples.ims.hod.pbe.SecondSignonScreenRecord as described in the Select class instructions. This creates an input record bean called ceInput.
  - d. Right click on ceInput and select Promote Bean Feature.
  - e. Click on the **Property** radio button and select PASSWORD.
  - f. Repeat for USERID.
  - g. Click the **OK** button.
  - h. Right-click on Output, select Add IByteBuffer Bean, and select the class paa.mysamples.ims.hod.pbe.SingleLineRecord as described in the Select class instructions. This creates an input record bean called ceOutput1.
  - i. Click the **OK** button to close the Command Editor.

## Creating the CmdMenuToMenuDisplay Command

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class....**
2. Type the Class Name: CmdMenuToMenuDisplay
3. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.command.CommunicationCommand as described in the Select class instructions. This creates the CmdMenuToMenuDisplay class.
4. From the menu for the class, select Open To → BeanInfo.
5. In this dialog:
  - a. Select Features → Generate BeanInfo.
  - b. Select Features → Add Available Features.
  - c. Select items class through output, both execute() items, and both executionSuccessful items.
6. Right-click on CmdMenuToMenuDisplay, and select Tools → Command Editor....This opens the Command Editor.
7. Right-click on Communication, select Add InteractionSpec and select the class com.ibm.connector.hod.HODInteractionSpec as described in the Select class instructions. This creates a bean called ceInteractionSpec.
8. Right-click on ceInteractionSpec, and select Properties. This opens the Properties window.
  - a. Click on the **name property** and type #ENTER

- b. Click the **OK** button to close the properties window.
9. Right-click on Input, select Add IByteBuffer Bean Variable, and select the class `paa.mysamples.ims.hod.pbe.PhoneBookRecord` as described in the Select class instructions. This creates an input record bean called `[celInput]`.
10. Right-click on Output, select Add IByteBuffer Bean, and select the class `paa.mysamples.ims.hod.pbe.PhoneBookRecord` as described in the Select class instructions. This creates an input record bean called `ceOutput1`.
11. Right-click on `ceOutput1`, select Add Mapper, and select the class `paa.mysamples.ims.hod.pbe.PhoneBookRecordMapper` as described in the Select class instructions. This creates a mapper bean called `ceMapperCeOutput1`.
12. Right-click on `ceOutput1`, select Promote Bean Feature. This opens the Promoted features dialog.
  - a. Click the **Property** radio button, and select `MSGRWB`.
  - b. Click the `>>` button, and then click the **OK** button. This closes the Promoted features dialog.
  - c. Repeat the previous two steps for `EXTRWB`, `NAME2RWB`, and `ZIPRWB` from the list of properties.
  - d. Repeat the previous two steps for `EXTRWB`, `NAME2RWB`, and `ZIPRWB` from the list of properties.
  - e. Click the **OK** button to close the command editor.
13. Select the `CmdMenuToMenuDisplay` class.
14. In this window, click the **BeanInfo** tab. This will display all property features defined form `CmdMenuToMenuDisplay`. There is at least one already: `ceOutput1MSG`, which was promoted there with Command Editor. Now add two more property features, `celInputCMD` and `celInputNAME1`. At each addition, only enter the feature name and leave everything else at default.
 

**Note:** `CMD` corresponds to the field in the `MENU` panel where requests such as “DISPLAY,” “ADD,” “DELETE,” etcetera are entered.
15. Click the **Visual Composition** tab.
16. Right-mouse click at the `celInput` icon and select `Connect → this`, then move the cursor back to `celInput` icon again and click. In the popup menu, select `Connectable Features`.
17. In the dialog that comes up, click the **Method** radio button first. Then in the list window, select `CMD` and click the **OK** button.
18. A green, dotted, arrowed connection line should appear which goes off `celInput` and then loops back to it. If the line does not look so, delete it and redo the last two steps.
 

**Note:** As the spot around `celInput` gets crowded, make sure when you “click at X” it is indeed X not Y that gets clicked at. This can be achieved by paying attention to the highlighting on the artifacts involved.
19. Right-mouse click at the dotted green loop and select `Connect → value`, then move the cursor to any background area and click on it. In the popup menu select `Connectable Features`.
20. In the dialog that comes up, click the **Property** radio button first. Then in the list window, select `celInputCMD` and click **OK**.
21. The dotted green loop should turn solid, and a violet-colored line should connect the loop to the edge of the window.
22. This finishes the promotion of the property feature `CMD`. Repeat the last six steps to promote `NAME1` also.

23. In the window select the menu Bean → Save Bean. This completes the creation of CmdMenuToMenuDisplay.
24. Close the window.

## Creating the CmdMenuToMenuAddUpdt Command

1. Right-click on paa.mysamples.ims.hod.pbe, and select Add → Class...
2. Type the Class Name: CmdMenuToMenuAddUpdt
3. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.command.CommunicationCommand as described in the Select class instructions. This creates the CmdMenuToMenuAddUpdt class.
4. In this dialog:
  - a. Select Features → Generate BeanInfo.
  - b. Select Features → Add Available Features.
  - c. Select items class through output, both execute() items, and both executionSuccessful items.
5. Right-click on CmdMenuToMenuAddUpdt, and select Tools → Command Editor.... This opens the Command Editor.
6. Right-click on Communication, select Add InteractionSpec and select the class com.ibm.connector.hod.HODInteractionSpec as described in the Select class instructions. This creates a bean called ceInteractionSpec.
7. Right-click on ceInteractionSpec, and select Properties. This opens the Properties window.
  - a. Click on the **name property** and type #ENTER
  - b. Click the **OK** button to close the properties window.
8. Right-click on Input, select Add IByteBuffer Bean Variable, and select the class paa.mysamples.ims.hod.pbe.PhoneBookRecord as described in the Select class instructions. This creates an input record bean called ceInput.
9. Right-click on [ceInput],select Add Mapper,and select the class paa.mysamples.ims.hod.pbe.PhoneBookRecordMapper as described in the Select class instructions. This creates a mapper bean called ceMapperCeInput.
10. Right-click on Output, select Add IByteBuffer Bean, and select the class paa.mysamples.ims.hod.pbe.PhonebookRecord as described in the Select class instructions. This creates an output record bean called ceOutput1.
11. Right-click on ceOutput1, select Promote Bean Feature. This opens the Promoted features dialog.
  - a. Click the **Property** radio button, and select MSG<sup>RWB</sup> from the list of properties.
  - b. Click the >> button to move the selected properties, and click the **OK** button. This closes the Promoted features dialog.
12. Click the **OK** button to close the Command Editor.
13. Select the CmdMenuToMenuAddUpdt class.
14. In this window, click the **BeanInfo** tab. This will display all property features defined form CmdMenuToMenuAddUpdt. There is at least one already: ceOutput1MSG, which was promoted there with Command Editor. Now add two more property features, ceInputCMD and ceInputNAME1. At each addition, only enter the feature name and leave everything else at default.
 

**Note:** CMD corresponds to the field in the MENU panel where requests such as “DISPLAY,” “ADD,” “DELETE,” etcetera are entered.

15. Click the **Visual Composition** tab.
16. Right-mouse click at the celInput icon and select **Connect** → this, then move the cursor back to celInput icon again and click. In the popup menu select **Connectable Features**.
17. In the dialog that comes up, click the **Method** radio button first. Then in the list window, select CMD and click **OK**.
18. A green, dotted, arrowed connection line should appear which goes off celInput and then loops back to it. If the line does not look so, delete it and redo the last two steps. Note: As the spot around celInput gets crowded, make sure when you "click at X" it is indeed X not Y that gets clicked at. This can be achieved by paying attention to the highlighting on the artifacts involved.
19. Right-mouse click at the dotted green loop and select **Connect** → value, then move the cursor to any background area and click. In the popup menu select **Connectable Features**.
20. In the dialog that comes up, click the **Property** radio button first. Then in the list window, select celInputCMD and click **OK**.
21. The dotted green loop should turn solid, and a violet-colored line should connect the loop to the edge of the window.
22. This finishes the promotion of the property feature CMD. Repeat the last six steps to promote NAME1 also.
23. In the window, select the menu Bean → Save Bean. This completes the creation of CmdMenuToMenuDisplay.
24. Close the window.

## Creating the CmdMenuToMenuDel Command

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class....**
2. Type the Class Name: CmdMenuToMenuDel
3. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.command.CommunicationCommand as described in the Select class instructions. This creates the CmdMenuToMenuDel class.
4. In this dialog:
  - a. Select Features → Generate BeanInfo.
  - b. Select Features → Add Available Features.
  - c. Select items class through output, both execute() items, and both executionSuccessful items.
5. Right-click on CmdMenuToMenuDel, and select Tools → Command Editor.... This opens the Command Editor.
6. Right-click on Communication, select Add InteractionSpec and select the class com.ibm.connector.hod.HODInteractionSpec as described in the Select class instructions. This creates a bean called celInteractionSpec.
7. Right-click on celInteractionSpec, and select Properties. This opens the Properties window
  - a. Click on the **name property** and type #ENTER.
  - b. Click the **OK** button to close the properties window.
8. Right-click on Input, select Add IByteBuffer Bean Variable, and select the class paa.mysamples.ims.hod.pbe.PhoneBookRecord as described in the Select class instructions. This creates an input record bean called celInput.

9. Right-click on Output, select Add IByteBuffer Bean, and select the class paa.mysamples.ims.hod.pbe.PhonebookRecord as described in the Select class instructions. This creates an output record bean called ceOutput1.
10. Right-click on ceOutput1, select Promote Bean Feature. This opens the Promoted features dialog.
  - a. Click the **Property** radio button, and select MSG<sup>RWB</sup> from the list of properties.
  - b. Click the >> button to move the selected properties, and click the **OK** button. This closes the Promoted features dialog.
11. Click the **OK** button to close the Command Editor.
12. Select the CmdMenuToMenuDel class.
13. In this window, click the **BeanInfo** tab. This will display all property features defined form CmdMenuToMenuDel. There is at least one already: ceOutput1MSG, which was promoted there with Command Editor. Now add two more property features, ceInputCMD and ceInputNAME1. At each addition, only enter the feature name and leave everything else at default.
 

**Note:** CMD corresponds to the field in the MENU panel where requests such as “DISPLAY,” “ADD,” “DELETE,” etcetera are entered.
14. Click the **Visual Composition** tab.
15. Right-mouse click at the ceInput icon and select Connect → this, then move the cursor back to ceInput icon again and click. In the popup menu select Connectable Features.
16. In the dialog that comes up, click the **Method** radio button first. Then in the list window, select CMD and click the **OK** button.
17. A green, dotted, arrowed connection line should appear which goes off ceInput and then loops back to it. If the line does not look so, delete it and redo the last two steps.
 

**Note:** As the spot around ceInput gets crowded, make sure when you “click at X” it is indeed X not Y that gets clicked at. This can be achieved by paying attention to the highlighting on the artifacts involved.
18. Right-mouse click at the dotted green loop and select Connect → value, then move the cursor to any background area and click. In the popup menu select Connectable Features.
19. In the dialog that comes up, click the **Property** radio button first. Then in the list window, select ceInputCMD and click **OK**.
20. The dotted green loop should turn solid, and a violet-colored line should connect the loop to the edge of the window.
21. This finishes the promotion of the property feature CMD. Repeat the last six steps to promote NAME1 also.
22. In the window, select the menu **Bean** → **Save Bean**. This completes the creation of CmdMenuToMenuDisplay.

## Creating Logoff/Logon/Class

1. Right-click on paa.mysamples.ims.hod.pbe and select **Add** → **Class....**
2. Type the Class Name: PBELogonLogoff
3. Click the **Next** button to set the interface implement class.
4. Click the **Add** button, and then select com.ibm.ivj.eab.command.LogonLogoff.
5. Click the **Add** button.
6. Click the **Finish** button to create the PBELogonLogoff class.

## Creating Navigator Beans

In this section, you will create three Navigators used to interact with the CICS application.

### Creating the NavigatorRetrieve Navigator

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class...**
2. Type the Class Name: NavigatorRetrieve
3. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.command.CommunicationNavigator as described in the Select class instructions.
4. Click the **Finish** button.
5. From the menu for the class, select Open To → BeanInfo.
6. In this dialog:
  - a. Select Features → Generate BeanInfo.
  - b. Select Features → Add Available Features.
  - c. Select items class through output, both internalExecutionStarting() items, and returnExecutionStatus() items.
7. Click on the **Visual Composition** tab.
8. Follow the to add a new bean with Class = com.ibm.connector.hod.HODConnectionSpec and Name = connSpec.

Adding a Bean in the Visual Composition Editor
<ol style="list-style-type: none"><li>1. Click the Choose Bean... icon, (in the top-right corner of the tool palette). In this dialog:</li><li>2. Set the Bean Type to <b>Class</b>.</li><li>3. Click the <b>Browse</b> button and select the correct class. (See the Select class instructions.)</li><li>4. Type the appropriate name in the <b>Name:</b> field.</li><li>5. Click the <b>OK</b> button.</li><li>6. Drop the bean onto the Visual Composition Editor canvas by clicking somewhere on the canvas.</li></ol>

1. Click the Choose Bean... icon, (in the top-right corner of the tool palette). In this dialog:
  2. Set the Bean Type to **Class**.
  3. Click the **Browse** button and select the correct class. (See the Select class instructions.)
  4. Type the appropriate name in the **Name:** field.
  5. Click the **OK** button.
  6. Drop the bean onto the Visual Composition Editor canvas by clicking somewhere on the canvas.
9. Right-click on connSpec and select Properties. This opens the Properties Window.
    - a. Change the debugScreenEnabled property to True.
    - b. Change the hostname property to csdmec06.stl.ibm.com.
    - c. Change the portNumber property to 23.
    - d. Close the properties window.

**Note:** The connection spec here is only useful in the unit test environment with VAJ. To run from the CB environment, the connection spec is set in each of the CRUD methods as described in the following sections.
  10. Right-click on the connSpec and select Connect → this.
    - a. Click on the window background. This opens the End connection dialog.
    - b. Select connectionSpec<sup>RWB</sup> and click the **OK** button.
  11. Follow the to add new beans with the following combinations of Class and Name.

<i>Table 3. Class and Name combinations in the NavigatorRetrieve (IMS-HOD)</i>	
<b>Class</b>	<b>Name</b>
paa.mysamples.cics.menu.CmdBaseToMenu	BaseToMenu
paa.mysamples.cics.menu.CmdMenuToMenuDisplay	MenuToMenuDisplay
paa.mysamples.cics.menu.CmdMenuToClear	MenuToClear
paa.mysamples.cics.menu.CmdClearToBase	ClearToBase

12. Use the to create the following connections in the NavigatorRetrieve.

<i>Table 4. NavigatorRetrieve Navigator connections (IMS-HOD)</i>	
<b>Source/Target</b>	<b>Event</b>
Source: background Target: BaseToMenu	internalExecutionStarting(CommandEvent) execute(CommandEvent)
Source: BaseToMenu Target: MenuToDisplay	executionSuccessful(CommandEvent) execute(CommandEvent)
Source: MenuToDisplay Target: MenuToClear	executionSuccessful(CommandEvent) execute(CommandEvent)
Source: MenuToClear Target: ClearToBase	executionSuccessful(CommandEvent) execute(CommandEvent)
Source: ClearToBase Target: background	executionSuccessful(CommandEvent) returnExecutionSuccessful(CommandEvent)

13. Follow the as follows:

<b>Adding a Connection in the Visual Composition Editor:</b>
<ol style="list-style-type: none"> <li>Right-click on the Source bean (or background), and select Connect → Connectable Features...(Connect...). This opens the Start connection from dialog. When selecting on the background, select only Connect.</li> <li>Click the <b>Event</b> radio button.</li> <li>Select the Source Event from the list, and click the <b>OK</b> button.</li> <li>Click on the Target bean (or background), and select Connectable Features.... This opens the End connection to dialog. When selecting on the background, select only Connect.</li> <li>Click the <b>Event</b> radio button.</li> <li>Select the Target Event from the list, and click the OK button. This creates a connection and draws a dashed green line between the source and target.</li> <li>Right-click on the dashed green line, and select Properties. This opens the Event-to-method connection dialog.</li> <li>Select the <b>Pass event data</b> check box, and click the <b>OK</b> button. This changes the line to a solid green line.</li> </ol>

14. Right-click on the MenuToDisplay bean, and select Promote Bean Feature...

15. Click the **Property** radio button.

16. Select ceInputCMD<sup>RWB</sup> from the list, and click the >> button. Similarly, move ceInputNAME1<sup>RWB</sup> and ceOutput1MSG<sup>RWB</sup> to the Promoted features list.

17. Click the **OK** button to close the Promoted features dialog.

18. Select the pull-down menu **Bean** → **Save Bean**.

- Close the Visual Composition Editor by clicking on the **X** button in the upper-right corner of the window.

## Creating the NavigatorAddUpdate Navigator

- Right-click on paa.mysamples.cics.menu, and select **Add** → **Class....**
- Type the Class Name: NavigatorAddUpdate
- Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.command.CommunicationNavigator as described in the Select class instructions. Select the check box to Compose the class visually.
- Click the **Finish** button. This creates the NavigatorAddUpdate class and opens it in the Visual Composition editor.
- Follow the to add a new bean with Class = com.ibm.connector.hod.HODConnectionSpec and Name = connSpec.
- Right-click on connSpec and select Properties. This opens the Properties Window.
  - Change the debugScreenEnabled property to True.
  - Change the hostname property to csdmec06.stl.ibm.com.
  - Change the portNumber property to 23.
  - Close the properties window.
- Right-click on the connSpec and select **Connect** → **this**.
  - Click on the window background. This opens the End connection dialog.
  - Select connectionSpec<sup>RWB</sup> and click the **OK** button.
- Follow the to add a new bean for each of the Class and Name combinations in the following table:

*Table 5. Class and Name combinations in NavigatorAddUpdate Navigator (IMS-HOD)*

Class	Name
paa.mysamples.cics.menu.CmdBaseToMenu	BaseToMenu
paa.mysamples.cics.menu.CmdMenuToMenuAddUpdt	MenuToMenuAddUpdt
paa.mysamples.cics.menu.CmdMenuToClear	MenuToClear
paa.mysamples.cics.menu.CmdClearToBase	ClearToBase

- Use the to create the following connections in the navigator.

*Table 6. NavigatorAddUpdate Navigator connections (IMS-HOD)*

Source/Target	Event
Source: background Target: BaseToMenu	internalExecutionStarting(CommandEvent) execute(CommandEvent)
Source: BaseToMenu Target: MenuToMenuDisplay	executionSuccessful(CommandEvent) execute(CommandEvent)
Source: MenuToDisplay Target: MenuToClear	executionSuccessful(CommandEvent) execute(CommandEvent)
Source: MenuToClear Target: ClearToBase	executionSuccessful(CommandEvent) execute(CommandEvent)
Source: ClearToBase Target: background	executionSuccessful(CommandEvent) returnExecutionSuccessful(CommandEvent)

10. Right-click on the MenuToMenuAddUpdt bean, and select Promote Bean Feature...
11. Click the **Property** radio button.
12. Select ceInputCMD<sup>RWB</sup> from the list, and click the >> button. Similarly, move ceInputNAME1<sup>RWB</sup> and ceOutput1MSG<sup>RWB</sup> to the Promoted features list.
13. Click the **OK** button to close the Promoted features dialog.
14. Select the pull-down menu **Bean** → **Save Bean**.
15. Close the Visual Composition Editor by clicking on the **X** button in the upper-right corner of the window.

## Creating the NavigatorDel Navigator

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class....**
2. Type the Class Name: NavigatorDel
3. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.command.CommunicationNavigator as described in the Select class instructions. Select the check box to Compose the class visually.
4. Click the **Finish** button.
5. From the menu for the class, select **Open To** → **BeanInfo**.
6. In this dialog:
  - a. Select Features → Generate BeanInfo.
  - b. Select Features → Add Available Features.
  - c. Select items class through output, both internalExecutionStarting() items, and returnExecutionStatus() items.
7. Click on the **Visual Composition** tab.
8. Follow the to add a new bean with Class = com.ibm.connector.hod.HODConnectionSpec and Name = connSpec.
9. Right-click on connSpec and select Properties. This opens the Properties Window.
  - a. Change the debugScreenEnabled property to True.
  - b. Change the hostname property to csdmec06.stl.ibm.com.
  - c. Change the portNumber property to 23
  - d. Close the properties window.
10. Right-click on the connSpec and select Connect → this.
  - a. Click on the window background. This opens the End connection dialog.
  - b. Select connectionSpec<sup>RWB</sup> and click the **OK** button.
11. Follow the to add a new bean for each of the Class and Name combinations in the following table:

<i>Table 7. Class and Name combinations in the NavigatorDel Navigator</i>	
<b>Class</b>	<b>Name</b>
paa.mysamples.ims.hod.pbe.CmdBaseToMenu	BaseToMenu
paa.mysamples.cics.menu.CmdMenuToMenuDel	MenuToMenuDel
paa.mysamples.cics.menu.CmdMenuToClear	MenuToClear
paa.mysamples.cics.menu.CmdClearToBase	ClearToBase

12. Follow the to create the connections listed in the following table:

<i>Table 8. NavigatorDel Navigator connections</i>	
Source/Target	Event
Source: background Target: BaseToMenu	internalExecutionStarting(CommandEvent) execute(CommandEvent)
Source: BaseToMenu Target: MenuToMenuDel	executionSuccessful(CommandEvent) execute(CommandEvent)
Source: MenuToMenuDel Target: MenuToClear	executionSuccessful(CommandEvent) execute(CommandEvent)
Source: MenuToClear Target: ClearToBase	executionSuccessful(CommandEvent) execute(CommandEvent)
Source: ClearToBase Target: background	executionSuccessful(CommandEvent) returnExecutionSuccessful(CommandEvent)

13. Right-click on the MenuToMenuDel bean, and select Promote Bean Feature....
14. Click the **Property** radio button.
15. Select ceInputCMD<sup>RWB</sup> from the list, and click the >> button. Similarly, move ceInputNAME1<sup>RWB</sup>, ceOutput1MSG<sup>RWB</sup> to the Promoted features list.
16. Click the **OK** button to close the Promoted features dialog.
17. Select the pull-down menu **Bean** → **Save Bean**.
18. Close the Visual Composition Editor by clicking on the **X** button in the upper-right corner of the window

## Creating the NavigatorSignon Navigator

1. Right-click on paa.mysamples.ims.hod.pbe, and select **Add** → **Class....**
2. Type the Class Name: NavigatorSignon
3. Click the **Browse** button to set the Superclass to com.ibm.ivj.eab.command.CommunicationNavigator as described in the Select class instructions. Select the check box to Compose the class visually.
4. Click the **Finish** button. This creates the NavigatorSignon class and opens it in the Visual Composition editor.
5. Follow the to add a new bean with Class = com.ibm.connector.hod.HODConnectionSpec and Name = connSpec.
6. Right-click on connSpec and select Properties. This opens the Properties Window.
  - a. Change the debugScreenEnabled property to True.
  - b. Change the hostname property to csdmec06.stl.ibm.com.
  - c. Change the LogonLogoff property to PBELogonLogoff.
  - d. Change the portNumber property to 23
  - e. Close the properties window.
7. Right-click on the connSpec and select Connect → this.
  - a. Click on the window background to open the End connection dialog.
  - b. Select connectionSpec<sup>RWB</sup> and click the **OK** button.
8. Follow the to add a new bean for each of the Class and Name combinations in the following table:

<i>Table 9. Class and Name combinations in the NavigatorSignon Navigator</i>	
<b>Class</b>	<b>Name</b>
paa.mysamples.ims.hod.pbe.CmdFirstToSecondSignon	FirstToSecondSignon
paa.mysamples.ims.hod.pbe.CmdSecondSignonToBase	SecondSignonToBase
paa.mysamples.ims.hod.pbe.CmdClearToBase	ClearToBase

9. Follow the to create the connections listed in the following table.

<i>Table 10. NavigatorSignon Navigator connections</i>	
<b>Source/Target</b>	<b>Event</b>
Source: background Target: FirstToSecondSignon	internalExecutionStarting(CommandEvent) execute(CommandEvent)
Source: FirstToSecondSignon Target: ClearToBase	executionSuccessful(CommandEvent) execute(CommandEvent)
Source: ClearToBase Target: background	executionSuccessful(CommandEvent) execute(CommandEvent)

10. Right-click on the FirstToSecondSignon bean, and select Promote Bean Feature....
11. Click the **Property** radio button.
12. Select APPLICATION<sup>RWB</sup> from the list, and click the >> button. Similarly, move PASSWORD<sup>RWB</sup> and USERID<sup>RWB</sup>.
13. Right-click on the ClearToBase bean, and select Promote Bean Feature....
14. Click the **Property** radio button.
15. Select PASSWORD<sup>RWB</sup> from the list, and click the >> button. Similarly, move USERID<sup>RWB</sup>.
16. Select the pull-down menu **Bean** → **Save Bean**.
17. Close the Visual Composition Editor by clicking on the **X** button in the upper-right corner of the window

## Using the Navigators

To use the navigators, you need to add code to the PAO methods. The PAO methods that use the Navigators are the PAO constructor, the CRUD methods, and the push-down methods. The following instructions guide you to add code to the PhoneBookPAO PAO methods.

### Creating and Editing the PBELogonLogoff Method

1. Open the Visual Age for Java Workbench window Projects view.
2. Expand the PBELogonLogoff class by clicking the + button next to it.
3. Click on logon (communication,LogonInfoltems) method and add the following code:

```
try
{
    NavigatorsSignon navigator = new NavigatorsSignon();
    navigator.setCommunication (arg1);
    navigator.execute();
}
catch (Exception e)
{
```

```

        e.printStackTrace();
    }

```

4. Save the changes to the modified method.

## Editing the PhoneBookPAO::retrieve method

1. In the Workbench, select the PhoneBookPAO method del(). The source code for the retrieve method appears in the Source pane in the lower half of the window.
2. Change the implementation to the following:

```

/**
 * This method was created in VisualAge.
 * @exception com.ibm.ipaa.IDataKeyNotFoundException The exception description. */
public void retrieve() throws com.ibm.ipaa.IDataKeyNotFoundException {
    NavigatorRetrieve navigator = new NavigatorRetrieve();

    navigator.setConnectionSpec(this.getConnectionSpec());
    navigator.setMenuToMenuDisplayCeInputCMD("DISPLAY");
    navigator.setMenuToMenuDisplayCeInputNAME1(this.getLastName());
    navigator.execute();

    String message = navigator.getMenuToMenuDisplayCeOutput1MSG();

    if (!message.equals("ENTRY WAS DISPLAYED"))
        throw new com.ibm.ipaa.IDataKeyNotFoundException(); }

```

3. Select the pull-down menu **Edit** → **Save**.

## Editing the PhoneBookPAO::del method

1. In the Workbench, select the PhoneBookPAO method del(). The source code for the del method appears in the Source pane in the lower half of the window.
2. Change the implementation to the following:

```

/**
 * This method was created in VisualAge.
 * @exception com.ibm.ipaa.IDataKeyNotFoundException The exception description. */
public void del() throws com.ibm.ipaa.IDataKeyNotFoundException {
    NavigatorDel navigator = new NavigatorDel();

    navigator.setConnectionSpec(this.getConnectionSpec());
    navigator.setMenuToMenuDelCeInputCMD("DELETE");
    navigator.setMenuToMenuDelCeInputNAME1(this.getLastName());
    navigator.execute();

    String message = navigator.getMenuToMenuDelCeOutput1MSG();

    if (!message.equals("ENTRY WAS DELETED"))
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}

```

3. Select the pull-down menu **Edit** → **Save**.

## Editing the PhoneBookPAO::insert method

1. In the Workbench, select the PhoneBookPAO method insert(). The source code for the insert method appears in the Source pane in the lower half of the window.
2. Change the implementation to the following:

```
/**
 * This method was created in VisualAge.
 * @exception com.ibm.ipaa.IDataKeyAlreadyExistsException The exception
 * description.
 */
public void insert() throws com.ibm.ipaa.IDataKeyAlreadyExistsException {
    NavigatorAddUpdate navigator = new NavigatorAddUpdate();

    navigator.setConnectionSpec(this.getConnectionSpec());
    navigator.setMenuToMenuAddUpdtCeInputCMD("ADD");
    navigator.setMenuToMenuAddUpdtCeInputNAME1(this.getLastName());
    navigator.execute();

    String message = navigator.getMenuToMenuAddUpdtCeOutput1MSG();
    if (!message.equals("ENTRY WASADDED"))
        throw new com.ibm.ipaa.IDataKeyAlreadyExistsException();
}
```

3. Select the pull-down menu **Edit** → **Save**.

## Editing the PhoneBookPAO::update method

1. In the Workbench, select the PhoneBookPAO method insert(). The source code for the insert method appears in the Source pane in the lower half of the window.
2. Change the implementation to the following.

```
/**
 * This method was created in VisualAge.
 * @exception com.ibm.ipaa.IDataKeyAlreadyExistsException The exception description.
 */
public void update() throws com.ibm.ipaa.IDataKeyNotFoundException {
    NavigatorAddUpdate navigator = new NavigatorAddUpdate();

    navigator.setConnectionSpec(this.getConnectionSpec());
    navigator.setMenuToMenuAddUpdtCeInputCMD("UPDATE");
    navigator.setMenuToMenuAddUpdtCeInputNAME1(this.getLastName());
    navigator.execute();

    String message = navigator.getMenuToMenuAddUpdtCeOutput1MSG();
    if (!message.equals("ENTRY WAS UPDATED"))
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}
```

3. Select the pull-down menu **Edit** → **Save**.

## Creating an Executable Class

- Select your package.
- From the **Selected** menu choose **Add** → **Class**, a wizard will appear to request all the necessary information required to create a class. Enter the current project and package in the appropriate fields in the wizard and ensure that the **Create a new class** radio button is selected. In the **Class** name field, enter Execute. The Superclass should be set to java.lang.Object. Ensure that the **Compose the class visually** radio button is **NOT** selected and click the **Next** button to continue to the next screen.
- There are three classes that should be imported when the executable is run. To include these classes as import statements, click the **Add Package** button. A list of available packages will appear. From the list, select each of the following and click **Add** to include them in the import statements: (after adding the last one, click **Close**).

```
com.ibm.connector.appc
com.ibm.connector.infrastructure>
com.ibm.connector.infrastructure.java
```

- Finally, ensure that the following fields are selected (checkmark beside them):
  1. **public** (in modifiers section)
  2. **Methods which must be implemented** (Recommended)
  3. **main(String[])**
- To generate the class, click the **Finish** button and the class appears inside the package you have specified.
- Enter the code listed below into the main(String[]) method created in the Execute class and select **Save** from the **Edit** pull-down menu. Note that the User and Password for the CICS server must be inserted into this code where CBUSER appears:

```
public static void main(java.lang.String[] args) {
    try {
        JavaRuntimeContext runtimeContext = new JavaRuntimeContext();
        ((DefaultLogonInfo) runtimeContext.getLogonInfo()).setUser("CBUSER");
        ((DefaultLogonInfo) runtimeContext.getLogonInfo()).setPassword("CBUSER");
        JavaRuntimeContext.setCurrent(runtimeContext);
        ((JavaRASService) runtimeContext.getRASService()).setTraceLevel(1);
        com.ibm.ivj.communications.Session.startSession();
        PhoneBookPAOKey key = new PhoneBookPAOKey();
        key.setAccount_ID("87654321");
        key.setfirstName("01");
        PhoneBookPAO bec = (PhoneBookPAO) key.getTarget(true);
        // Retrieve Cash Account information
        System.out.println("Retrieving ...");
        try {
            bec.retrieve();
        } catch (Exception e) {
            System.out.println("\n!!! Exception from Cash Account Retrieve " + e.toString());
            e.printStackTrace();
        }
        // com.ibm.ivj.communications.Session.endSession(true);
        // com.ibm.ivj.communications.Session.startSession();
        System.out.println("\n\n" + bec.toString());
        bec = (PhoneBookPAO) key.getTarget(false);
        System.out.println("\n\n" + bec.toString());
        PhoneBookPAOKey key1 = new PhoneBookPAOKey();
        key1.setAccount_ID("97654321");
        key1.setfirstName("01");
        PhoneBookPAO bec1 = (PhoneBookPAO) key1.getTarget(true);
```

```

bec1.setUtilities("This is the utilities");
bec1.setType("2");
bec1.setlastname(100);
// Create a Cash Account
System.out.println("Creating ...");
    try {
        bec1.insert();
    } catch (Exception e) {
        System.out.println("\n!!! Exception from Cash Account Insert " + e.toString());}
//      ((com.ibm.connector.infrastructure.java.JavaCoordinator)
        runtimeContext.getCoordinator()).commit();
//bec1 = (PhoneBookPA0) key1.getTarget(false);
System.out.println("\n\n" + bec1.toString());
// com.ibm.ivj.communications.Session.endSession(true);
// com.ibm.ivj.communications.Session.startSession();
// Update a Cash Account
System.out.println("Updating ...");
bec1 = (PhoneBookPA0) key1.getTarget(true);
bec1.setUtilities("Changed Utilities");
bec1.setType("B");
bec1.setlastname(300);
    try {
        bec1.update();
    } catch (Exception e) {
        System.out.println("\n!!! Exception from Cash Account Update " + e.toString());
        e.printStackTrace();}
//      ((com.ibm.connector.infrastructure.java.JavaCoordinator)runtimeContext.getCoordinator()).commit();
//bec1 = (PhoneBookPA0) key1.getTarget(false);
System.out.println("\n\n" + bec1.toString());
// com.ibm.ivj.communications.Session.endSession(true);
// com.ibm.ivj.communications.Session.startSession();
// Delete a Cash Account
System.out.println("Deleting ...");
    try {
        bec1.del();
    } catch (Exception e) {
        System.out.println("\n!!! Exception from Cash Account Delete " + e.toString());
        e.printStackTrace();}
//      ((com.ibm.connector.infrastructure.java.JavaCoordinator)
//      runtimeContext.getCoordinator()).commit();
// com.ibm.ivj.communications.Session.endSession(true);
// com.ibm.ivj.communications.Session.startSession();
key = new PhoneBookPAOKey();
key.setAccount_ID("97654321");
key.setfirstName("01");
bec = (PhoneBookPA0) key.getTarget(true);
// Retrieve Cash Account information
System.out.println("Retrieving ...");
    try {
        ((PhoneBookPA0)key.getTarget(true)).retrieve();
    } catch (Exception e) {
        System.out.println("\n!!! Expected Exception from
        Cash Account Retrieve " + e.toString()); }
//      ((com.ibm.connector.infrastructure.java.JavaCoordinator)
//      runtimeContext.getCoordinator()).commit();
// com.ibm.ivj.communications.Session.endSession(true);
runtimeContext.close();
}

```

```

    catch (Exception e)
    {
        e.printStackTrace();
        System.out.println("Error is " + e);
    }
}

```

## Exporting the PBE Package

After building the Execute class and creating and testing the Component Broker procedural adaptor object within the VisualAge for Java environment, you can run the unit test program outside of the VisualAge for Java environment. This object needs to be imported to Object Builder as a persistent object. Importing this object requires that the procedural adaptor object and its corresponding BeanInfo class is exported outside of VisualAge for Java. To run the sample outside of the VisualAge for Java environment, you must export all classes you created and modify the CLASSPATH environment variable.

For ease, export the entire package. This package should contain:

- The new procedural adapter object
- Its corresponding BeanInfo class
- All EAB transaction objects

To export the package outside of VisualAge for Java:

1. Select the `paa.mysamples.ims.pbe` package to export.
2. From the VisualAge for Java Workbench menu, select **File** → **Export**. Select the **Directory** radio button and click **Next**.
3. Select the **Both Java and Class Files** check box. This opens the Type of Export - wizard. You are going to export all .java and .class files.
4. Click the **Next** button.
5. To verify that you exported the package correctly, you can run the unit test program from the command line.

**WIN** For Windows NT, type `x:\MyProj` in the **Directory** field. This will be your Working Directory.

**AIX** For AIX, type `$HOME/MyProj` in the **Directory** field. This will be your Working Directory.

6. Click the **Finish** button.

When the export completes, the `paa.mysamples.ims.pbe` package is created under the MyProj directory.

To verify that you exported the package correctly, you can run the unit test program from the command line.

1. Ensure that your Working Directory is in your CLASSPATH.

**WIN** From a command prompt, for Windows NT, enter:

```
java paa.mysamples.ims.pbe.Execute -nojit
```

**AIX** From a command prompt, for AIX, enter:

```
java paa.mysamples.ims.pbe.Execute
```

You should have the same results as you did when running inside of VisualAge for Java.

---

## Developing an IMS-HOD Business Object

This section contains Object Builder and System Management procedures required to create a component named "PhoneBookEntry." To create this component, perform the procedures in the following sections.

1. "Importing the Bean" on page 88
2. "Defining the PhoneBookEntry Component" on page 89
3. "Creating Client and Server DLL Files" on page 93
4. "Packaging the Application" on page 94
5. "Building the Application - Client and Server" on page 96
6. "Installing the Application" on page 97
7. "Running the Sample Application" on page 99

### Notes:

1. Before starting Object Builder, ensure that your classpath includes your Working Directory.
2. When you start Object Builder, you must select **Enable Object Builder beta capability** when the Open Project dialog is displayed in Object Builder.
3. Specify your Working Directory as the base directory for the project.
4. The procedures contained in this section assume that you have correctly set your classpath to include your Working Directory before starting Object Builder and that you have started Object Builder.

## Importing the Bean

The bean to import is PhoneBookBean from the paa.mysamples.ims.pbe package from your Working Directory.

To import this bean:

1. Select the User-Defined PA Schemas folder from the Object Builder Tasks and Objects pane.
2. Open the pop-up menu for User-Defined PA Schemas, and select **Import Bean**. This opens the Import Procedural Adaptor Bean - wizard.
3. On this page:
  - a. Type paa.mysamples.imshod.pbe.PhoneBookPA0 in the **Class Name** field.
  - b. Click the **Next** button to accept the remaining defaults and to continue.
4. On this page:
  - a. Leave the **Module Name** field blank.
  - b. Leave the default value in the **Persistent Object Name** field.
  - c. Select **HOD** for the Connector Type.
  - d. Click the **Next** button to continue.
5. On this page:
  - a. Select the lastName property from the **Properties** list box.
  - b. Click the >> button to move the associated key required to import the bean.
6. Click the **Finish** button.

The bean is imported into Object Builder. The PhoneBookPA0 schema and its corresponding persistent object (PhoneBookBeanPO) are now in the tree view of User-Defined PA Schemas.

## Defining the PhoneBookEntry Component

This exercise defines the objects required to create a component named “PhoneBookEntry.” For this component, you will:

1. Create a new business object file
2. Define the business object
3. Connect the data object implementation to the persistent object
4. Define the managed object
5. Generate the code

### Creating the Business Object File

To create the PhoneBookEntry business object file:

1. From the Tasks and Objects pane, select the User-Defined Business Objects folder.
2. Open the pop-up menu for User-Defined Business Objects, and select **Add File**, which opens the Business Object File - wizard to the Name and Attributes page.
3. On this page:
  - a. Type PhoneBookEntry in the **Name** field.
  - b. Accept the other defaults.
4. Click the **Finish** button.

The PhoneBookEntry file is now under the User-Defined Business Objects folder.

### Defining the Business Object

After creating the new business object file, the business object needs to be defined. A fully-configured business object consists of the following:

- A business object interface
- An associated key
- An associated copy helper
- A business object implementation

**Defining the Business Object Interface:** To create the PhoneBookEntry business object interface:

1. Expand the User-Defined Business Objects folder, and select **PhoneBookEntry**.
2. Open the pop-up menu for PhoneBookEntry, and select **Add Interface**, which opens the Business Object Interface - wizard to the Name and Attributes page.
3. On this page:
  - a. Type PhoneBookEntry in the **Name** field.
  - b. Click the **Next** button to continue to the Constructs page.
4. Click the **Next** button to accept the defaults and to continue to the Interface Inheritance page.
5. Click the **Next** button to accept the defaults and to continue to the Attributes page.
6. Define the user-defined attributes.
  - a. Select Attributes from the tree view.
  - b. Open the pop-up menu for Attributes, and select **Add**. This displays the Add dialog.
  - c. In this dialog:
    - 1) Type lastName in the **Attribute Name** field.

- 2) Select **string** as the **Type**. This displays the Size field.
  - 3) Type 0 in the **Size** field.
  - 4) Click the **Add Another** button.
- d. Repeat the previous step for the remaining attributes of the PhoneBookEntry interface. The remaining attributes are:
- *firstName*, and click the **Add Another** button.
  - *extNumber*, and click the **Add Another** button.
  - *internalZip*, and click the **Refresh** button.
- e. Click the **Next** button to continue to the Methods page.
7. Define the user-defined methods.
- a. Right-click on Methods from the tree view.
  - b. From the pop-up menu for Methods, select **Add**, which opens the editor pane.
  - c. In the this pane:
    - 1) Type showAll in the **Method Name** field.
    - 2) Click the **Refresh** button.
  - d. Right-click on Parameters from the tree view.
  - e. From the pop-up menu for Parameters, select **Add**, which opens the editor pane.
  - f. In this pane:
    - 1) Type 1nm in the **Parameter Name** field.
    - 2) Select string as the **Type**. This displays the Size field.
    - 3) Type 0 in the **Size** field.
    - 4) Set the **Out** radio button.
    - 5) Click the **Add Another** button.
  - g. Repeat the previous step for the remaining parameters. These parameters are:
    - *fnm*, and click the **Add Another** button.
    - *ext*, and click the **Add Another** button.
    - *zip*, and click the **Refresh** button.
8. Click the **Finish** button.

The PhoneBookEntry interface is now under the PhoneBookEntry file.

**Defining the Key:** To add the key:

1. From the User-Defined Business Object folder, select the PhoneBookEntry interface.
2. Open the pop-up menu for PhoneBookEntry, and select **Add Key**, which opens the Key - wizard.
3. Select the *lastName* attribute from the **Business Object Attributes** list.
4. Click the **>>** button to move this attribute to the **Key Attributes** list.
5. Click the **Finish** button.

The **PhoneBookEntryKey** key is now under the PhoneBookEntry interface.

**Defining the Copy Helper:** To add the Copy Helper:

1. From the User-Defined Business Object folder, select the PhoneBookEntry interface.
2. Open the pop-up menu for PhoneBookEntry, and select **Add Copy Helper**, which opens the Copy Helper - wizard.

3. Click the **All>>** button to move the attributes from the **Business Object Attributes** list to the **Copy Helper Attributes** list.
4. Click the **Finish** button.

The PhoneBookEntryCopy copy helper is now under the PhoneBookEntry interface.

**Defining the Business Object Implementation:** To add the business object implementation and data object interface:

1. From the User-Defined Business Object folder, select the PhoneBookEntry interface.
2. Open the pop-up menu for PhoneBookEntry, and select **Add Implementation**, which opens the Business Object Implementation - wizard to the Name and Data Access Pattern page.
3. Define the implementation.
  - a. Select the **Delegating** radio button from the **Pattern for Handling State Data** group box.
  - b. Ensure that the **Create a new one now** radio button is selected from the **Data Object Interface** group box. This option allows you to define the business object attributes that need to be preserved in the data object.
  - c. Deselect 390 in the Select deployment platform group box.
  - d. Click the **Next** button to continue to the Implementation Inheritance page.
4. Click the **Next** button to accept the defaults and to continue to the Implementation Language page.
5. Click the **Next** button to accept the defaults and to continue to the Attributes page.
6. Click the **Next** button to accept the defaults and to continue to the Methods page.
7. Click the **Next** button to accept the defaults and to continue to the Key and Copy Helper page.
8. On this page:
  - a. Verify that the **PhoneBookEntryKey** key is selected from the **Key** list.
  - b. Verify that PhoneBookEntryCopy is selected from the **Copy Helper** list.
  - c. Click the **Next** button to continue to the Handle Selection page.
9. Click the **Next** button to accept the defaults and to continue to the Attributes to Override page.
10. Click the **Next** button to accept the defaults and to continue to the Methods to Override page.
11. Click the **Next** button to accept the defaults and to continue to the Data Object Interface page.
12. Click the **All>>** button to move the attributes in the **Business Object Attributes** list to the **State Data** list.
13. Click the **Finish** button.

The PhoneBookEntryBO business object implementation is now under the PhoneBookEntry interface, and the PhoneBookEntryDO data object interface is now under the PhoneBookEntryBO business object implementation.

## Connecting the Data Object Implementation to the Persistent Object

To create the data object implementation and to connect the data object implementation to the persistent object, perform the following procedure.

1. From the User-Defined Business Object folder, select the PhoneBookEntryDO data object interface.
2. Open the pop-up menu for PhoneBookEntryDO, and select **Add Implementation**. This displays the Data Object Implementation - wizard.

3. Deselect **390** in the Select Deployment platform group box and click the **Next** button to continue to the Behavior page.
4. Set the environment.
  - a. Set the **BOIM with any key** radio button from the **Environment** group box to indicate that the data object is part of a component installed in a business object application adaptor with instances being located by key objects.
  - b. Set the **Procedural Adaptors** radio button from the **Form of Persistent Behavior and Implementation** group box.
  - c. Click the **Next** button to continue to the Implementation Inheritance page.
5. On this page:
  - a. Verify that IPAAExtLocalToServer IPAAExtLocalToServer is selected as a parent.
  - b. Click the **Next** button to continue to the Attributes page.
6. Click the **Next** button to accept the defaults and to continue to the Methods page.
7. Click the **Next** button to accept the defaults and to continue to the Key and Copy Helper page.
8. Click the **Next** button to accept the defaults and to continue to the Associated Persistent Objects page.
9. On this page:
  - a. Select **Persistent Object Instances**.
  - b. Open the pop-up menu for Persistent Object Instances, and select **Add**.
  - c. Type iPhoneBookBeanPAOPO in the **Instance Name** field.
  - d. Click the **Next** button.
10. On this page:
  - a. Select lastName from the **Attributes** list.
  - b. Open the pop-up menu for lastName, and select **Primitive**.
  - c. Select **iPhoneBookBeanPAOPO.lastName** from the **Persistent Object Attribute** list.
  - d. Add 1-to-1 mappings for the other attributes under the **Attributes** tree view as you just did for lastName.
  - e. Click the **Next** button.
11. On this page:
  - a. Select **insert** from the **Special Framework Methods** list.
  - b. Open the pop-up menu for insert, and select **Add Mapping**.
  - c. Select **iPhoneBookBeanPAOPO.insert** from the **Persistent Object Method** list.
  - d. Add 1-to-1 mappings for the other methods under the **Special Framework Methods** tree view as you just did for insert. In addition, add a mapping for the setConnectin() method.
12. Click the **Finish** button.
13. If you receive the prompt One or more data objects are not mapped to the persistent object. Do you want to continue? Click the **Yes** button.

The PhoneBookEntryDOImpl data object implementation is now under the PhoneBookEntryDO interface, and the PhoneBookBeanPO persistent object is now under the PhoneBookEntryDOImpl data object implementation.

## Defining the Managed Object

To add the managed object:

1. From the User-Defined Business Object folder, select the PhoneBookEntryBO business object implementation.
2. Open the pop-up menu for PhoneBookEntryBO, and select **Add Managed Object**, which opens the Managed Object - wizard to the Name and Service page.
3. Deselect **390** in the Select Deployment platform group box.
4. Select the session service.
5. Click the **Next** button to accept the defaults and continue to the Implementation Inheritance page.
6. Click the **Finish** button.

## Generating the Code

To generate the application code:

1. From the User-Defined Business Object folder, select PhoneBookEntry.
2. Open the pop-up menu for PhoneBookEntry, and select **Generate** → **All**.

Code generation starts. Progress is indicated in the lower-left corner of the window.

## Creating Client and Server DLL Files

The defined objects need to be built into two separate DLL files.

- One that runs on the client and provides access to the business object interface, key and copy helper.
- One that runs on the server and provides access to the managed object and the rest of the component.

The client DLL file needs to be defined before the server DLL file. When the server DLL file is defined, it needs to link to the client DLL file. After defining the objects that comprise each DLL file, these files can be built.

## Defining the Client DLL File

To add the client DLL file:

1. Select the Build Configuration folder.
2. Open the pop-up menu for Build Configuration, and select **Add client DLL**. This displays the Name and Option page of the Add Client DLL - wizard.
3. Type pbeC in the **Name** field.
4. Click the **Next** button to continue to the Client Source Files page.
5. Click the **All>>** button to move the client source files to the **Items chosen** list.
6. Click the **Finish** button.

The pbeC client DLL file is now under the Build Configuration folder.

## Defining the Server DLL File

To add the server DLL.

1. Select the Build Configuration folder.
2. Open the pop-up menu for Build Configuration, and select **Add Server DLL**. This displays the Name and Option page of the Server DLL - wizard.
3. Type pbeS in the **Name** field.
4. Deselect **390** in the Select Deployment platform group box.
5. Click the **Next** button to continue to the Server Source Files page.
6. Click the **All>>** button to move the server source files to the **Items chosen** list.
7. Click the **Next** button to continue to the Libraries to Link With page.
8. Click the **All>>** button to move all the files from the **Items Available** list to the **Items chosen** list.
9. Click the **Finish** button.

The pbeS server DLL file is now under the Build Configuration folder.

## Building the DLL Files

To generate the makefiles to build the configuration:

1. Select the Build Configuration folder.
2. Open the pop-up menu for Build Configuration, and select **Generate** → **All** → **All Targets**.

The code generation begins.

## Packaging the Application

Packaging the application consists of the following procedures:

1. Creating the application family
2. Defining the application
3. Creating the container instance
4. Configuring the managed object
5. Generating the application

## Creating the Application Family

To add the application family:

1. Select the Application Configuration folder.
2. Open the pop-up menu for Application Configuration, and select **Add Application Family**. This displays the Name page of the Application Family - wizard.
3. Type pbeAppFam in the **Name** field.
4. Click the **Finish** button.

The pbeAppFam application family is now under the Application Configuration folder.

## Defining the Application

To add the Application:

1. Select the pbeAppFam application family.
2. Open the pop-up menu for pbeAppFam, and select **Add Application**, which opens the Add Application - wizard to the Name and Environment page.
3. Type pbeApp in the **Application Name** field.
4. Click the **Finish** button.

The pbeApp application is now under the pbeAppFam application family.

## Creating the Container Instance

To add the new container instance:

1. Select the Container Definition folder.
2. Open the pop-up menu for Container Definition, and select **Add Container Instance**, which opens the Container - wizard.
3. Type pbeContainer in the **Name** field.
4. Click the **Next** button to continue to the Work Load Manager Container page.
5. Click the **Next** button to continue to the Policies and Services page.
6. On this page:
  - a. Set the **Use PAA Services** radio button.
  - b. Set the **Session Services** radio button.
  - c. Click the **Next** button to continue to the Sessional Policies and Services page.
7. On this page:
  - a. Set the **Throw an exception and abandon the call** radio button under **Behavior for Methods Called Outside a Session**.
  - b. Set the **Host on Demand** radio button under **Type of Connection used by Session**.
  - c. Type IMS\_pbe\_Connection in the **Connection Name** field.
  - d. Click the **Next** button to continue to the Data Access Patterns page.
8. On this page, ensure that the **Delegating** check box is set under both **Business Object** and **Data Object** blocks.
9. Click the **Finish** button.

The pbeContainer container is now under the Container Definition folder.

## Configuring the Managed Object

To add the managed object for the Application:

1. Select the **pbeApp** application.
2. Open the pop-up menu for pbeApp, and select **Add Managed Object**, which opens the Configure Manage Object - wizard.
3. In this window:
  - a. Verify that PhoneBookEntryMO PhoneBookEntryMO is in the **Managed Object** field.

- b. Click the **Next** button to continue to the Data Object Implementations page.
4. On this page:
  - a. Select **Implementation**.
  - b. Open the pop-up menu for Implementation, and select **Add**.
  - c. Select **PhoneBookEntryDOImpl PhoneBookEntryDOImpl** from the **Data Object Implementation** list.
  - d. Click the **Next** button to continue to the Container page.
5. Click the **Next** button to continue to the Home page.
6. On this page, select BOIMHomeOfRegHomes from the **Home Name** list.
7. On this page, select **pbeContainer** from the **Name** list.
8. Click the **Finish** button.

The PhoneBookEntryMO managed object is now under the Application Configuration folder.

## Generating the Applications

To generate the application family:

1. Select the pbeAppFam application.
2. Open the pop-up menu for pbeApp, and select **Generate**.

**Note:** If you do not have InstallShield installed on your system, Click the **Yes** button when the dialog concerning InstallShield is displayed.

When code generation completes, the Method Implementation pane contains the pbeApp.ddl file. You can now close Object Builder.

## Building the Application - Client and Server

### Building the Application - Client and Server

**WIN**

For Windows NT, all imported and generated files are placed in the `x:\MyProj\Working\NT` directory (where `x:\MyProj` is the working directory when Object Builder was started).

1. Change directory to:
 

```
x:\MyProj\Working\NT
```
2. Enter:
 

```
nmake -f all.mak cpp java
```

**AIX** For AIX, all imported and generated files are placed in the `$HOME/MyProj/Working/AIX` directory (where `$HOME/MyProj` is the working directory when Object Builder was started).

```
$HOME/MyProj/Working/AIX
make -f all.mak cpp java
```

Everything in the sample application is built.

## Installing the Application

Installing an application consists of:

1. Loading the application
2. Configuring the application

These procedures assume that you are currently logged on to DCE and that you are currently using the System Manager User Interface. If not, logon to DCE and start the System Manager User Interface.

## Loading the Application onto System Management

To install the pbe server application:

1. Start the System Manager User Interface, if it is not already started.
2. Become an Expert user (**View** → **User Level** → **Expert**).
3. Expand Host Images, and select <your host name>.
4. From the pop-menu, select **Load Application**. The opens the Load Application dialog.
5.  **WIN** For Windows NT, browse for and select **pbcAppFam.ddl**

(x:\MyProj\Working\NT\pbeAppFam\pbeAppFam.ddl).  **AIX** For AIX, browse for and select **pbcAppFam.ddl** (\$HOME/MyProj/Working/AIX/pbeAppFam/pbeAppFam.ddl)

**Note:** A warning may be displayed about iCachedWLMSystemManagedObjects while the DDL is loading. You can ignore this warning.

## Configuring the Application with System Management

To configure the application:

1. Configure the pbeApp application.
  - a. Expand Available Applications, and select pbeApp.
  - b. Open the pop-up menu for pbeApp, and select **Drag**.
  - c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select Sample Configuration.
  - d. Open the pop-up menu for Sample Configuration, and select **Add Application**.
2. Configure the HOD connection.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → HOD Connections, and select IMS\_pbe\_Connection.
  - b. Open the pop-up menu for IMS\_pbe\_Connection, and select **Edit**, which opens the Object Editor.
  - c. Click the **Main** tab.
  - d. Modify the **host name** and **port number** fields to match the IMS region with which you are communicating.
  - e. Change the security mechanism to IMS.
  - f. Click the **OK** button.
3. Define the server.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select Sample Configuration.
  - b. Open the pop-up menu of Sample Configuration, and select **New** → **Server (free standing)**. This displays a new dialog box.

- c. Type pbeSrv as the name for the server group.
  - d. Click the **OK** button. The pbeSrv is now under Server (free standing).
4. Associate the application with the server.
- a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Applications, and select pbeApp.
  - b. Open the pop-up menu of pbeApp, and select **Drag**.
  - c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Server (free standing), and select pbeSrv.
  - d. Open the pop-up menu of pbeSrv, and select **Configure Application**.
5. Associate the iPAAServices application with the server.
- a. Host Images → *myhost* → Application Family Installs → iPAAApplications → Application Installs, and select iPAAServices.
  - b. Open the pop-up menu for IPAAServices, and select **Drag**.
  - c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select Sample Configuration.
  - d. Open the pop-up menu for Sample Configuration, and select **Add Application**.
  - e. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Applications, and select iPAAServices.
  - f. Open the pop-up menu for IPAAServices, and select **Drag**.
  - g. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Server (free standing), and select pbeSrv.
  - h. Open the pop-up menu for pbeSrv, and select **Configure Application**.
6. Configure the server with the host.
- a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Server (free standing), and select pbeSrv
  - b. From the pop-up menu for pbeSrv, select **Drag**.
  - c. Expand Hosts, and select your server.
  - d. From the pop-up menu for your server, select **Configure Server (free standing)**.
7. Enable security services for the server.
- a. Expand Management Zones → Sample Cell and Work Group Zone → Configuration → Sample Configuration → Server (free standing), and select pbeSrv.
  - b. Open the pop-up menu for pbeSrv, and select Edit, which opens the Object Editor.
  - c. In this notebook:
    - 1) Select the **Security Service** tab.
    - 2) Change the value for the **data system principal** field to the user ID that the server will use when connecting to the IMS system.
    - 3) Change the value for the **data system password** field to the password that the server will use when connecting to the IMS system.
    - 4) Change the value for the **security enabled** field from no to yes.
    - 5) Click the **OK** button. The changes are applied and the Object Editor closes.

8. Enable security services for the client.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configuration → Sample Configuration → Client Styles, and select **myClient** .
  - b. Open the pop-up menu for **myClient**, and select Edit, which opens the Object Editor.
  - c. In this notebook:
    - 1) Select the **Security Service** tab.
    - 2) Change the value for the **security enabled** field from no to yes.
    - 3) Click the **OK** button. The changes are applied and the Object Editor closes.
9. Activate the configuration.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select Sample Configuration.
  - b. Open the pop-up menu for Sample Configuration, and select **Activate**, which automatically starts the application server. Wait for a completion message in the Action Console window before continuing.

## Running the Sample Application

For IVP install instructions for IMS, see *IMS/ESA Version 6 Install Volume 1*. The entire book contains information on installing and configuring the IVP sample. Chapter 11, entitled “Install/IVP Application,” discusses the sample IMS application.

**WIN** To run the sample client application:

1. Copy the pbeclient.mak and pbeclient.cpp from:
 

```
x:\CBroker\samples\PAAsamples\Application\PhoneBookCli
```

 to:
 

```
x:\MyProj\Working\NT
```
2. Change directory to:
 

```
x:\MyProj\Working\NT
```
3. Enter:
 

```
nmake -f pbeclient.mak
```
4. Enter:
 

```
pbeclient
```

**AIX**

1. Copy the pbecclient.mak and pbeclient.cpp from:
 

```
$HOME/samples/PAAsamples/Application/PhoneBookCli
```

 to:
 

```
$HOME/MyProj/Working/AIX
```
2. Change directory to:
 

```
$HOME/MyProj/Working/AIX
```
3. Enter:
 

```
make -f pbeclient.mak
```

4. Enter:

pbeclient

---

## Chapter 6. Developing a CICS-HOD Application

This chapter provides information for building a sample Component Broker application with a CICS backend. This information, for the most part, can be pertinent for building an IMS-backend Component Broker application.

This chapter contains the following information:

- “The CICS Sample Application”
- “Enterprise Access Builder Procedures” on page 103
- “Developing a CICS-HOD Business Object” on page 132

**Note:** To walk-through this sample, the CICS and IMS Application Adaptor SDK and IBM VisualAge Java with EAB must be installed.

### Important Information

Before walking through this sample, please refer to the *Late Breaking News* provided with Component Broker before performing the exercise in this chapter. This document provides the latest information regarding the CICS and IMS application adaptor samples, which may differ from the instructions for this sample application.

---

## The CICS Sample Application

The CICS-HOD sample application is based on a CICS Installation Verification Procedure (IVP). The IVP is a mock customer account database consisting of the following fields:

- number** The account number of the customer. This attribute is used as the key.
- name** The name of the customer.
- address** The address of the customer.
- phone** The phone number of the customer.
- date** The date of the last update to the record.
- amount** The balance that the customer has in his or her account.
- comment** Any comment about the record.

For more information about this sample, see the following:

WIN For Windows NT

`CBroker\samples\InstallVerification\PAA\readme.htm`

AIX For AIX

`$HOME/samples/InstallVerification/PAA/readme.htm`

This sample application works on a CICS flat file and permits adding, updating, and inquiring the account through the ADDS, UPDT, and INQY transactions.

Although this sample application is not a full-blown CICS application, it captures the essence of an application involving multiple 3270 screen navigations and delivering some amount of business function.

This sample application can be extended and customized to explore different CICS-HOD application issues.

## Interacting with the CICS IVP

Using the CICS IVP involves navigating a sequence of 3270 screens. The following sequence takes you through a full cycle for the ADDS transaction.

1. If your CICS server uses the IBM Transaction Server for the Windows NT environment, the *Installation Guide* discusses the required steps to create a telnet server listening port in Chapter 5, "Connecting Telnet Clients." Issue the following command on the CICS server Windows NT system:

```
cicscp create telnet_server TNSERVER -P 7777 -r CBCICS
```

where 7777 is an unused port number on the system. The cicscp command is fully described in the "cicscp - telnet server commands" in the Administration Reference (CICS).

2. Start your 3270 emulator session from the client system.

3.  WIN For Windows NT you type the following:

```
cicsterm /S=hosthame
```

AIX For AIX you type the following: `xant hosthame`

4. In the 3270 emulator, type:

```
MENU
```

**Note:** Some 3270 emulators use the **Ctrl** key on the right side of the keyboard instead of the **Enter** key. The transaction is started, and the next window is displayed.

5. At this window, the following entry fields are displayed:

- **ENTER TRANSACTION**
- **NUMBER**
- **AMOUNT**

At this window:

- a. Type ADDS in the **TRANSACTION** field.
- b. Type 111111 in the **NUMBER** field.
- c. Press the **Enter** key.

This displays a new window.

6. This window allows you to add the input fields for the new account record. Press the **Enter** key when finished adding the field.

7. This window contains the menu screen again. If the account was added successfully, the following message appears in the message field:

```
RECORD ADDED
```

If the account was already established, the following message appears in the message field:

```
DUPLICATE RECORD
```

8. Clear the screen to finish the navigation to reset the terminal to accept new transactions.

All transactions follow the same sequence for a full cycle. The full cycle is described, because in this sample IBM VisualAge Java with Enterprise Access Builder (EAB) transaction objects are used to perform these IVP transactions, and each of these navigates through a full cycle.

---

## Enterprise Access Builder Procedures

This exercise defines the classes required to create a Component Broker procedural adapter object (PAO) named “MenuCustomer.” For this object, you will be:

1. “Creating a Project and Package for the Samples” on page 103.
2. “Creating the Procedural Adapter Object and Key” on page 104.
3. “Creating Command Beans” on page 111.
4. “Creating Navigator Beans” on page 120.

### Creating a Project and Package for the Samples

To create a new project and package under VisualAge for Java:

1. Start VisualAge for Java.

**WIN** For Windows NT, from the Windows NT **Start** menu, select **IBM VisualAge for Java for Windows** → **IBM VisualAge for Java**.

**AIX** For AIX, type `vajide` on the command line.

Verify that all of the following projects are listed in the VisualAge for Java workspace.

- Connector CICS 3.0
- IBM Common Connector Framework V2.0
- IBM Component Broker Host On Demand 1.0
- IBM Component Broker Connectors 1.0
- IBM Procedural Application Adapter 1.0
- IBM Enterprise Access Builder Library V2.0
- IBM Java Record Library V2.0

If any projects are missing, add them as follows:

- a. Select **File** → **Quick Start**.
- b. Select **Features** in the left pane and **Add Feature** in the right pane.
- c. Click the **OK** button.
- d. Select the projects listed above.
- e. Click the **OK** button.

**Note:** The sample EAB objects described in this section are contained in the “IBM Component Broker PAA Samples” project in the `paa.samples.cics.menu` package. If you want to see the completed sample, use the “Add Feature” to load IBM Component Broker PAA Samples into your VAJ workspace.

2. Select the pull-down menu **Window**→**Options....** Select the Design Time options window and uncheck the **Inherit BeanInfo** of bean superclass. Also, you can change the colors to suit your taste using the Appearance options windows.
3. Create the **paa.mysamples.cics.menu** package, if it does not exist.
  - a. From the list of projects, right-click on the **CBSamples** project to open the pop-up menu. Select **Add** → **Package....**
  - b. Type `paa.mysamples.cics.menu` for the new package, and click the **Finish** button.

## Creating the Procedural Adapter Object and Key

In this section, you will create the Procedure Adapter Object (PAO), and create the key for the PAO, and link the PAO with its key.

The PAO inherits from `com.ibm.ivj.eab.paa.EntityProceduralAdapterObject`, which serves as a base implementation for all PAOs. As a subclass of `EntityProceduralAdapterObject`, the PAO contains the CRUD methods (Create, Retrieve, Update, and Delete). However, these methods are all empty-bodied. You must define their implementation for your PAO.

The properties defined in the PAO interface must contain the data attributes that the Component Broker data object requires from the backend system. Also, the PAO must contain any *push-down* methods that the Component Broker data object requires for running special procedures on the backend system.

The PAO key must contain the data attributes used by the Component Broker data object to select the data on the backend system.

## Creating the MenuCustomer Class

1. Right-click on `paa.mysamples.cics.menu`, and select **Add** → **Class...**
2. In this dialog:
  - a. Type the **Class Name**: `MenuCustomer`
  - b. Click the **Browse** button to set the **Superclass** to `com.ibm.ivj.eab.paa.EntityProceduralAdapterObject` as described in the **Select class** instructions below.
3. Click the **Finish** button.

### Using the Select class dialog::

1. In the **Pattern** field, type the first few letters of the class name.
2. In the **Type Names**: list, click on the desired class name.
3. In the **Package Names**: list, if more than one package appears, click on the desired package name.
4. Click the **OK** button to close the dialog.

## Adding Properties to the MenuCustomer Class

1. Right-click the **MenuCustomer class** and select **Open**. This opens a new notebook.
2. Select the **BeanInfo** tab in the notebook.
3. Use the following procedure to add properties to the class.
  - a. Select the pull-down menu **Features** → **New Property Feature**. This opens the New Property Feature dialog.
  - b. Type the name of the property you want to add in the **Property Name** field.
  - c. Leave **java.lang.String** as the Property type.
  - d. Click the **Finish** button.

Repeat this procedure to add the following properties:

- **name**
- **address**
- **phone**

- **date**
- **amount**
- **comment**

Use the default property type `java.lang.String` for each of the properties.

Also add the property: **number**. However, because the **number** is *key data*, you must uncheck the **Writable** check box in the New Property dialog box for the **number** property.

4. Add the **debit** push-down method as follows:
  - a. Select the pull-down menu **Features** → **New Method Feature**. This opens the New Method Feature dialog.
  - b. Type the Method name: `debit`
  - c. Set the **Parameter count**: to 1.
  - d. Click the **Next** > button.
  - e. Set the **Parameter name**: to `amount`
  - f. Set the **Parameter type**: to `int`
  - g. Click the **Finish** button.
1. Verify that the list of MenuCustomer properties matches the following list.
  - P `address` R W B
  - P `amount` R W B
  - P `comment` R W B
  - P `date` R W B
  - P `name` R W B
  - P `number` R B
  - P `phone` R W B
  - M `debit(int)`

WIN For Windows NT, close the MenuCustomer notebook by clicking on the **X** button in the upper-right corner of the window.

AIX For AIX, click the > button in the upper left corner of the window and select **Close**.

## Creating the MenuCustomerKey Class

1. Right-click on `paa.mysamples.cics.menu`, and select **Add** → **Class....**
2. In this dialog:
  - a. Type the Class Name: `MenuCustomerKey`
  - b. Click the **Browse** button to set the **Superclass** to `BusinessObjectKey` in package `com.ibm.ivj.eab.businessobject` as described in the Select class instructions.
3. Click the **Finish** button.

## Adding Properties to the MenuCustomerKey Class

1. Right-click the **MenuCustomerKey** class and select **Open**. This opens a new notebook.
2. Select the **BeanInfo** tab in the notebook.
3. Add the **number** property to the class as follows:
  - a. Select the pull-down menu **Features** → **New Property Feature**. This opens the New Property Feature dialog.

- b. Type the **Property Name:** number
  - c. Select **java.lang.String** for the type of property.  
Leave everything else in the dialog as default. The **number** property needs to be writable.
  - d. Click the **Finish** button.
4. Verify that the MenuCustomerKey has only the following property. P **number** R W B
  5.  WIN For Windows NT, close the MenuCustomerKey notebook by clicking on the **X** button in the upper-right corner of the window.
  6.  AIX For AIX, click the > button in the upper left corner of the window and select **Close**.

## Linking the PAO and its Key Class

1. In the **Workbench** window, expand the **MenuCustomerKey** class by clicking the **+** button next to it.
2. Click on the **getPropertyValues()** method. This puts the method source in the **Source** pane.
3. Change the return statement in the **Source** pane as follows:

```
protected java.lang.Object[] getPropertyValues() {
    return new Object[] { this.getNumber() };
}
```

4. Press the **Ctrl-S** key to save the **getPropertyValues()** method.
  5. Collapse the **MenuCustomerKey** class by clicking the **-** button next to it.
  6. Expand the **MenuCustomer** class by clicking the **+** button next to it.
  7. Click on the **getNumber()** method. This put the method source in the **Source** pane.
  8. Edit the **getNumber()** method so that it delegates to its key object as follows,
- ```
public String getNumber() {
    MenuCustomerKey key = (MenuCustomerKey) this.getKey();
    return key.getNumber();
}
```
9. Press the **Ctrl-S** key to save the **getNumber()** method.
  10. Collapse the **MenuCustomer** class by clicking the **-** button next to it.

## Creating Record Beans and a Record Mapper

In this section you will create two **record beans**. A record bean represents records in the CICS system. In the case of HOD, the record data is taken from the CICS application screens, so the record beans correspond to screens.

The two screens in this sample are as follows:

**DFHDGA** is the main transaction menu screen. It looks like the following:

```

                INSTRUCTION C version
OPERATOR INSTR - ENTER MENU
FILE INQUIRY   - ENTER INQY AND NUMBER
FILE BROWSE    - ENTER BRWS AND NUMBER
FILE ADD       - ENTER ADDS AND NUMBER
FILE UPDATE    - ENTER UPDT AND NUMBER
FILE DELETE    - ENTER DELE AND NUMBER
CUSTOMER DEBIT - ENTER DEBT AND NUMBER AND AMOUNT
PRESS CLEAR TO EXIT
ENTER TRANSACTION: ____ NUMBER ____ AMOUNT ____

```

**DFHDGB** is the customer data screen. It looks like the following.

```

                FILE ADD/UPDATE/INQUIRY
NUMBER: 123456
NAME: _____
ADDRESS: _____
PHONE: _____
DATE: _____
AMOUNT: $0000.00
COMMENT: _____

```

For each screen record bean, import the **BMS** file to generate the **dynamic record type**. Then use the record type to generate the record bean.

To exchange data between a record bean and a PAO, you must create a **record mapper**. The mapper describes the mapping between the properties of the PAO and the fields on the CICS application screens. In this sample, you will create a mapper to exchange data between **DFHDGB** and the **MenuCustomer** PAO.

The DFHDGA record bean is used by a command bean (“Creating Command Beans” on page 111 ), so a mapper is not needed.

## Creating the DFHDGA Record Type and Record Bean

1. Right-click on `paa.mysamples.cics.menu`, and select **Tools** → **Records** → **Create BMS RecordType...**
2. In the **Create BMS RecordType** wizard:
  - a. Type the **Class Name**: `DFHDGARecordType`
  - b. Click the **Add...** button to locate the file `dfhdga.bms`.
    - WIN** For Windows NT, the BMS files are located in  
`C:\Brokersamples\InstallVerification\PAA\Backend\Acct\`  
where **C:\CBroker** is the directory where you installed **IBM Component Broker**.
    - AIX** For AIX, the BMS files are located in  
`$HOME/samples/InstallVerification/PAA/Backend/Acct`  
where **\$HOME** is the directory where you installed **IBM Component Broker**.
  - c. Click the **Next** > button.
  - d. Click the > button. This causes **DFHDGA** to appear in the **Maps:** column.

- e. Select **DFHDGA** in the **Maps:** column.
- f. Click the other → button to move **DFHDGA** from the **Maps:** column to the **Selected:** column.
- g. Click the **Finish** button. This creates the class **DFHDGARecordType**.

**Important Note About Removing Record Type Initial Values:** In later sections, you will create navigator beans and you will see how the navigators decide which commands to execute by matching the CICS application screens with the record types. It is important, therefore, that the screen values in the record types are actual screen constants, i.e, they do not change. Otherwise, the navigator will not match the screen to the record and you will see an error such as:

IVJC0850: No output candidate matches data returned from the connector.

The MSG field of the DFHDGA screen has an initial value of "PRESS CLEAR TO EXIT," but its value can change. For example, after adding a record it says "RECORD ADDED SUCCESSFULLY." You need to change the DFHDGARecordType to remove the initial value of MSG so that the navigator does not use it for matching.

You could have modified the BMS file by removing the INITIAL string of MSG before you ran the Create BMS RecordType tool. Instead, complete the following steps to modify the generated code of the DFHDGARecordType.

3. Type the **Class Name:** DFHDGARecord
4. Leave the other fields as defaulted, and click the **Finish** button. This creates the classes **DFHDGARecord** and **DFHDGARecordBeanInfo**.
5. Expand the DFHDGARecordType class by clicking the + button next to it.
6. Select the DFHDGARecordType() constructor. The source code for the constructor appears in the Source pane in the lower half of the window.
7. Look for the following line of code (near the middle of the constructor).
 

```
addField(new Field(MSGType, "MSG",null,new java.lang.String("PRESS CLEAR TO EXIT"),true));
```

 Change the line of code to the following.
 

```
addField(new Field(MSGType, "MSG"));
```
8. Select the pull-down menu **Edit** → **Save**.
9. Collapse the DFHDGARecordType class by clicking the - button next to it.
10. Right-click on DFHDGARecordType, and select Tools → Records → Generate Records....

## Creating the DFHDGB Record Type and Record Bean

1. Right-click on paa.mysamples.cics.menu, and select Tools → Records → Create BMS RecordType....
2. In the Create a BMS RecordType SmartGuide:
  - a. Type the Class Name: DFHDGBRecordType
  - b. Click the **Add...** button to locate the file dfhdgb.bms. The BMS files are located in:
 

```
C:\CBroker\samples\InstallVerification\PAA\Backend\Acct\
```
  - c. where C:\CBroker is the directory where you installed IBM Component Broker.
  - d. Click the **Next** → button.
  - e. Click the → button. This causes DFHDGB to appear in the Maps: column.
  - f. Select DFHDGB in the Maps: column

- g. Click the other → button to move DFHDGB from the Maps: column to the Selected: column.
  - h. Click the **Finish** button. This creates the class DFHDGBRecordType.
3. Right-click on DFHDGBRecordType, and select Tools → Records → Generate Records....
  4. Type the Class Name: DFHDGBRecord
  5. Leave the other fields as defaulted, and click the **Finish** button. This creates the classes DFHDGBRecord and DFHDGBRecordBeanInfo.

## Creating the Record Mapper

In this section you create a record mapper that is used to exchange data between the **DFHDGBRecord** and the **MenuCustomer** PAO.

1. Right-click on the **DFHDGBRecord** class, and select **Tools** → **Mapper Editor...**
2. In the Mapper Editor window, select the pull-down menu **Input Bean View** → **Record view**. This displays the **DFHDGBRecord** record in the right-hand pane.
3. Click the **Add...** button and select the class MenuCustomer in package paa.mysamples.cics.menu as described in the **Select class** instructions.
4. In the following steps, you associate each property in the **MenuCustomer** with the corresponding field on the screen of the CICS application. Every association is bi-directional (read and write) except for the **number** property which is read-only, because it is a key.

Move the cursor slowly across the column of blank fields in the right-hand pane and a small hover window will show the name of each field. The names include NUMB, NAME, ADDR, PHONE, DATE, AMOUNT, and COMMENT.

- a. Click the **address** property and the **ADDR** field, and click the <—> button.
  - b. Click the **amount** property and the **AMOUNT** field, and click the <—> button.
  - c. Click the **comment** property and the **COMMENT** field, and click the <—> button.
  - d. Click the **date** property and the **DATE** field, and click the <—> button.
  - e. Click the **name** property and the **NAME** field, and click the <—> button.
  - f. Click the **number** property and the **NUMB** field, and click the <— button.
  - g. Click the **phone** property and the **PHONE** field, and click the <—> button.
5. Click the **OK** button.
  6. Click the **OK** button again. This creates the DFHDGBRecordMapper class.

## Create the SingleLine Record Type and Record Bean

The CICS application base state is just a blank screen. The SingleLineRecord class is needed to interact with the blank screen.

1. Right-click on paa.mysamples.cics.menu, and select **Add** → **Class...**
2. In this dialog:
  - a. Type the Class Name: SingleLineRecordType
  - b. Click the **Browse** button to set the Superclass to FixedLengthTerminalRecordType in package com.ibm.ivj.eab.record.terminal as described in the Select class instructions.
  - c. Click the **Finish** button. This creates the SingleLineRecordType class.
3. Right-click on the SingleLineRecordType class, and and select Tools → Records → Edit Record Type.... This opens the Java Record Editor.

4. Right-click on the SingleLineRecordType record, and and select Create New Field As Child.... This opens the Create a Field wizard.
  - a. Select Simple and click the **Next** → button.
  - b. Type the Field Name: Value\_attByte
  - c. Select Field Type: com.ibm.ivj.eab.record.terminal.FixedLengthTerminalAttributeType
  - d. Click the **Finish** button. This creates the Value\_attByte field
5. Change the Read Only property of the Value\_attByte field to True by clicking on its value.
6. Right-click on the **Value\_attByte** field, and select Create New Field As Sibling.... This opens the Create a Field wizard.
  - a. Select Simple and click the **Next** button.
  - b. Type the Field Name: Value
  - c. Select Field Type: com.ibm.ivj.eab.record.terminal.FixedLengthTerminalFieldType
  - d. Click the **Finish** button. This creates the **Value** field.
7. Click on the **Value** column of the Type Size property of the Value. Type the number 10 to change the Type Size.
8. Click the **Done** button. If you want to save your changes, click **yes**. This closes the Java Record Editor.
9. Right-click on SingleLineRecordType, and select Tools → Records → Generate Records....
10. Type the Class Name: SingleLineRecord
11. Leave the other fields as default, and click the **Finish** button. This creates the classes SingleLineRecord and SingleLineRecordBeanInfo.

## Verifying the Project Contents

At this point in the sample you have created all the beans used to describe the data for interacting with the CICS application. Verify that CBSamples project in the VAJ workspace looks like the following figure.

```

■ CBSamples
  ■ paa.mysamples.cics.menu
    □ DFHDGARRecord
    □ DFHDGARRecordBeanInfo
    □ DFHDGARRecordType
    □ DFHDGBRecord
    □ DFHDGBRecordBeanInfo
    □ DFHDGBRecordMapper
    □ DFHDGBRecordType
    □ MenuCustomer
    □ MenuCustomerBeanInfo
    □ MenuCustomerKey
    □ MenuCustomerKeyBeanInfo
    □ SingleLineRecord
    □ SingleLineRecordBeanInfo
    □ SingleLineRecordType
  
```

## Creating Command Beans

In this section you will create seven **Commands** used to interact with the CICS application.

Commands dictate the data that gets passed to and from the backend system in a single interaction. Commands use Record beans to define the layout of the input and output data.

After you create the commands, you will create **Navigators** that encapsulate sequences of Commands to perform functions for the PAO.

The following table summarizes all the Navigators and Commands used by each of the PAO methods.

| PAO Method Name | Navigator Used     | Commands Used                                                                        |
|-----------------|--------------------|--------------------------------------------------------------------------------------|
| retrieve        | NavigatorRetrieve  | CmdBaseToMenu<br>CmdMenuToListing<br>CmdListingToMenu<br>CmdMenuToBase               |
| insert          | NavigatorAddUpdate | CmdBaseToMenu<br>CmdMenuToListingAddUpdt<br>CmdListingToMenuAddUpdt<br>CmdMenuToBase |
| update          |                    |                                                                                      |
| del             | NavigatorDelDebit  | CmdBaseToMenu<br>CmdMenuToMenuDelDebit<br>CmdMenuToBase                              |
| debit           |                    |                                                                                      |

### Creating the CmdBaseToMenu Command

1. Right-click on `paa.mysamples.cics.menu`, and select **Add** → **Class...**
2. Type the Class Name: `CmdBaseToMenu`
3. Click the **Browse** button to set the **Superclass** to `CommunicationCommand` in package `com.ibm.ivj.eab.command` as described in the **Select class** instructions.
4. Click the **Finish** button. This creates the **CmdBaseToMenu** class.
5. Right-click on **CmdBaseToMenu**, and select **Tools** → **Command Editor...** This opens the Command Editor window which looks like the following:



6. Right-click on **Communication**, select **Add InteractionSpec**, and select the class `HODInteractionSpec` in package `com.ibm.connector.hod` as described in the **Select class** instructions. This creates a bean called **celInteractionSpec**.
7. Right-click on **celInteractionSpec**, and select **Properties**. This opens the **Properties** window.
  - a. Click on the **name** property and type `#ENTER`
  - b. Click the **OK** button to close the properties window.

8. Right-click on **Input**, select **Add IByteBuffer Bean**, and select the class `SingleLineRecord` in package `paa.mysamples.cics.menu` as described in the **Select class** instructions. This creates an input record bean called `celInput`.
9. Right-click on **celInput**, and select **Properties**. This opens the **Properties** window.
  - a. Scroll down to the **Value** property, click on it, and type `MENU`.
  - b. Click the **OK** button to close the properties window.
10. Right-click on **Output**, select **Add IByteBuffer Bean**, and select the class `DFHDGARRecord` in package `paa.mysamples.cics.menu` described in the **Select class** instructions. This creates an output record bean called `ceOutput1`.
11. Click the **OK** button to close the Command Editor.

## Command Bean Summary

The following table summarizes the properties of all the Command Beans used in this sample. You created the `CmdBaseToMenu` in the previous section.

| <i>Table 12. Command Bean summary (CICS-HOD)</i> |                   |                                                                                     |
|--------------------------------------------------|-------------------|-------------------------------------------------------------------------------------|
| <b>Command Name</b>                              | <b>Properties</b> | <b>Values</b>                                                                       |
| CmdBaseToMenu                                    | ceInteractionSpec | Class: HODInteractionSpec<br>Name: #ENTER                                           |
|                                                  | ceInput           | Class:SingleLineRecord<br>Value: MENU                                               |
|                                                  | ceOutput1         | Class: DFHDGARecord                                                                 |
| CmdMenuToBase                                    | ceInteractionSpec | Class:HODInteractionSpec<br>Name: #CLEAR                                            |
|                                                  | [ceInput]         | Class:DFHDGARecord                                                                  |
|                                                  | ceOutput1         | Class: SingleLineRecord                                                             |
| CmdMenuToListing                                 | ceInteractionSpec | Class:HODInteractionSpec#<br>Name: #ENTER                                           |
|                                                  | [ceInput]         | Class: DFHDGARecordProperty<br>Features: _DFHDGA11, ceInputKEY                      |
|                                                  | ceOutput1         | Class: DFHDGBRecord<br>Add Mapper: DFHDGBRecordMapper                               |
|                                                  | ceOutput2         | Class: DFHDGARecordPromoted<br>Feature: MSG                                         |
| CmdListingToMenu                                 | ceInteractionSpec | Class: HODInteractionSpec<br>Name: #ENTER                                           |
|                                                  | [ceInput]         | Class: DFHDGBRecord                                                                 |
|                                                  | ceOutput1         | Class: DFHDGARecord                                                                 |
| CmdMenuToListingAddUpdt                          | ceInteractionSpec | Class: HODInteractionSpec<br>Name: #ENTER                                           |
|                                                  | [ceInput]         | Class: DFHDGARecord<br>Property Features: ceInput_DFHDGA11,<br>ceInputKEY           |
|                                                  | ceOutput1         | Class: DFHDGBRecord                                                                 |
|                                                  | ceOutput2         | Class: DFHDGARecord<br>Promoted Feature: MSG                                        |
| CmdListingToMenuAddUpdt                          | ceInteractionSpec | Class: HODInteractionSpec<br>Name: #ENTER                                           |
|                                                  | [ceInput]         | Class: DFHDGBRecord<br>Add Mapper: DFHDGBRecordMapper                               |
|                                                  | ceOutput1         | Class: DFHDGBRecord<br>Promoted Feature: MSG                                        |
| CmdMenuToMenuDelDebit                            | ceInteractionSpec | Class: HODInteractionSpec<br>Name: #ENTER                                           |
|                                                  | [ceInput]         | Class: DFHDGARecord<br>Property Features: ceInput_DFHDGA11,<br>ceInputKEY, ceAMOUNT |
|                                                  | ceOutput1         | Class: DFHDGARecord<br>Promoted Feature: MSG                                        |

**Note:** The square brackets around ceInput indicate that it is a Variable IByteBuffer Bean. You can use the Command Editor to promote features. However, for Variable IByteBuffer Beans, such as

[celInput], you must use the Visual Composition Editor to add each property feature and connect it to the corresponding record property.

## Creating the CmdMenuToBase Command

1. Right-click on the **paa.mysamples.cics.menu**, and select **Add → Class....**
2. Type the Class Name: **CmdMenuToBase**
3. Click the **Browse** button to set the **Superclass** to **CommunicationCommand** in package **com.ibm.ivj.eab.command** as described in the **Select class** instructions. Click **Finish**. This creates the **CmdMenuToBase** class.
4. Right-click on **CmdMenuToBase**, and select **Tools → Command Editor....** This opens the Command Editor.
5. Right-click on **Communication**, select **Add InteractionSpec** and select the class **HODInteractionSpec** in package **com.ibm.connector.hod** as described in the **Select class** instructions. This creates a bean called **celInteractionSpec**.
6. Right-click on **celInteractionSpec**, and select **Properties**. This opens the **Properties** window.
  - a. Click on the **name** property and type **#CLEAR**
  - b. Click the **OK** button to close the properties window.
7. Right-click on **Input**, select **Add IByteBuffer Bean Variable**, and select the class **DFHDGARRecord** in package **paa.mysamples.cics.menu** as described in the **Select class** instructions. This creates an input record bean called **celInput**.
8. Right-click on **Output**, select **Add IByteBuffer Bean**, and select the class **SingleLineRecord** in package **paa.mysamples.cics.menu** as described in the **Select class** instructions. This creates an output record bean called **ceOutput1**.
9. Click the **OK** button to close the Command Editor.

## Creating the CmdMenuToListing Command

1. Right-click on **paa.mysamples.cics.menu**, and select **Add → Class....**
2. Type the Class Name: **CmdMenuToListing**
3. Click the **Browse** button to set the **Superclass** to **CommunicationCommand** in package **com.ibm.ivj.eab.command** as described in the **Select class** instructions. Click **Finish**. This creates the **CmdMenuToListing** class.

Follow the steps in .

### Adding Available Features

1. From the pop-up menu for the class, select **Open To → BeanInfo**.
2. In this Dialog:
  - a. Select **Features → Generate BeanInfo** class. This will generate a new **BeanInfo** class for your command class.
  - b. Select **Features → Add Available Features**.
  - c. In the **Add Available Features** dialog, select all features that may appear and then click on the **OK** button.
  - d. Close the Command Class window.

4. Right-click on **CmdMenuToListing**, and select **Tools → Command Editor....** This opens the Command Editor.

5. Right-click on **Communication**, select **Add InteractionSpec** and select the class HODInteractionSpec in package com.ibm.connector.hod as described in the **Select class** instructions. This creates a bean called **celInteractionSpec**.
6. Right-click on **celInteractionSpec**, and select **Properties**. This opens the **Properties** window.
  - a. Click on the **name** property and type #ENTER
  - b. Click the **OK** button to close the properties window.
7. Right-click on **Input**, select **Add IByteBuffer Bean Variable**, and select the class DFHDGARRecord in package paa.mysamples.cics.menu as described in the **Select class** instructions. This creates an input record bean called **ceInput**.
8. Right-click on **Output**, and select **Add IByteBuffer Bean** and select the class DFHDGBRecord in package paa.mysamples.cics.menu as described in the **Select class** instructions. This creates a bean called **ceOutput1**.
9. Right-click on **ceOutput1**, select **Add Mapper**, and select the class DFHDGBRecordMapper in package paa.mysamples.cics.menu as described in the **Select class** instructions. This creates an output record bean called **ceMapperCeOutput1**.
10. Right-click on **Output**, and select **Add IByteBuffer Bean** and select the class DFHDGARRecord in package paa.mysamples.cics.menu as described in the **Select class** instructions. This creates a bean called **ceOutput2**.
11. Right-click on **CeOutput2**, select Promote Bean Feature. This opens the Promoted Feature dialog.
  - a. Click the **Property** radio button select MSG<sup>RWB</sup> from the list of properties.
  - b. Click the >> button and then click the **OK**. This closes the Promoted features dialog.
12. Click the **OK** button to close the Command Editor.

## Adding Features to the CmdMenuToListing Command

1. Right-click on **CmdMenuToListing**, select Open To → 5 BeanInfo. This opens the **BeanInfo** tab in the CmdMenuToListing window.
2. Select the pull-down menu **Features** → **New Property Feature....**
3. Type the Property name: ceInput\_DFHDGA11 and click the **Finish** button.
4. Select the pull-down menu **Features** → **New Property Feature....**
5. Type the Property name: ceInputKEY and click the **Finish** button.
6. Verify that the list of Features contains the following list:
  - P ceInput\_DFHDGA11<sup>RWB</sup>
  - P ceInputKEY<sup>RWB</sup>
  - P ceOutput2MSG<sup>RWB</sup>
7. Click on the Visual Composition tab in the CmdMenuToListing window. You see several icons and one of them is labeled ceInput.
8. Follow the steps in “Connecting a Command Input Feature” to connect the \_DFHDGA11 feature.
9. Follow the steps in “Connecting a Command Input Feature” to connect the KEY feature.
10. Select the pull-down menu **Bean** → **Save Bean**.
11. Close the Visual Composition Editor by clicking on the **X** button in the upper-right corner of the window.

### Connecting a Command Input Feature

1. In the Visual Composition Editor, right-click on celInput and select Connect → this.
2. Click on celInput. This opens a popup menu.
3. Select **Connectable Features...** from the popup menu.
4. Click the **Method** radio button.
5. Select the feature name from the list and click the **OK** button.
6. Optional: Along the right edge of the celInput icon are three black dots which are the control points for the dashed green arrow loop. Use the mouse to drag the middle black dot slightly to the right. This will enlarge the loop so that it does not overlap the celInput icon.
7. Now you can see a dashed green arrow that leaves the celInput icon and then loops back to it.
8. Right-click on the dashed green arrow and select Connect → value from the popup menu.
9. Click on the background of the **Visual Composition** window. This opens a popup menu.
10. Select **Connectable Features...** from the popup menu.
11. Select the celInput property that corresponds to the feature name from the list. For example celInputKEY corresponds to the KEY feature. Click the **OK** button.
12. Verify the arrow is now solid green, and it is connected to the edge of the window with a solid magenta line.

## Creating the CmdListingToMenu Command

1. Right-click on paa.mysamples.cics.menu, and select **Add** → **Class...**
2. Type the Class Name: CmdListingToMenu
3. Click the **Browse** button to set the Superclass to CommunicationCommand in package com.ibm.ivj.eab.command as described in the Select class instructions.
4. Click the **Finish** button. This creates the CmdListingToMenu class.
5. Follow the steps in .
6. Right-click on CmdListingToMenu, and select **Tools** → **Command Editor...** This opens the Command Editor.
7. Right-click on Communication, select Add InteractionSpec and select the class HODInteractionSpec in package com.ibm.connector.hod as described in the Select class instructions. This creates a bean called celInteractionSpec.
8. Right-click on celInteractionSpec, and select Properties. This opens the **Properties** window.
  - a. Click on the **name property** and type #ENTER
  - b. Click the **OK** button to close the properties window.
9. Right-click on Input, select Add IByteBuffer Bean Variable, and select the class DFHDGARRecord in package paa.mysamples.cics.menu as described in the Select class instructions. This creates an input record bean called celInput.
10. Right-click on Output, select Add IByteBuffer Bean, and select the class DFHDGARRecord in package paa.mysamples.cics.menu as described in the Select class instructions. This creates an output record bean called ceOutput1.
11. Click the **OK** button to close the Command Editor.

## Creating the CmdMenuToListingAddUpdt Command

1. Right-click on paa.mysamples.cics.menu, and select **Add** → **Class....**
2. Type the Class Name: CmdMenuToListingAddUpdt.
3. Click the **Browse** button to set the Superclass to CommunicationCommand in package com.ibm.ivj.eab.command as described in the Select class instructions. This creates the CmdMenuToListingAddUpdt class.  
Follow the steps in .
4. Right-click on CmdMenuToListingAddUpdt, and select Tools → Command Editor.... This opens the Command Editor.
5. Right-click on Communication, select Add InteractionSpec and select the class HODInteractionSpec in package com.ibm.connector.hod as described in the Select class instructions. This creates a bean called ceInteractionSpec.
6. Right-click on ceInteractionSpec, and select Properties. This opens the Properties window.
  - a. Click on the **name property** and type #ENTER.
  - b. Click the **OK** button to close the properties window.
7. Right-click on Input, select Add IByteBuffer Bean Variable, and select the class DFHDGBRecord in package paa.mysamples.cics.menu as described in the Select class instructions. This creates an input record bean called ceInput.
8. Right-click on Output, select Add IByteBuffer Bean, and select the class DFHDGBRecord in package paa.mysamples.cics.menu as described in the Select class instructions. This creates an output record bean called ceOutput1.
9. Right-click on Output, select Add IByteBuffer Bean, and select the class DFHDGARRecord in package paa.mysamples.cics.menu as described in the Select class instructions. This creates an output record bean called ceOutput2.
10. Right-click on ceOutput2, select Promote Bean Feature. This opens the Promoted features dialog.
  - a. Click the **Property** radio button, and select MSG<sup>RWB</sup> from the list of properties.
  - b. Click the >> button, and click the **OK** button. This closes the Promoted features dialog.
11. Click the **OK** button to close the Command Editor.

## Adding Features to the CmdMenuToListingAddUpdt Command

1. Right-click on CmdMenuToListingAddUpdt, select **Open To** → **5 BeanInfo**. This opens the BeanInfo tab in the **CmdMenuToListingAddUpdt** window.
2. Select the pull-down menu **Features** → **New Property Feature....**
3. Type the Property name: ceInput\_DFHDGA11 and click the **Finish** button.
4. Select the pull-down menu **Features** → **New Property Feature....**
5. Type the Property name: ceInputKEY and click the **Finish** button.
6. Verify that the list of Features contains the following list.
  - P ceInput\_DFHDGA11<sup>RWB</sup>
  - P ceInputKEY<sup>RWB</sup>
  - P ceOutput2MSG<sup>RWB</sup>
7. Click on the **Visual Composition** tab in the CmdMenuToListingAddUpdt window. You see several icons and one of them is labeled ceInput.
8. Follow the steps in Connecting a Command Input Feature to connect the \_DFHDGA11 feature.

9. Follow the steps in Connect a Command Input Feature to connect the KEY feature.
10. Select the pull-down menu **Bean** → **Save Bean**.
11. Close the Visual Composition Editor by clicking on the **X** button in the upper-right corner of the window.

## Creating the CmdListingToMenuAddUpdt Command

1. Right-click on paa.mysamples.cics.menu, and select **Add** → **Class....**
2. Type the Class Name: CmdListingToMenuAddUpdt
3. Click the **Browse** button to set the Superclass to CommunicationCommand in package com.ibm.ivj.eab.command as described in the Select class instructions. This creates the CmdListingToMenuAddUpdt class.

Follow the steps in .

4. Right-click on CmdListingToMenuAddUpdt, and select **Tools** → **Command Editor....** This opens the Command Editor.
5. Right-click on Communication, select Add InteractionSpec and select the class HODInteractionSpec in package com.ibm.connector.hod. as described in the Select class instructions. This creates a bean called ceInteractionSpec.
6. Right-click on ceInteractionSpec, and select Properties. This opens the Properties window.
  - a. Click on the **name property** and type #ENTER
  - b. Click the **OK** button to close the properties window.
7. Right-click on Input, select Add IByteBuffer Bean Variable, and select the class DFHDGBRecord in package paa.mysamples.cics.menu as described in the Select class instructions. This creates an input record bean called ceInput.
8. Right-click on ceInput, select Add Mapper, and select the class DFHDGBRecordMapper in package paa.mysamples.cics.menu as described in the Select class instructions. This creates a mapper bean called ceMapperCeInput.
9. Right-click on Output, select Add IByteBuffer Bean, and select the class DFHDGAREcord in package paa.mysamples.cics.menu as described in the Select class instructions. This creates an output record bean called ceOutput1.
10. Right-click on ceOutput1, select Promote Bean Feature. This opens the Promoted features dialog.
  - a. Click the **Property** radio button, and select MSG<sup>RWB</sup> from the list of properties.
  - b. Click the >> button, and click the **OK** button. This closes the Promoted features dialog.
11. Click the **OK** button to close the Command Editor.

## Creating the CmdMenuToMenuDelDebit Command

1. Right-click on paa.mysamples.cics.menu, and select **Add** → **Class....**
2. Type the Class Name: CmdMenuToMenuDelDebit
3. Click the **Browse** button to set the Superclass to CommunicationCommand in package com.ibm.ivj.eab.command as described in the Select class instructions. This creates the CmdMenuToMenuDelDebit class.

Follow the steps in .

4. Right-click on CmdMenuToMenuDelDebit, and select **Tools** → **Command Editor....** This opens the Command Editor.

5. Right-click on Communication, select Add InteractionSpec and select the class HODInteractionSpec in package com.ibm.connector.hod. as described in the Select class instructions. This creates a bean called ceInteractionSpec.
6. Right-click on ceInteractionSpec, and select Properties. This opens the Properties window.
  - a. Click on the **name property** and type #ENTER
  - b. Click the **OK** button to close the properties window.
7. Right-click on Input, select Add IByteBuffer Bean Variable, and select the class DFHDGARRecord in package paa.mysamples.cics.menu as described in the Select class instructions. This creates an input record bean called ceInput.
8. Right-click on Output, select Add IByteBuffer Bean, and select the class DFHDGARRecord in package paa.mysamples.cics.menu as described in the Select class instructions. This creates an output record bean called ceOutput1.
9. Right-click on **ceOutput1**, select Promote Bean Feature. This opens the Promoted features dialog.
  - a. Click the **Property** radio button, and select MSG<sup>RWB</sup> from the list of properties.
  - b. Click the >> button, and click the **OK** button. This closes the Promoted features dialog.
10. Click the **OK** button to close the Command Editor.

## Adding Features to the CmdMenuToMenuDelDebit Command

1. Right-click on **CmdMenuToMenuDelDebit**, select Open To → 5 BeanInfo. This opens the BeanInfo tab in the **CmdMenuToMenuDelDebit**.
2. Select the pull-down menu **Features** → **New Property Feature....**
3. Type the Property name: ceInput\_DFHDGA11 and click the **Finish** button.
4. Select the pull-down menu **Features** → **New Property Feature....**
5. Type the Property name: ceInputKEY and click the **Finish** button.
6. Select the pull-down menu **Features** → **New Property Feature....**
7. Type the Property name: ceInputAMOUNT and click the **Finish** button.
8. Verify that the list of Features contains the following list.
  - P ceInput\_DFHDGA11<sup>RWB</sup>
  - P ceInputAMOUNT<sup>RWB</sup>
  - P ceInputKEY<sup>RWB</sup>
  - P ceOutput1MSG<sup>RWB</sup>
9. Click on the **Visual Composition** tab in the CmdMenuToMenuDelDebit window. You see three icons labeled ceInteractionSpec, ceInput, and ceOutput1.
10. Follow the steps in “Connecting a Command Input Feature” to connect the \_DFHDGA11 feature.
11. Follow the steps in “Connecting a Command Input Feature” to connect the KEY feature.
12. Follow the steps in “Connecting a Command Input Feature” to connect the AMOUNT feature.
13. Select the pull-down menu **Bean** → **Save Bean**.
14. Close the Visual Composition Editor by clicking on the **X** button in the upper-right corner of the window.

This completes the creation of the Command beans. In the next section, you will create the Navigator beans that will use the Command beans.

## Creating Navigator Beans

In this section, you will create three Navigators used to interact with the CICS application.

### Creating the NavigatorRetrieve Navigator

1. Right-click on paa.mysamples.cics.menu, and select **Add** → **Class....**
2. Type the Class Name: NavigatorRetrieve
3. Click the **Browse** button to set the Superclass to CommunicationNavigator in package com.ibm.ivj.eab.command as described in the Select class instructions. Select the **Compose The Class Visually** check box.
4. Click the **Finish** button. This creates the NavigatorRetrieve class and opens it in the Visual Composition editor.
5. Click the **BeanInfo** tab. Follow the steps in the , then click the **Visual Composition** tab.
6. Follow the to add a new bean with the following:
  - Class = com.ibm.connector.hod.HODConnectionSpec
  - Name = connSpec

| Adding a Bean in the Visual Composition Editor                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none"><li>1. Click the Choose Bean... icon, (in the top-right corner of the tool palette). In this dialog:</li><li>2. Set the Bean Type to <b>Class</b>.</li><li>3. Click the <b>Browse</b> button and select the correct class. (See the Select class instructions.)</li><li>4. Type the appropriate name in the <b>Name:</b> field.</li><li>5. Click the <b>OK</b> button.</li><li>6. Drop the bean onto the Visual Composition Editor canvas by clicking somewhere on the canvas.</li></ol> |



7. Right-click on connSpec and select Properties. This opens the Properties Window.
  - a. Change the debugScreenEnabled property to True.
  - b. Change the hostname property to trutycics.austin.ibm.com
  - c. Change the portNumber property to 5555.
  - d. Close the properties window.

**Note:** The connection specification here is only useful in the unit test environment with VAJ. To run from the Component Broker environment, the connection spec is set in each of the CRUD methods as described in a later section.
8. Right-click on the connSpec and select **Connect** → this.
  - a. Click on the window background. This opens the End connection dialog.
  - b. Select connectionSpec<sup>RWB</sup> and click the **OK** button.
9. Follow the to add a new bean for each of the Class and Name combinations are in the following table:

| <i>Table 13. Class and Name combinations in the NavigatorRetrieve (CICS-HOD)</i> |                   |
|----------------------------------------------------------------------------------|-------------------|
| <b>Class</b>                                                                     | <b>Name</b>       |
| paa.mysamples.cics.menu.CmdBaseToMenu                                            | BaseToMenu        |
| paa.mysamples.cics.menu.CmdMenuToMenuListing                                     | MenuToMenuListing |
| paa.mysamples.cics.menu.CmdListingToMenu                                         | ListingToMenu     |
| paa.mysamples.cics.menu.CmdMenuToBase                                            | MenuToBase        |

10. Follow the listed below:

| <i>Table 14. Adding a Connection Procedure (CICS-HOD)</i>                                                                                                               |  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <b>Adding a Connection in the Visual Composition Editor:</b>                                                                                                            |  |
| 1. Right-click on the Source bean (or background) and select Connect → Connectable Features...(Connect...). This opens the Start connection from dialog.                |  |
| 2. Click the <b>Event</b> radio button.                                                                                                                                 |  |
| 3. Select the <b>Source Event</b> from the list, and click the <b>OK</b> button.                                                                                        |  |
| 4. Click on the Target bean (or background) and select Connectable Features.... This opens the End connection to dialog.                                                |  |
| 5. Select the <b>Target Event</b> from the list, and click the <b>OK</b> button. This creates a connection and draws a dashed green line between the source and target. |  |
| 6. Right-click on the dashed green line, and select <b>Properties</b> . This opens the Event-to-method connection dialog.                                               |  |
| 7. Select the Pass event data check box, and click the <b>OK</b> button. This changes the line to a solid green line.                                                   |  |

11. Follow the to create the connections listed in the following table:

| <i>Table 15. NavigatorRetrieve Navigator connections (CICS-HOD)</i> |                                                                              |
|---------------------------------------------------------------------|------------------------------------------------------------------------------|
| <b>Source/Target</b>                                                | <b>Event</b>                                                                 |
| Source: background<br>Target: BaseToMenu                            | internalExecutionStarting(CommandEvent)<br>execute(CommandEvent)             |
| Source: BaseToMenu<br>Target: MenuToListing                         | executionSuccessful(CommandEvent)<br>execute(CommandEvent)                   |
| Source: MenuToListing<br>Target: ListingToMenu                      | executionSuccessful(CommandEvent)<br>execute(CommandEvent)                   |
| Source: ListingToMenu<br>Target: MenuToBase                         | executionSuccessful(CommandEvent)<br>execute(CommandEvent)                   |
| Source: MenuToBase<br>Target: background                            | executionSuccessful(CommandEvent)<br>returnExecutionSuccessful(CommandEvent) |

12. Right-click on the MenuToListing bean, and select Promote Bean Feature...

13. Click the **Property** radio button.

14. Select ceInput\_DFHDGA11<sup>RWB</sup> from the list, and click the >> button. Similarly, move ceInputKEY<sup>RWB</sup> and ceOutput2MSG<sup>RWB</sup> to the Promoted features list.

15. Click the **OK** button to close the Promoted features dialog.

16. Select the pull-down menu **Bean** → **Save Bean**.

17. Close the Visual Composition Editor by clicking on the **X** button in the upper-right corner of the window.
18. Perform the that follows.

**Changing the HandleException Method**

1. Expand the class by clicking the **+** button next to it.
2. Select the handleException method. The source code for the method appears in the Source pane in the lower half of the window.
3. Change the implementation to the following:
 

```
/**
 * called whenever the part throws an exception.
 * @param exception java.lang.Throwable
 */
private void handleException(Throwable exception) {

    /* Uncomment the following lines to print uncaught exceptions to stdout */
    System.out.println("----- UNCAUGHT EXCEPTION -----");
    exception.printStackTrace(System.out);

    this.internalExceptionHandler(exception);
}

```
4. Select the menu **Edit** → **Save**.

## Creating the NavigatorAddUpdate Navigator

1. Right-click on paa.mysamples.cics.menu, and select **Add** → **Class....**
2. Type the Class Name: NavigatorAddUpdate
3. Click the **Browse** button to set the Superclass to CommunicationNavigator in package com.ibm.ivj.eab.command as described in the Select class instructions. Select the **Compose The Class Visually** check box.
4. Click the **Finish** button.
5. This creates the NavigatorAddUpdate class and opens it in the Visual Composition editor.
6. Click the **BeanInfo** tab. Follow the steps in the , then click the **Visual Composition** tab.
7. Follow the to add a new bean with
  - Class = com.ibm.connector.hod.HODConnectionSpec
  - Name = connSpec
8. Right-click on connSpec and select Properties. This opens the Properties Window.
  - a. Change the debugScreenEnabled property to True
  - b. Change the hostname property to trutycics.austin.ibm.com
  - c. Change the portNumber property to 5555
  - d. Close the properties window
9. Right-click on the connSpec and select Connect → this.
  - a. Click on the window background. This opens the End connection dialog.
  - b. Select connectionSpec<sup>RWB</sup> and click the **OK** button.
10. Follow the to add a new bean for each of the Class and Name combinations in the following table:

| <i>Table 16. Class and Name combinations in NavigatorAddUpdate Navigator (CICS-HOD)</i> |                      |
|-----------------------------------------------------------------------------------------|----------------------|
| <b>Class</b>                                                                            | <b>Name</b>          |
| paa.mysamples.cics.menu.CmdBaseToMenu                                                   | BaseToMenu           |
| paa.mysamples.cics.menu.CmdMenuToListingAddUpdt                                         | MenuToListingAddUpdt |
| paa.mysamples.cics.menu.CmdListingToMenuAddUpdt                                         | ListingToMenuAddUpdt |
| paa.mysamples.cics.menu.CmdMenuToBase                                                   | MenuToBase           |

11. Follow the to create the connections. Class and Name combinations are listed in the following table:

| <i>Table 17. Class and Name combinations in NavigatorAddUpdate Navigator (CICS-HOD)</i> |                                                                              |
|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| <b>Source/Target</b>                                                                    | <b>Event</b>                                                                 |
| Source: background<br>Target: BaseToMenu                                                | internalExecutionStarting(CommandEvent)<br>execute(CommandEvent)             |
| Source: BaseToMenu<br>Target: MenuToListingAddUpdt                                      | executionSuccessful(CommandEvent)<br>execute(CommandEvent)                   |
| Source: MenuToListingAddUpdt<br>Target: ListingToMenuAddUpdt                            | executionSuccessful(CommandEvent)<br>execute(CommandEvent)                   |
| Source: ListingToMenuAddUpdt<br>Target: MenuToBase                                      | executionSuccessful(CommandEvent)<br>execute(CommandEvent)                   |
| Source: MenuToBase<br>Target: background                                                | executionSuccessful(CommandEvent)<br>returnExecutionSuccessful(CommandEvent) |

12. Right-click on the MenuToListingAddUpdt bean, and select Promote Bean Feature....

13. Click the **Property** radio button.

14. Select ceInput\_DFHDGA11<sup>RWB</sup> from the list, and click the >> button. Similarly, move ceInputKEY<sup>RWB</sup> and ceOutput2MSG<sup>RWB</sup> to the Promoted features list.

15. Click the **OK** button to close the Promoted features dialog.

16. Right-click on the ListingToMenuAddUpdt bean, and select Promote Bean Feature....

17. Click the **Property** radio button.

18. Select ceOutput1MSG<sup>RWB</sup> from the list, and click the >> button.

19. Click the **OK** button to close the Promoted features dialog.

20. Select the pull-down menu **Bean** → **Save Bean**.

21. Close the Visual Composition Editor by clicking on the **X** button in the upper-right corner of the window.

22. Perform the .

## Creating the NavigatorDelDebit Navigator

1. Right-click on paa.mysamples.cics.menu, and select **Add** → **Class....**

2. Type the Class Name: NavigatorDelDebit

3. Click the **Browse** button to set the Superclass to CommunicationNavigator in package com.ibm.ivj.eab.command as described in the Select class instructions. Select the **Compose The Class Visually** check box.

4. Click the **Finish** button. This creates the NavigatorDelDebit class and opens it in the Visual Composition editor
5. Click the **BeanInfo** tab. Complete the steps in and then click the **Visual Composition** tab.
6. Follow the to add a new bean with the following:
  - Class = com.ibm.connector.hod.HODConnectionSpec
  - Name = connSpec
7. Right-click on connSpec and select Properties. This opens the Properties Window.
  - a. Change the debugScreenEnabled property to True
  - b. Change the hostname property to trutycics.austin.ibm.com
  - c. Change the portNumber property to 5555
  - d. Close the properties window.
8. Right-click on the connSpec and select **Connect** → this.
  - a. Click on the window background. This opens the End connection dialog.
  - b. Select connectionSpec<sup>RWB</sup> and click the **OK** button.
9. Follow the to add a new bean for each of the Class and Name combinations in the following table:

*Table 18. Class and Name combinations in NavigatorDelDebit Navigator*

| Class                                         | Name               |
|-----------------------------------------------|--------------------|
| paa.mysamples.cics.menu.CmdBaseToMenu         | BaseToMenu         |
| paa.mysamples.cics.menu.CmdMenuToMenuDelDebit | MenuToMenuDelDebit |
| paa.mysamples.cics.menu.CmdMenuToBase         | MenuToBase         |

10. Follow the to create the connections listed in the following table:

*Table 19. NavigatorDelDebit Navigator connections*

| Source/Target                                    | Event                                                                        |
|--------------------------------------------------|------------------------------------------------------------------------------|
| Source: background<br>Target: BaseToMenu         | internalExecutionStarting(CommandEvent)<br>execute(CommandEvent)             |
| Source: BaseToMenu<br>Target: MenuToMenuDelDebit | executionSuccessful(CommandEvent)<br>execute(CommandEvent)                   |
| Source: MenuToMenuDelDebit<br>Target: MenuToBase | executionSuccessful(CommandEvent)<br>execute(CommandEvent)                   |
| Source: MenuToBase<br>Target: background         | executionSuccessful(CommandEvent)<br>returnExecutionSuccessful(CommandEvent) |

11. Right-click on the MenuToMenuDelDebit bean, and select Promote Bean Feature....
12. Click the **Property** radio button.
13. Select celInput\_DFHDGA11<sup>RWB</sup> from the list, and click the >> button. Similarly, move celInputKEY<sup>RWB</sup>, celInputAMOUNT<sup>RWB</sup>, and ceOutput1MSG<sup>RWB</sup> to the Promoted features list.
14. Click the **OK** button to close the Promoted features dialog.
15. Select the pull-down menu **Bean** → **Save Bean**.
16. Close the Visual Composition Editor by clicking on the **X** button in the upper-right corner of the window.

## Using the Navigators

To use the navigators, you need to add code to the PAO methods. The PAO methods that use the Navigators are the CRUD methods and the push-down methods. The following instructions guide you to add code to the MenuCustomer PAO methods.

### Editing the MenuCustomer::debit method

1. In the Workbench, select the MenuCustomer method debit(int). The source code for the debit method appears in the Source pane in the lower half of the window.
2. Change the implementation to the following.

```
/**
 * Perform the debit method.
 * @param amount int
 */
public void debit(int amount) {
    String amnt = Integer.toString(amount);
    NavigatorDelDebit navigator = new NavigatorDelDebit();

    navigator.setConnectionSpec(this.getConnectionSpec());
    navigator.setMenuToMenuDelDebitCeInput_DFHDGA11("DEBT");
    navigator.setMenuToMenuDelDebitCeInputKEY(this.getNumber());
    navigator.setMenuToMenuDelDebitCeInputAMOUNT(amnt);
    navigator.execute();
    String message = navigator.getMenuToMenuDelDebitCeOutput1MSG();
    if (!message.equals("ACCOUNT DEBITED"))
        throw new RuntimeException("Unexpected message in debit: '"+message+"'");
}
```

3. Select the pull-down menu **Edit** → **Save**.

### Editing the MenuCustomer::del method

1. In the Workbench, select the MenuCustomer method del(). The source code for the del method appears in the Source pane in the lower half of the window.
2. Change the implementation to the following.

```
/**
 * This method was created in VisualAge.
 * @exception com.ibm.ipaa.IDataKeyNotFoundException The exception description.
 */
public void del() throws com.ibm.ipaa.IDataKeyNotFoundException {
    NavigatorDelDebit navigator = new NavigatorDelDebit();

    navigator.setConnectionSpec(this.getConnectionSpec());
    navigator.setMenuToMenuDelDebitCeInput_DFHDGA11("DELE");
    navigator.setMenuToMenuDelDebitCeInputKEY(this.getNumber());
    navigator.execute();

    String message = navigator.getMenuToMenuDelDebitCeOutput1MSG();

    if (!message.equals("RECORD DELETED"))
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}
```

3. Select the pull-down menu **Edit** → **Save**.

## Editing the MenuCustomer::insert method

1. In the Workbench, select the MenuCustomer method insert(). The source code for the insert method appears in the Source pane in the lower half of the window.
2. Change the implementation to the following.

```
/**
 * This method was created in VisualAge.
 * @exception com.ibm.ipaa.IDataKeyAlreadyExistsException The exception description.
 */
public void insert() throws com.ibm.ipaa.IDataKeyAlreadyExistsException {
    NavigatorAddUpdate navigator = new NavigatorAddUpdate();

    navigator.setConnectionSpec(this.getConnectionSpec());
    navigator.setMenuToListingAddUpdtCeInput_DFHGGA11("ADDS");
    navigator.setMenuToListingAddUpdtCeInputKEY(this.getNumber());
    navigator.execute();

    String message = navigator.getListingToMenuAddUpdtCeOutput1MSG();

    if (message.equals("DUPLICATE RECORD"))
        throw new com.ibm.ipaa.IDataKeyAlreadyExistsException();

    if (!message.equals("RECORD ADDED"))
        throw new RuntimeException("Unexpected message in insert: '"+message+"'");
}
```

3. Select the pull-down menu **Edit** → **Save**.

## Editing the MenuCustomer::retrieve method

1. In the Workbench, select the MenuCustomer method retrieve(). The source code for the retrieve method appears in the Source pane in the lower half of the window.
2. Change the implementation to the following.

```
/**
 * This method was created in VisualAge.
 * @exception com.ibm.ipaa.IDataKeyNotFoundException The exception description.
 */
public void retrieve() throws com.ibm.ipaa.IDataKeyNotFoundException {
    NavigatorRetrieve navigator = new NavigatorRetrieve();

    navigator.setConnectionSpec(this.getConnectionSpec());
    navigator.setMenuToListingCeInput_DFHGGA11("INQY");
    navigator.setMenuToListingCeInputKEY(this.getNumber());
    navigator.execute();

    String message = navigator.getMenuToListingCeOutput2MSG();

    if (message.equals("INVALID NUMBER - PLEASE REENTER"))
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}
```

3. Select the pull-down menu **Edit** → **Save**.

## Editing the MenuCustomer::update method

1. In the Workbench, select the MenuCustomer method update(). The source code for the update method appears in the Source pane in the lower half of the window.
2. Change the implementation to the following:

```
/**
 * This method was created in VisualAge.
 * @exception com.ibm.ipaa.IDataKeyAlreadyExistsException The exception description.
 */
public void update() throws com.ibm.ipaa.IDataKeyNotFoundException {
    NavigatorAddUpdate navigator = new NavigatorAddUpdate();

    navigator.setConnectionSpec(this.getConnectionSpec());
    navigator.setMenuToListingAddUpdtCeInput_DFHDGA11("UPDT");
    navigator.setMenuToListingAddUpdtCeInputKEY(this.getNumber());
    navigator.execute();

    String message;
    message = navigator.getMenuToListingAddUpdtCeOutput2MSG();
    if (message.equals("INVALID NUMBER - PLEASE REENTER"))
        throw new com.ibm.ipaa.IDataKeyNotFoundException();

    message = navigator.getListingToMenuAddUpdtCeOutput1MSG();

    if (!message.equals("RECORD UPDATED"))
        throw new RuntimeException("Unexpected message in update: '"+message+"'");
}
```

3. Select the pull-down menu Edit → Save.

## Unit Testing the EAB Object

You are now ready to unit test the object you built using EAB. It is called a unit test because it does not involve the entire Component Broker sample application, but only the portion from the procedural adapter object downward to the CICS/IMS server.

The unit test is simply a class called Execute with a main method. The main method creates a MenuCustomer and invokes the CRUD and push-down methods on it.

To create the Unit Test Execute Class:

1. Right-click on paa.mysamples.cics.menu, and select **Add** → **Class....**
2. In this dialog:
  - a. Type the Class Name: Execute
  - b. Leave the default Superclass set to java.lang.Object, and click the **Finish** button.
  - c. Deselect the **Compose the Class Visually** and **Browse** check boxes.

To create the Unit Test main Method:

1. Right-click the Execute class and select **Add** → **Method**. This opens the Create Method wizard.
2. Click the **Create a new main method** radio button.
3. Click the **Finish** button. This creates the main method.
4. Select the main method in the Execute class so that it is displayed in the Source pane at the bottom of the Workspace window.

5. Change the implementation to the following.

```
/**
 * This method was created in VisualAge.
 * @param args java.lang.String[]
 */

public static void main(String args[]) {

    // DECLARE LOCAL VARIABLES

    String keynum1 = "001671";
    MenuCustomerKey key;
    MenuCustomer customer;

    // CREATE THE RUNTIME CONTEXT

    com.ibm.connector.infrastructure.RuntimeContext rtc =
        new com.ibm.connector.infrastructure.RuntimeContext();
    com.ibm.ivj.trace.SARASService ras =
        new com.ibm.ivj.trace.SARASService();
    ras.setTraceLevel(0);
    rtc.setRASService(ras);
    com.ibm.connector.infrastructure.RuntimeContext.setCurrent(rtc);

    // CREATE THE EAB CACHE SPACE

    com.ibm.ivj.eab.businessobject.InstanceSpaceHolder.GlobalInstanceSpace =
        com.ibm.ivj.eab.paa.WSIDBasedInstanceSpace.getInstance();

    // CREATE THE CONNECTION SPEC

    com.ibm.connector.hod.HODConnectionSpec cs =
        new com.ibm.connector.hod.HODConnectionSpec();
    cs.setHostname("trutycics.austin.ibm.com");
    cs.setPortNumber("5555");
    cs.setDebugScreenEnabled(true);

    // START SESSION

    System.out.println("Start Session");
    try {
        com.ibm.ivj.communications.Session.startSession();
    }
    catch (com.ibm.ivj.communications.SessionAlreadyStartedException e) {
        System.out.println("SessionAlreadyStartedException caught and ignored.");
    }

    // PREPARE KEY AND PAO FOR INSERT

    key = new MenuCustomerKey();
    key.setNumber(keynum1);
    customer = (MenuCustomer) MenuCustomer.find(key);
    customer.setConnectionSpec(cs);
    customer.setName("Isaac Newton");
    customer.setAddress("1687 Principia St.");
    customer.setPhone("998263");
    customer.setDate("42/12/25");
}
```

```

customer.setAmount("1727");
customer.setComment("fluxions");
System.out.println("\nAttempt insert of key " + key.getNumber());
try {
    customer.insert();
    System.out.println("insert successful");
}
catch (com.ibm.ipaa.IDataKeyAlreadyExistsException e) {
    System.out.println("insert failed: IDataKeyAlreadyExistsException");
}

// ATTEMPT UPDATE

key = new MenuCustomerKey();
key.setNumber(keynum1);
customer = (MenuCustomer) MenuCustomer.find(key);
customer.setConnectionSpec(cs);
customer.setAmount("1700");
customer.setComment("jupiter");
System.out.println("\nAttempt update of key " + key.getNumber() + ", amount 1700");
try {
    customer.update();
    System.out.println("update successful");
}
catch (com.ibm.ipaa.IDataKeyNotFoundException e) {
    System.out.println("ERROR, update failed: IDataKeyNotFoundException");
    e.printStackTrace(System.out);
}

// ATTEMPT DEBIT

key = new MenuCustomerKey();
key.setNumber(keynum1);
customer = (MenuCustomer) MenuCustomer.find(key);
customer.setConnectionSpec(cs);
System.out.println("\nAttempt debit of key " + key.getNumber() + ", amount $150");
customer.debit(150);

// ATTEMPT RETRIEVE

key = new MenuCustomerKey();
key.setNumber(keynum1);
customer = (MenuCustomer) MenuCustomer.find(key);
customer.setConnectionSpec(cs);
System.out.println("\nAttempt retrieve of key " + key.getNumber());
try {
    customer.retrieve();
    System.out.println("retrieve successful, data is...");
    System.out.println(" Name is " + customer.getName());
    System.out.println(" Address is " + customer.getAddress());
    System.out.println(" Phone is " + customer.getPhone());
    System.out.println(" Date is " + customer.getDate());
    System.out.println(" Amount is " + customer.getAmount());
    System.out.println(" Comment is " + customer.getComment());
}
catch (com.ibm.ipaa.IDataKeyNotFoundException e) {
    System.out.println("ERROR, retrieve failed: IDataKeyNotFoundException");
    e.printStackTrace(System.out);
}

```

```

    }

    // END SESSION

    System.out.println("\nEnd Session");
    try {
        com.ibm.ivj.communications.Session.endSession(true);
    }
    catch (com.ibm.ivj.communications.NoSessionStartedException e) {
        System.out.println("NoSessionStartedException caught and ignored.");
    }

    // EXIT SUCCESS

    System.out.println("\nExit testcase");
    System.exit(0);
}

```

Notice the pattern of MenuCustomer instance creation. The role of a workspace ID is similar to that of a session. Through your main method, therefore, you should use the same number on all setWorkspaceId calls. The call on the find method is related to cache management. If you already have an instance with the same key (for example, number), this call allows you to reuse that instance.

## Run the Unit Test Main Method

To run this unit test program perform the following procedure:

1. Right-click on the Execute class and select Properties. This opens the properties window.
  - a. Click on the **Class Path** tab.
  - b. Select the Include '.' (dot) in class path check box.
  - c. Select the Project path check box.
  - d. Click the **Edit...** button that is next to the project path. This opens a list of projects.
  - e. Select the following projects:
    - IBM Common Connector Framework
    - IBM Component Broker Connectors
    - IBM Component Broker Host On Demand
    - IBM Enterprise Access Builder Library
    - IBM Java Record Library
    - IBM Procedural Application Adapter
  - f. Click the **OK** button.
2. Right-click on the Execute class and select **Run** → **Run main**. You should see the Console window appear and the following messages are shown in the output pane.

```
Start Session
```

```
Attempt insert of key 001671
```

```
IBM eNetwork Host Access Class Library, Version 1.0.2
Copyright IBM Corporation 1997, 1998. All rights reserved.
```

```
insert successful
```

```
Attempt update of key 001671, amount 1700
update successful
```

```
Attempt debit of key 001671, amount $150
```

```
Attempt retrieve of key 001671
retrieve successful, data is...
  Name is Isaac Newton
  Address is 1687 Principia St.
  Phone is 998263
  Date is 42/12/25
  Amount is $1550.00
  Comment is
```

```
End Session
```

```
Exit testcase
```

As the unit test runs, you can also see a CICS terminal window in which the application screens appear, flashing quickly.

## Exporting the MenuCustomer Package

After building the Execute class and creating and testing the Component Broker procedural adapter object within the VisualAge for Java environment, you can run the unit test program outside of the VisualAge for Java environment. This object needs to be imported to Object Builder as a persistent object. Importing this object requires that the procedural adapter object and its corresponding BeanInfo class is exported outside of VisualAge for Java. To run the sample outside of the VisualAge for Java environment, you must export all classes you created and modify the CLASSPATH environment variable.

Export the entire package. The package should contain:

- The new procedural adapter object
- Its corresponding BeanInfo class
- All EAB transaction objects

To export the package outside of VisualAge for Java:

1. Select the package to export.
2. From the VisualAge for Java Workbench menu, select **File** → **Export**. This opens the Export - wizard. You are going to export all
3. Select **Directory**.
4. Click **Next**.

**WIN** For Windows NT, type x: \MyProj in the Directory field.

**AIX** For AIX, type \$HOME/MyProj in the Directory field.

5. Select the **.class** and **.java** check boxes.
6. Click the **Finish** button.

When the export completes, the paa.mysamples.cics.menu directory is created under the MyProj directory. You can exit VisualAge for Java.

To verify that you exported the package correctly, you can run the unit test program from the command line.

1. Ensure that your Working Directory is in your CLASSPATH.
2. From a command prompt, enter one of the following:

```
WIN java -nojit paa.mysamples.cics.menu.Execute
```

```
AIX java paa.mysamples.cics.menu.Execute
```

You should have the same results as you did when running inside of VisualAge for Java.

---

## Developing a CICS-HOD Business Object

This section contains Object Builder and System Management procedures required to create a component named "Acct." To create this component, perform the procedures in the following sections:

1. "Importing the Bean"
2. "Defining the Acct Component" on page 133
3. "Creating Client and Server DLL Files" on page 137
4. "Packaging the Application" on page 138
5. "Building the Application - Client and Server" on page 140
6. "Installing the Application" on page 141
7. "Running the Sample Application" on page 142

### Note:

1. Before starting Object Builder, ensure that your classpath includes your Working Directory.
2. Specify your Working Directory as the base directory for the project.
3. The procedures contained in this section assume that you have correctly set your classpath to include your Working Directory before starting Object Builder and that you have started Object Builder.

## Importing the Bean

The bean to import is MenuCustomer from the **paa.mysamples.cics.menu** package from your Working Directory.

To import this bean:

1. From the Object Builder Tasks and Objects pane, select the User-Defined PA Schemas folder.
2. Open the pop-up menu for User-Defined PA Schemas, and select **Import Bean**. This opens the Import Procedural Adaptor Bean - SmartGuide.
3. On this page:
  - a. Type `paa.mysamples.cics.menu.MenuCustomer` in the **Bean Name** field.
  - b. Click **Next** to accept the remaining defaults and to continue to the Procedural Adaptor Bean Names and Services page.
4. On this page
  - a. A panel is displayed prompting for the Module Name. Leave this field blank and click the **HOD** radio button.
  - b. click **Next** to accept the defaults and continue to the next page.
5. On this page:
  - a. Select the **number** property from the **Properties** list box.
  - b. Click >> to move the associated key required to import the bean.
6. Click the **Finish** button.

The bean is imported into Object Builder. The MenuCustomer schema and its corresponding persistent object (MenuCustomerPO) are now in the tree view of User-Defined PA Schemas.

## Defining the Acct Component

This exercise defines the objects required to create a component named Acct. For this component, you will:

1. Create a new business object file
2. Define the business object
3. Modify the data object interface (optional)
4. Connect the pushdown methods to the data object (optional)
5. Connect the data object implementation to the persistent object
6. Define the managed object
7. Generate the code

## Creating the Business Object File

To create the Acct business object file:

1. From the Tasks and Objects pane, select the User-Defined Business Objects folder.
2. Open the pop-up menu for User-Defined Business Objects, and select **Add File**, which opens the Business Object File - wizard to the Name page.
3. On this page:
  - a. Type Acct in the **Name** field.
  - b. Accept the other defaults.
4. Click the **Finish** button.

The Acct file is now under the User-Defined Business Objects folder.

## Creating the Business Object

After creating the new business object file, the business object needs to be defined. A fully-configured business object consists of the following:

- A business object interface
- An associated key
- An associated copy helper
- A business object implementation and data object interface

**Defining the Business Object Interface:** To create the Acct business object interface:

1. Expand the User-Defined Business Object folder and select Acct.
2. Open the pop-up menu for Acct and select **Add Interface**. This displays the Name page of the Business Object Interface - SmartGuide.
3. On this page:
  - a. Type Acct in the **Name** field.
  - b. Click **Next** to continue to the Constructs page.
4. Click **Next** to accept the defaults and to continue to the Interface Inheritance page.
5. Click **Next** to accept the defaults and to continue to the Attributes page.
6. Define the user-defined attributes.

- a. Select Attributes from the tree view.
- b. Open the pop-up menu for Attributes and select **Add**. This displays the Add dialog.
- c. In this dialog:
  - 1) Type number in the **Attribute Name** field.
  - 2) Select string as the **Type**. This displays the **Size** field.
  - 3) Type 0 in the **Size** field
  - 4) Click **Add Another**.
  - 5) Repeat steps 1 — 3 above for each attribute of the Acct interface, using string on each attribute. The remaining attributes are:
    - name, and click Add Another.
    - address and click Add Another.
    - phone and click Add Another.
    - Date and click Add Another.
    - amount and click Add Another.
    - comment and click Refresh.
  - 6) Click **Next** to continue to the Methods page.
- d. (Optional) On this page:
  - 1) Select Methods from the tree view.
  - 2) Open the pop-up menu for Methods and select **Add**. This displays the Add dialog.
  - 3) In this dialog:
    - a) Type debit in the Method Name field.
    - b) Click **Refresh**.
  - 4) Select Parameters from the tree view.
  - 5) Open the pop-up menu for Parameters and select **Add**. This displays the Add dialog.
  - 6) In this dialog:
    - a) Type amount in the **Parameter Name** field.
    - b) Click **Refresh**.
- e. Click **Finish**.

The Acct interface is now under the Acct file.

**Defining the Key:** To add the key:

1. From the User-Defined Business Object folder, select the Acct interface.
2. Open the pop-up menu of Acct, and select **Add Key**. This displays the Key -Name and Key Attributes wizard.
3. Select the **number** attribute from the **Business Object Attributes** list.
4. Click >> to move the attribute to the **Key Attributes** list.
5. Click **Finish**.

The AcctKey key is now under the Acct interface.

**Defining the Copy Helper:** To add the copy helper:

1. From the User-Defined Business Object folder, select the Acct interface.
2. Open the pop-up menu for Acct and select **Add Copy Helper**. This displays the Copy Helper - Name and Attributes wizard.
3. Click >> to move the attributes from the **Business Object Attributes** list to the **Copy Helper Attributes** list.
4. Click **Finish**.

The AcctCopy copy helper is now under the Acct interface.

**Defining the Business Object Implementation and Data Object Interface:** To add the Business Object Implementation and Data Object interface:

1. From the User-Defined Business Object folder, select the Acct interface.
2. Open the pop-up menu for Acct and select **Add Implementation**. This displays the Name and Data Access Pattern page of the Business Object Implementation - wizard.
3. Type AcctB0 in the **File Name** field.
4. Define the implementation.
  - a. Select the **Delegating** radio button from the **Pattern for Handling State Data** group.
  - b. Ensure that the **Create a new one now** radio button is selected from the **Data Object Interface** group box. This option allows you to define the business object attributes that need to be preserved in the data object.
  - c. Deselect 390 in the Select Deployment platform group box.
  - d. Click **Next** to continue to the Implementation Inheritance page.
5. Click **Next** to accept the defaults and to continue to the Implementation Language page.
6. Select C++ for the implementation language, and then click **Next** to accept the defaults and to continue to the Attributes page.
7. Click **Next** to accept the defaults and to continue to the Methods page.
8. Click **Next** to accept the defaults and to continue to the Key and Copy Helper page.
9. On this page:
  - a. Verify that the **AcctKey** key is selected from the **Key** list.
  - b. Verify that **AcctCopy** is selected from the **Copy Helper** list.
  - c. Click **Next** to continue to the Handle Selection page.
10. Click **Next** to accept the defaults and to continue to the Attributes to Override page.
11. Click **Next** to accept the defaults and to continue to the Methods to Override page.
12. Click **Next** to accept the defaults and to continue to the Data Object Interface page.
13. Click the >> button to move the attributes in the **Business Object Attributes** list to the **State Data** list.
14. Click **Next** to continue to the Data Object Methods page.
15. Select debit (if defined) on the left panel and move it to the right panel.
16. Click **Finish**.

The AcctBO business object implementation is now under the Acct interface, and the AcctDO data object interface is now under the AcctBO business object implementation.

## Connecting the Data Object Implementation to the Persistent Object

To create the data object implementation and to connect the data object implementation to the persistent object:

1. From the User-Defined Business Objects folder, select the AcctDO data object interface.
2. Open the pop-up menu for AcctDO and select **Add Implementation**, which opens the Data Object Implementation - Name and Platform page.
3. Deselect 390 in the Select deployment platform group box.
4. Click **Next** to accept the defaults and to continue to the Behavior page.
5. On this page:
  - a. Set the **BOIM with any key** radio button from the **Environment** group box to indicate that the data object is part of a component installed in a business object application adapter with instances being located by key objects.
  - b. Set the **Procedural Adaptors** radio button from the **Form of Persistent Behavior and Implementation** group box.
  - c. Click the **Next** button to continue to the Implementation Inheritance page.
6. On this page:
  - a. Verify that IPAAExtLocalToServer::IDataObject is selected as a parent.
  - b. Click **Next** to continue to the Attributes page.
7. Click **Next** to accept the defaults and to continue to the Methods page.
8. Click **Next** to accept the defaults and to continue to the Key and Copy Helper page.
9. Click **Next** to accept the defaults and to continue to the Associated Persistent Objects page.
10. On this page:
  - a. Select Persistent Object Instances.
  - b. Open the pop-up menu for Persistent Object Instances and select **Add**.
  - c. Type iMenuCustomerPAOPO in the **Instance Name** field.
  - d. Click **Next** to continue to the Attributes Mapping page.
11. On this page:
  - a. Select number from the **Attributes** list.
  - b. Open the pop-up menu for number and select **Primitive**.
  - c. Select **iMenuCustomerPAOPO .number** from the **Persistent Object Attribute** list.
  - d. Add 1-to-1 mappings for the other attributes in the **Attributes** tree view as you did for number.
  - e. Click **Next** to continue to the Methods Mapping page.
12. On this page:
  - a. Select insert from the **Special Framework Methods** list.
  - b. Open the pop-up menu for insert and select **Add Mapping**.
  - c. Select **iMenuCustomerPAOPO .insert** from the **Persistent Object Method** list.
  - d. Add 1-to-1 mappings for the other methods in the **Special Framework Methods** tree view as you did for insert. In addition, add a mapping from setConnection() to iMenuCustomerPO.setConnection(metadata).

- e. Select debit from the **User-defined Methods** list (if you defined this method).
  - f. Open the pop-up menu for debit and select **Add Mapping**.
  - g. Ensure that **iMenuCustomerPAOPO .debit** is selected from the **Persistent Object Method** list.
13. Click **Finish**.

The AcctDOImpl data object implementation is now under the AcctDO interface, and the MenuCustomerPO persistent object is now under the AcctDOImpl data object implementation.

## Defining the Managed Object

To add the managed object:

1. From the User-Defined Business Objects folder, select the AcctBO business object implementation.
2. Open the pop-up menu for AcctBO and select **Add Managed Object**. This displays the Name and Services page of the Managed Object - wizard.
3. On this page, deselect **390** on the Select deployment platform group box.
4. Set the **Session Service** radio button.
5. Click the **Next** button to accept the defaults and continue to the Implementation Inheritance page.
6. Click **Finish**.

## Generating the Code

To generate the application code:

1. From the User-Defined Business Objects folder, select Acct.
2. Open the pop-up menu for Acct and select **Generate** → **All**.

Code generation starts. Progress is indicated in the lower-left corner of the window.

## Creating Client and Server DLL Files

The defined objects need to be built into two separate DLL files.

- One that runs on the client and provides access to the business object interface, key, and copy helper.
- One that runs on the server and provides access to the managed object and the rest of the component.

The client DLL file needs to be defined before the server DLL file. When the server DLL file is defined, it needs to link to the client DLL file. After defining the objects that comprise each DLL file, these files can be built.

## Defining the Client DLL File

To define the client DLL file:

1. Select the Build Configuration folder.
2. Open the pop-up menu for Build Configuration, and select **Add Client DLL**. This displays the Name and Options page of the Client DLL - wizard.
3. Type AcctC in the **Name** field.
4. Check only the Applicable Platforms you want.

5. Click **Next** to continue to the Client Source Files page.
6. Click >> to move the client source files to the **Items chosen** list.
7. Click **Finish** on.

The AcctC client DLL file is now under the Build Configuration folder.

## Defining the Server DLL File

To define the server DLL.

1. Select the Build Configuration folder.
2. Open the pop-up menu for Build Configuration and select **Add Server DLL**. This displays the Name and Options page of the Server DLL - wizard.
3. Type AcctS in the **Name** field.
4. Check only the Applicable Platforms you want.
5. Click the **Next** button to continue to the Server Source Files page.
6. Click >> to move the server source files to the **Items chosen** list.
7. Click **Next** to continue to the Libraries to Link With page.
8. Select AcctC from the **Items Available** list.
9. Click >> to move AcctC.dll to the **Items Chosen** list.
10. Click **Finish**.

The AcctS server DLL file is now under the Builder Configuration folder.

## Generating the Makefiles

To generate the makefiles to build the configuration:

1. Select the Build Configuration folder.
2. Open the pop-up menu for Build Configuration, and select **Generate** → **All** → **All Targets**.

The code generation begins.

## Packaging the Application

Packaging the application consists of the following procedures:

1. Creating the application family
2. Defining the application
3. Creating the container instance
4. Configuring the managed object
5. Generating the applications.

## Creating the Application Family

To add the application family:

1. Select the Application Configuration folder.
2. Open the pop-up menu for Application Configuration and select **Add Application Family**. This displays the Name page of the Add Application Family - wizard.

3. Type AcctApp in the **Name** field.
4. Click the **Finish** button.

The AcctApp application family is now under the Application Configuration folder.

## Defining the Application

To add the Application:

1. Select the AcctApp application family.
2. Open the pop-up menu for AcctApp, and select **Add Application**. This displays the Name and Environment page of the Add Application - wizard.
3. Type Acct in the **Application Name** field.
4. Click the **Finish** button.

The Acct application is now under the AcctApp application family.

## Creating the Container Instance

To add the new container instance:

1. Select the Container Definition folder.
2. Open the pop-up menu for Container Definition and select **Add Container Instance**. This displays the Container - wizard.
3. Type AcctContainer in the **Name** field.
4. Deselect **390** on the Select deployment platform group box.
5. Click **Next** to continue to the Work Load Management page.
6. Click **Next** to continue to the Services page.
7. On the Services page, set the **Use PAA Session Services** radio button.
8. Click **Next** to continue to the Services Details page.
9. On this page, type **CICS\_Acct\_Server** in the **Connection Name** field.
10. Set the **HOD** radio button in the Connector Type used by a **Session** group box.
11. Click **Finish**.

The AcctContainer container is now under the Container Definition folder.

## Configuring the Managed Object

To add the managed object for the Application:

1. Open AcctApp under the Application Configuration folder.
2. Select the Acct application.
3. Open the pop-up menu for Acct and select **Add Managed Object**. This displays the Configure Object - wizard.
4. In this window:
  - a. Verify that AcctMO AcctMO is in the **Managed Object** field.
  - b. Click the **Next** button to continue to the Data Object Implementations page.

5. On this page:
  - a. Select Implementations.
  - b. Open the pop-up menu for Implementations and select **Add**.
  - c. Select AcctDOImpl AcctDOImpl from the **Data Object Implementation** list.
  - d. Click **Next** to continue to the Container page.
6. Click the **Next** button to continue to the Home page.
7. On this page, select BOIMHomeOfRegHomes from the **Home Name** list.
8. Click the **Finish** button.

The AcctMO managed object is now under the Application Configuration folder.

## Generating the Application

To generate the files for the application family:

1. Select the AcctApp application.
2. Open the pop-up menu for AcctApp and select **Generate**.

WIN For Windows NT only, if you do not have InstallShield installed on your system, click the **Yes** button when the dialog concerning InstallShield is displayed, or set the InstallShield location with File → Preferences → Tasks and Objects.

When code generation completes, the Method Implementation pane contains the AcctApp.ddl file. You can now close Object Builder.

## Building the Application - Client and Server

All imported and generated files are placed in one of the following directories:

WIN The Working\NT directory (where Working\NT is the subdirectory of your Working Directory).

Change your directory to:

```
x:\MyProj\Working\NT
```

Enter:

```
nmake -f all.mak cpp java
```

AIX The Working/AIX directory (where Working/AIX is the subdirectory of your Working Directory).

Change your directory to:

```
$HOME/MyProj/Working/AIX
```

Enter:

```
make -f all.mak cpp java
```

Everything in the sample application is built.

## Installing the Application

Installing an application consists of:

1. Loading the application
2. Configuring the application

These procedures assume that you are currently logged on to DCE and that you are currently using the System Manager User Interface. If not, logon to DCE and start the System Manager User Interface.

## Loading the Application onto System Management

To install the Acct server application:

1. Start the System Manager User Interface, if it is not already started.
2. Become an Expert user (**View** → **User Level** → **Expert**).
3. Expand Host Images and select <your host name>.
4. From the pop-menu, select **Load Application**. This opens the Load Application dialog.
5. Browse for and select AcctApp.ddl for one of the following:

For Windows NT, x:\MyProj\Working\NT\AcctApp\AcctApp.ddl

For AIX, \$HOME/MyProj/Working/AIX/AcctApp/AcctApp.ddl

**Note:** A warning may be displayed about iCachedWLMSystemManagedObjects while the DDL is loading. You can ignore this warning.

## Configuring the Application with System Management

To configure the application:

1. Expand Available Applications and select **Acct**.
2. Open the pop-up menu for Acct and select **Drag**.
3. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select Sample Configuration.
4. Open the pop-up menu for Sample Configuration, and select **Add Application**.
5. Configure the HOD connection.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → HOD Connections, and select CICS\_Acct\_Server.
  - b. Open the pop-up menu for CICS\_Acct\_Server, and select **Edit**, which opens the Object Editor.
  - c. Click the **Main** tab.
  - d. Modify the **host name** and **port number** fields to match the CICS region with which you are communicating.
  - e. Click the **OK** button to validate and apply the changes.
6. Define the server.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select Sample Configuration.
  - b. Open the pop-up menu for Sample Configuration and select **New** → **Server (free standing)**. This displays a new dialog box.
  - c. Type AcctSrv as the name for the server.

- d. Click the **OK** button. The AcctSrv is now under Server (free standing).
7. Associate the application with the server.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Applications, and select Acct.
  - b. Open the pop-up menu for Acct and select **Drag**.
  - c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Server (free standing), and select AcctSrv.
  - d. Open the pop-up menu for AcctSrv and select **Configure Application**.
8. Associate the iPAAServices application with the server.
  - a. Open the pop-up menu for iPAAServices and select **Drag**.
  - b. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Server (free standing), and select AcctSrv.
  - c. Open the pop-up menu for AcctSrv and select **Configure Application**.
9. Configure the server with the host.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Server (free standing), and select AcctSrv
  - b. From the pop-up menu for AcctSrv, select **Drag**.
  - c. Expand Hosts and select your server.
  - d. From the pop-up menu for your server, select **Configure Server (free standing)**.
10. Activate the configuration.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select Sample Configuration.
  - b. Open the pop-up menu for Sample Configuration, and select **Activate**, which automatically starts the application server. Wait for the completion message in the Action Console window before continuing.

## Running the Sample Application

For IVP install instructions for Windows NT, see Chapter 7 in the *IBM Transaction Server for Windows NT Installation Guide, Version 4*. This chapter, entitled “Performing the Installation Verification Procedures,” discusses how to load and run the IVP programs for CICS.

For IVP install instruction for MVS, see Section 2.6 in the *IBM CICS Transaction Server for OS/390 CICS Installation Guide*. This section discusses installing and running the IVP jobs.

To run the sample client application:

1. Do one of the following:

WIN

- a. Copy `acctcli.mak` and `acctcli.cpp` from:

`x:\cbroker\samples\InstallVerification\PAA\Application\acctcli`

to

`x:\Myproj\Working\NT`

- b. Change directory to:

x:\MyProj\Working\NT

c. Enter the following:

```
nmake -f acctcli.mak cpp java
```

AIX

a. Copy acctcli.mak and acctCli.cpp from:

```
$HOME/samples/InstallVerification/PAA/Application/acctcli
```

to

```
$HOME/MyProj/Working/AIX
```

b. Change directory to:

```
$HOME/MyProj/Working/AIX
```

c. Enter the following:

```
make -f acctcli.mak cpp java
```

d. Enter: acctcli



---

## Chapter 7. Developing a CICS-ECI Application

This chapter provides information for building a sample Component Broker application with a CICS backend. This chapter contains the following information:

- “The CICS-ECI Sample Application”
- “Developing an Application to Retrieve Cash Account Information from a CICS Server” on page 146
- “Developing a CICS-ECI Business Object” on page 159

**Note:** To walk-through this sample, the CICS and IMS Application Adaptor SDK and IBM VisualAge for Java with EAB must be installed.

### Important Information

Before walking through this sample, please refer to the *Late Breaking News* provided with Component Broker before performing the exercise in this chapter. This document provides the latest information regarding the CICS and IMS application adaptor samples, that may differ from the instructions for this sample application.

---

## The CICS-ECI Sample Application

The CICS-ECI sample application is a mock account database consisting of the following fields:

- Account Balance
- Account Number
- Type of customer
- Type of account
- Utilities

The CashAcct interface will be implemented during this exercise. The data object implementation for this business object will leverage a procedural adaptor object that in turn uses CICS through the ECI to provide the state data back to the data object.

Although this sample application is not a full-blown CICS application, it captures the essence of an application involving multiple ECI requests and delivering some amount of business function. This sample application can be extended and customized to explore different CICS-ECI application issues.

The sample that you build in this section is included with the product and can be built by following the steps in the HTML file in one of the following:

WIN

`CBroker\samples\InstallVerification\PAA\readme.htm`

AIX

`$HOME/samples/InstallVerification/PAA/readme.htm`

---

## Developing an Application to Retrieve Cash Account Information from a CICS Server

The following document explains the necessary steps to create an application that interfaces with a CICS transaction. Cash Account information is retrieved from a CICS server using an account ID and a `res_type`. The information is returned to a Procedural Adapter Object that has various attributes that map to the fields that are returned from the CICS transaction.

This process will create a variety of different components that work together to form the basis of our interaction to the CICS system. A Connection specification holds all connection relevant properties, for example hostname and portnumber. An Interaction specification holds all request relevant properties, for example, the name of the program to call, the interaction mode (SEND\_RECEIVE, SEND, RECEIVE, ...).

A Record Bean wraps a data buffer in a Java representation of an external structural definition. Their implementation is based on the RecordJava Framework. Importers generate DynamicRecordTypes (meta data format) from COBOL, BMS, MFS. An Editor allows to either create DynamicRecordTypes by hand, or to refine a generated one. From DynamicRecordTypes, a Generator creates static or dynamic Record/RecordBeans.

A Command is a bean wrapping a single interaction with a backend system. It contains both inputs and outputs defined by a RecordBean (or multiple RecordBeans). Properties (or the RecordBean itself) get exposed as inputs/outputs to the system. Once created, an execute method drives the execution of the command. A successful execution is signaled by the successful event, while an unsuccessful execution (exception occurred) is signaled by the unsuccessful event. Commands are constructed using the Visual Composition Editor (VCE), or the Command Builder. Commands are used to implement behavior of BusinessObjects, that is *persistence* and pushdown.

A Navigation is a bean wrapping multiple interactions with a backend system. It is composed from Commands and other Navigations. From the outside, it looks like a Command, since it is a Command. Navigators are constructed using the Visual Composition Editor. Navigators are used to implement behavior of BusinessObjects, that is persistence and pushdown.

A Business Object/ProceduralAdapterObject is an object that maintains state. Business Objects are beans managed on a Key basis. For each class there is a Key class that defines the key properties of the Business Object. The Key provides the behavior to get an instance of a Business Object. These business objects can be instantiated over time and are managed by an instance space. A Business Object and its Key are created, either by implementing the interfaces IBusinessObject/IBusinessObjectKey, or extending the classes BusinessObject/BusinessObjectKey. A Business Object and its Key are constructed using the standard class/bean creation wizards offered by VisualAge for Java Commands, and Navigators are used to implement Business Object behavior, that is persistence (CRUD) or pushdown. The Procedural Application Object (PAO) is extending the Business Object with an interface that makes it consumable by Component Broker and EJS.

Lastly, a Mapper is a bean mapping properties between a base bean and a collection of beans. It is just a special case that the base bean is a RecordBean and the collection is a set of Business Objects. Mapping can be key based, or non-key based. Key based maps between the base bean and bean instances managed by an instance space. Non-key based maps between the base bean and a defined collection of beans. Mappers are constructed using the Mapper Builder. The Mapper Builder uses appropriate metaphors for the base bean, for example panel. Mappers can be associated with the input and output beans of a Command, automating the flow of property values to and from the Command.

Following is an overview of the steps:

1. "Creating a Project and Package Under VisualAge for Java" on page 148

2. “Creating the Procedural Adaptor Object and Key” on page 148
3. “Importing the Customer COBOL File” on page 150
4. “Creating the Record Mapper” on page 151
5. “Creating the BeCashAcctCommand Class” on page 152
6. “Modifying the Procedural Adaptor Object to Call the Commands” on page 154
7. “Creating an Executable Class” on page 156
8. “Running the Sample Application” on page 170

**WIN** If you are using VisualAge for Java on Windows 95 or Windows NT, from the **Start** menu, select **Programs → IBM VisualAge for Java for Windows → IBM VisualAge for Java**.

**AIX** If you are using Visual Age for Java on AIX, type `vajide` on the command line.

If the VisualAge Quick Start dialog appears, select **Go to the Workbench** and click **OK**. The IDE appears.

From the Window pull down, select **Options**. Select **Design Time** and un-check **Inherit BeanInfo** of bean superclass. Click **OK**.

## Importing Prerequisite Features into the Workspace

1. Select **File → Quick Start**.
2. Select **Features** in the left pane and **Add Feature** in the right pane.
3. Click **OK**.
4. Select the following:
  - IBM Procedural Application Adapter 1.0
  - CICS Connector 3.0
  - IBM Component Broker Host On Demand 1.0
  - IBM Component Broker Connectors 1.0
  - IBM Enterprise Access Builder Library 2.0
  - IBM Enterprise CICS Access Builder Library 1.0
  - IBM Component Broker PAA Samples for CICS 1.0
5. Click **OK**.

You may ignore the following expected errors this introduces in the following packages:

- `com.ibm.ivj.communications`
- `com.ibm.ivj.trace`
- `com.ibm.eNetwork.ECL`
- `com.ibm.eNetwork.ncod.services.RAS`

**Note:** If you do not see all of these features listed, they have been previously installed. To confirm, perform the following steps:

- a. Select **File → Quick Start**.
- b. Select **Features → Delete Feature** and see which features are already loaded (then **Cancel**).

## Creating a Project and Package Under VisualAge for Java

If the Component Broker Samples project does not exist, complete the following steps:

1. Right click on the Visual Age for Java desktop icon.

**WIN** For Windows NT, click on **Start** → **Programs** → **IBM Visual Age for Java for Windows**.

**AIX** For AIX, type `vajide` on the command line. Ensure that you follow the preceding instructions for un-checking Inherit BeanInfo of bean superclass.

2. From the VisualAge for Java menu, select **Add Project**.
3. In the **Project Name** field, type `CBSamp1es` and click **Finish**. The Component Broker Samples project should be under the VisualAge for Java list of projects.
4. From the list of projects, select **Component Broker Samples**.
5. From the **Component Broker Samples** menu, select **Add Package**. This creates a package for the project.
6. Type `paa.mysamp1es.cics.eci.acct` for the new package and click **Finish**.

### Note:

If you are using the default mouse configuration, right-click on the denoted item. You do not have to select the item before opening its menu. You can select the item and open its menu with a right-click.

## Creating the Procedural Adaptor Object and Key

The procedural adaptor object inherits from `com.ibm.ivj.eab.paa.EntityProceduralAdapterObject`, that serves as a base implementation for all procedural adaptor objects. As a subclass of `EntityProceduralAdapterObject`, the procedural adaptor object contains the CRUD methods (`insert()`, `retrieve()`, `update()`, and `del()`). However, these methods are all empty-bodied. You must define their implementation for your procedural adaptor object.

The attributes defined in the `BeCashAcct` interface are essential. Thus, the procedural adaptor object, as the adaptor that connects the Component Broker data object to the back-end system, should contain the properties that correspond to these attributes.

Perform the following steps:

1. From the VisualAge for Java desktop under the Component Broker Samples project, select **paa.mysamp1es.cics.eci.acct**.
2. From the pop-up menu for `paa.mysamp1es.cics.eci.acct`, select **Add** → **Class**.
3. In the dialog:
  - a. Type `BeCashAcctPA0` in the **Class name** field.
  - b. Select the **Superclass**:
    - 1) Click **Browse**.
    - 2) Select **EntityProceduralAdapterObject** as your Superclass.
    - 3) Click **OK**.
4. Click **Finish**.

Add the properties for the **BeCashAcctPA0** class by completing the following steps:

1. Select the **BeCashAcctPAO** class.
2. From the menu for BeCashAcctPAO, select **Open** to open the Object Editor notebook.
3. In the notebook, select the **BeanInfo** tab.
4. For each new property, perform the following steps:
  - a. From the menu bar, select **Features** → **New Property Feature** to open the wizard → New Property Feature.
    - 1) In the wizard, type the name of a new property in the **Property name** field. For simplicity, consider using the same name that is used in the CashAcct class. For example, specify balance as the first property as defined in the BeCashac.cpp file. Perform the following steps:
      - a) For all properties except balance, select `java.lang.String` from the **Property type** list. For the balance property, select `int` (not `int[]`).
      - b) Click **Finish**.
    - 1) Repeat the previous steps for each of the following properties:
      - `res_type`
      - `account_ID`
      - `type`
      - `utilities`
  - b. Close the object editor notebook.

Next, create the Key for the PAO object by performing the following steps:

1. From the VisualAge for Java desktop under the Component Broker Samples project, select **paa.mysamples.cics.eci.acct**.
2. From the pop-up menu for **paa.mysamples.cics.eci.acct**, select **Add** → **Class**.
3. In the dialog:
  - a. Type BeCashAcctPAOKey in the **Class name** field.
  - b. Select the Superclass:
    - 1) Click **Browse**.
    - 2) Select **BusinessObjectKey** as your Superclass.
    - 3) Click **OK**.
  - c. Click **Finish**.

Add the properties for the **BeCashAcctPAOKey** class by performing the following steps:

1. Select the **BeCashAcctPAOKey** class.
2. From the pop-up menu for BeCashAcctPAOKey, select **Open** to open the Object Editor notebook.
3. In the notebook, select the **BeanInfo** tab.
4. For each new property, perform the following steps:
  - a. From the menu bar, select **Features** → **New Property Feature** to open the wizard → New Property Feature.
  - b. In the wizard:
    - 1) Type the name of a new property in the **Property name** field. For example, type `res_type` as one of the properties that are going to be key attributes.
    - 2) Select **java.lang.String** for the type of the property.

- 3) Accept the other defaults and click **Finish**.
5. Repeat the steps above for the account\_ID property.
6. Close the Object Editor Notebook.

Modify the BeCashAcctPAOKey and BeCashAcctPAO to tie the PAO and key classes together. Perform the following steps listed in the steps below.

## BeCashAcctPAOKey

1. Select and expand the **BeCashAcctPAOKey** class.
2. Select the **getPropertyValues()** method. This method is used by the EAB run time to calculate a value to key into the EAB cache. It must be modified to specifically return just the key values.
3. In the source window, return an array of Objects that make up the key by invoking the methods that get the key properties. For example:
 

```
return new Object[] { this.getRes_type(), this.getAccount_ID() };
```
4. Save the changes to the modified PAO Key class by pressing **Ctrl-S**.

## BeCashAcctPAO

1. Select and expand the **BeCashAcctPAO** class.
2. Modify the getters for the key property values ( `getAccount_ID()` and `getRes_type()` ) by getting the key class associated with this PAO and returning that value.

In `getAccount_ID()`

```
BeCashAcctPAOKey key = (BeCashAcctPAOKey) this.getKey();
return key.getAccount_ID();
```

In `getRes_type()`

```
BeCashAcctPAOKey key = (BeCashAcctPAOKey) this.getKey();
return key.getRes_type();
```

3. Save the changes to the modified PAO by pressing **Ctrl-S**.

## Importing the Customer COBOL File

To import the customer COBOL file, perform the following steps:

1. Select the package that you created, for example, `paa.mysamples.cics.eci.acct`.
2. From the pop-up menu for the package with which you are working, select **Tools** → **Records** → **Create Cobol Record Type....** A wizard window appears.
3. In the wizard window:
  - a. In the **Class Name** field, enter `BeCashAcctInfo`.
  - b. In the **COBOL File** field, browse through the files to locate the `BeCashAcct.ccp` file. It should be located in one of the following:
 

```
WIN C:\Broke\samples\InstallVerification\PAA\Backend\CashAcct\Model
AIX $HOME/samples/InstallVerification/PAA/Backend/CashAcct/Model
```
  - c. Select **Open**.
  - d. Verify that the Project and Package names are correct. You can ignore the warning message in the log window about level 88 record(s) found.

- e. Select **Next** to continue.
4. In the next window:
  - a. In the list of **Available level 01 commareas** select **WS-COMMAREA-BUFFER** and select > to move it to the **Selected commareas** list.
  - b. Check the **Record Type intended for CICS** checkbox.
  - c. Select **Finish** when this is complete. A new class named BeCashAcctInfo appears in the designated package.
5. Select the **BeCashAcctInfo** class.
6. From the pop-up menu for the BeCashAcctInfo class, select **Tools** → **Records** → **Generate Records....** The Generate Records wizard appears.
7. In the wizard window:
  - a. In the **Class Name** field enter BeCashAcctRecord.
  - b. Select the **Beans** radio button to generate the records as beans.
  - c. Select the **Direct** radio button to access the record fields directly.
  - d. Select the **Dynamic Records** radio button to generate the records as dynamic records.
  - e. Check that the Project and Package names are correct and click **Next**.
8. In the next window:
  - a. Change the values in the following fields to the correct values for the CICS server:
    - Floating Point Format
    - Endian
    - Remote Integer Endian
    - Code Page
    - Machine Type

For example, the code page for North American MVS is 037 and for North American NT is 437.

WIN You must change all of the values if you are going to a Transaction Server on Windows NT. Also remember to change your endianness to littleEndian because you are running this from Windows NT.
  - b. Click the **Finish** button when this is complete. The following new classes appear in your package:
    - BeCashAcctRecord
    - BeCashAcctRecordBeanInfo
    - BeCashAcctRecordType

## Creating the Record Mapper

To create the mapper file, perform the following steps:

1. Select the sample package that you created.
2. Select the **BeCashAcctRecord** class.
3. From the pop-up menu for the package under which you are working, select **Tools** → **Mapper Editor....** A Mapper wizard appears.
4. In the wizard window:
  - a. From the **Code Generation** menu, select **Set Target mapper**. A window containing three fields appears.

- b. In the window:
  - 1) Enter the project and package names of this sample in their corresponding fields.
  - 2) In the **Class**, field enter BeCashAcctRecordMapper. Click **OK** when this is completed.
- c. Select **Change Input bean** from the **Code Generation** menu. A window containing one field appears.
- d. In the window:
  - 1) Click the **Browse...** button beside the **input** field. In the display window, enter **BeCashAcctRecord** in the **Pattern** field to display a list of matching classes.
  - 2) Select the **BeCashAcctRecord** class (corresponding to the sample package) and select **OK**.
  - 3) Click **OK** again to select the Input Bean class. The following message appears:  
All of your connections will be lost. Do you want to proceed?
  - 4) Select **Yes**.
  - 5) A list of fields available from **BeCashAcctRecord** is displayed.
- e. Select **Add** (located at the bottom left of the **Output Beans** window), and select the **BeCashAcctPAO** class corresponding to the package you are currently using. Click **OK** when this is complete. The java.lang.Object directory is displayed in the **Output Beans** side of the window.
- f. Expand this directory until the first instance of **BeCashAcctPAO** is visible (you should be able to see **account\_ID**, **res\_type**, **balance**, and so on).
- g. Select the **account\_ID** field of the BeCashAcctPAO object. Move the cursor to the right-hand side of the window and select **COMM\_\_ACCOUNTID**. At the bottom of the window, select the >> button to connect the two fields.  
  
**Note:** The **BeCashAcctPAO account\_ID** field should be connected to **COMM\_\_ACCOUNTID** on the input side. Repeat this step to form connections between the rest of the **BeCashAcctPAO** fields.
- h. Click **Apply** and **OK** when this is completed. A new class called BeCashAcctRecordMapper appears in the current package.

## Creating the BeCashAcctCommand Class

To create the BeCashAcctCommand Class, complete the following steps:

1. Select the package that you have been working in,  
`paa.mysamples.cics.eci.acct`  
.
2. Open the pop-up menu of the package, and select **Add** → **Class**. A wizard window appears.
3. In this window:
  - a. Enter the project and package names of the sample application into their corresponding fields.
  - b. Select the **Create a new class** radio button and type BeCashAcctCommand for the Class name.
  - c. To select the Superclass, click the **Browse** button and select **CommunicationCommand** from the list.
  - d. Click **OK**.
  - e. Ensure that the **Compose the class visually** radio button is NOT selected and click **Finish**.

4. Select the **BeCashAcctCommand** class again.
5. From the pop-up menu for the class, select **Open To** → **BeanInfo**.
6. In this Dialog:
  - a. Select **Features** → **Generate BeanInfo** class. This will generate a new BeanInfo class for your command class.
  - b. Select **Features** → **Add Available Features**.
    - 1) In the **Add Available Features** dialog, select the following features that may appear:
      - class
      - communication
      - connectionSpec
      - disconnectCommunication
      - expectedTriggerClass
      - input
      - interactionSpec
      - mappedObjects
      - mappingHelper
      - output
    - 2) Click on the **OK** button.
  - c. Close the Command Class.
7. Open the pop-up menu of the class, and select **Tools** → **Command Editor**. A new dialog is displayed.
8. In this dialog:
  - a. Right-mouse click on the **Communication Task** and select **Add ConnectionSpec**. A window will display all objects that inherit from ConnectionSpec.
  - b. In this window:
    - 1) Select **ECIConnectionSpec**.
    - 2) Click **OK** and the window will close and a connectionSpec entitled **ceConnectionSpec** is displayed under the **Communication Task**.

Right-mouse click on the **Communication Task** and select **Add InteractionSpec**. A window will display all objects that inherit from **InteractionSpec**
  - c. In this window:
    - 1) Select **ECIInteractionSpec**.
    - 2) Click **OK** and the window will close and an interactionSpec entitled **celInteractionSpec** is added under the **Communication Task**.

**Inbound side of command**
  - d. Right-mouse click on the **Input** task and select **Add IByteBuffer Bean**. A window will display all beans in VisualAge for Java. Select the RecordBean created earlier, **BeCashAcctRecord**, and click **OK**.
  - e. Right-mouse click on **celInput** and select **Promote Bean Feature**. Ensure that the **Property** radio button is selected and move **COMM\_\_ACCOUNTID**, **COMM\_\_REQUEST\_\_TYPE**, and **COMM\_\_RES\_\_TYPE** from the left pane to the right pane by highlighting those properties and clicking the >> button.
  - f. Click **OK** to generate run time code and the bean info class.

- g. Right-mouse click on **ceInput** and select **Add Mapper**. A window will display all mapper beans in VisualAge for Java. Select **BeCashAcctRecordMapper** and click **OK**. A **ceMapperCeInput** object should now be created under the **ceInput** object.

#### **Outbound side of command**

- h. Right-mouse click on the **Output** task and select **Add IByteBuffer Bean**. A window displays all beans in VisualAge for Java. Select the RecordBean created earlier, **BeCashAcctRecord**, and click **OK**.
- i. Right-mouse click on **ceOutput1** and select **Promote Bean Feature**. A window is displayed.
- j. Ensure that the **Property** radio button is selected, and highlight **COMM\_\_ACCOUNTID** and move it over. Do the same for **COMM\_\_RES\_\_TYPE** (These are the two key fields).
- k. Select the **Method** radio button and move the key attribute getters over:  
**getCOMM\_\_ACCOUNTID()** and **getCOMM\_\_RES\_\_TYPE()**. Move **getCOMM\_\_RETURN\_\_VALUE\_\_1()** over as well. Click **OK** when finished.
- l. Select **ceOutput1** and Add Mapper.
- m. Select **BeCashAcctRecordMapper** and click the **OK** button. A **ceMapperCeOutput1** object should now be created under the **ceOutput1** object. Setting properties.
- n. Select **ceConnectionSpec**, right-mouse click on it and select **Properties**. A window is displayed that allows you to change the bean properties.
- o. In this window:
- 1) In the **CICSServer** field, enter the name of the CICS server where the **BeCashAcct** program is located. for example, **H0TBOS**
  - 2) Enter in the **URL** field the address of the CICS gateway, for example, **local** :
  - 3) Click **OK** to close the property window and to save the changes.
9. Select **ceInteractionSpec**, right-mouse click on it and select **Properties**. A window is displayed that allows you to change the bean properties.
10. In this window:
- a. In the **programName** field, type the name of the COBOL file that is being used for the transaction, **BECASHAC**.
  - b. Optional - if the unit of work should be ended at the end of the command, change **CICSELUW** to **True**. If you do not care, leave **CICSELUW** as **False**.
  - c. Click **OK** to close the properties window.
11. Click **OK** to close the Command Editor and generate run time code.

## **Modifying the Procedural Adaptor Object to Call the Commands**

1. Select the **BeCashAcctPAO** class and expand it.
2. Highlight each of the CRUD methods (insert, retrieve update, and delete).
3. For each method, add in the necessary code to set up and call each Command Object. For the **insert()** method, the code should look like the following:

```
public void insert() throws com.ibm.ipaa.IDataKeyAlreadyExistsException {  
    BeCashAcctCommand bec = new BeCashAcctCommand();  
    bec.setConnectionSpec(this.getConnectionSpec());  
    bec.setCeInputCOMM__REQUEST__TYPE((short)1);  
    bec.setCeInputCOMM__ACCOUNTID(this.getAccount_ID());  
}
```

```

bec.setCeInputCOMM__RES__TYPE(this.getRes_type());
bec.execute();

if (bec.ceOutput1GetCOMM__RETURN__VALUE__1().equals("00000014"))
    throw new com.ibm.ipaa.IDataKeyAlreadyExistsException();
}

```

4. For the retrieve() method, the code should look like the following:

```

public void retrieve() throws com.ibm.ipaa.IDataKeyNotFoundException {
    BeCashAcctCommand bec = new BeCashAcctCommand();
    bec.setConnectionSpec(this.getConnectionSpec());
    bec.setCeInputCOMM__REQUEST__TYPE((short)2);
    bec.setCeInputCOMM__ACCOUNTID(this.getAccount_ID());
    bec.setCeInputCOMM__RES__TYPE(this.getRes_type());
    bec.setCeOutput1COMM__ACCOUNTID(this.getAccount_ID());
    bec.setCeOutput1COMM__RES__TYPE(this.getRes_type());
    bec.execute();

    if (bec.ceOutput1GetCOMM__RETURN__VALUE__1().equals("00000013") )
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}

```

5. For the update() method, the code should look like the following:

```

public void update() throws com.ibm.ipaa.IDataKeyNotFoundException {
    BeCashAcctCommand bec = new BeCashAcctCommand();
    bec.setConnectionSpec(this.getConnectionSpec());
    bec.setCeInputCOMM__REQUEST__TYPE((short)3);
    bec.setCeInputCOMM__ACCOUNTID(this.getAccount_ID());
    bec.setCeInputCOMM__RES__TYPE(this.getRes_type());
    bec.execute();

    if (bec.ceOutput1GetCOMM__RETURN__VALUE__1().equals("00000013") )
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}

```

6. For the del() method, the code should look like the following:

```

public void del() throws com.ibm.ipaa.IDataKeyNotFoundException {
    BeCashAcctCommand bec = new BeCashAcctCommand();
    bec.setConnectionSpec(this.getConnectionSpec());
    bec.setCeInputCOMM__REQUEST__TYPE((short)4);
    bec.setCeInputCOMM__ACCOUNTID(this.getAccount_ID());
    bec.setCeInputCOMM__RES__TYPE(this.getRes_type());
    bec.execute();
    if (bec.ceOutput1GetCOMM__RETURN__VALUE__1().equals("00000013") )
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}

```

## Modifying the Procedural Adapter Object to connect to the CICS Server

1. Select the **BeCashAcctPAO** class and expand it.
2. Select the **BeCashAcctPAO** constructor
3. The constructor should look like the following:

```

public BeCashAcctPAO() {
    com.ibm.ivj.communications.ECIConnectionSpec cs =
        new com.ibm.ivj.communications.ECIConnectionSpec();
    cs.setCICSServer("HOTBOSS");
    cs.setURL("jgateServerSystem.myCompany.com"); // system running jgate
    this.setConnectionSpec(cs);
}

```

## Creating an Executable Class

To create an executable class, complete the following steps:

1. Select your package.
2. From the **Selected** menu, select **Add** → **Class**, a wizard will appear to request all the necessary information required to create a class.
  - a. Enter the current project and package in the appropriate fields in the wizard and ensure that the **Create a new class** radio button is selected.
  - b. In the **Class name** field enter Execute.
  - c. Set the **Superclass** to java.lang.Object.
  - d. Ensure that the **Compose the class visually** radio button is NOT selected, and click the **Next** button to continue to the next screen.
3. There are three classes that should be imported when the executable is run. To include these classes as import statements, select the **Add Package** button. A list of available packages will appear. From the list, select each of the following and click **Add** to include them in the import statements: (after adding the last one, select Close).
  - com.ibm.connector.cics
  - com.ibm.connector.infrastructure
  - com.ibm.connector.infrastructure.java
4. Finally, ensure that the following fields are selected (checkmark beside them)
  - public (in modifiers section)
  - Methods that must be implemented (Recommended)
  - Copy constructors from superclass (Recommended)
  - main(String[])
5. To generate the class, press the **Finish** button and the class will appear inside the package you have specified.
6. Enter the code listed below into the main(String[]) method created in the Execute class, and select **Save** from the **Edit** pull-down menu.

**Note:** The User and Password for the CICS server must be inserted into this code where CBUSER appears:

```

public static void main(java.lang.String[] args) {
    try {
        JavaRuntimeContext runtimeContext = new JavaRuntimeContext();
        ((DefaultLogonInfo) runtimeContext.getLogonInfo()).setUser("CBUSER");
        ((DefaultLogonInfo) runtimeContext.getLogonInfo()).setPassword("CBUSER");
        JavaRuntimeContext.setCurrent(runtimeContext);
        ((JavaRASService) runtimeContext.getRASService()).setTraceLevel(1);
        com.ibm.ivj.communications.Session.startSession();
        BeCashAcctPAOKey key = new BeCashAcctPAOKey();
        key.setAccount_ID("00000990");
    }
}

```

```

key.setRes_type("01");
BeCashAcctPAO bec = (BeCashAcctPAO) BeCashAcctPAO.find(key);

// Retrieve Cash Account information
System.out.println("Retrieving ...");
try {
    bec.retrieve();
} catch (Exception e) {
    System.out.println("\n!!! Exception from Cash Account Retrieve " + e.toString());
    e.printStackTrace();
}
System.out.println("\n\n" + bec.toString());
BeCashAcctPAOKey key1 = new BeCashAcctPAOKey();
key1.setAccount_ID("00000995");
key1.setRes_type("01");
BeCashAcctPAO bec1 = (BeCashAcctPAO) BeCashAcctPAO.find(key1);
bec1.setUtilities("This is the utilities");
bec1.setType("2");
bec1.setBalance(100);

// Create a Cash Account
System.out.println("Creating ..." + bec1.toString());
try {
    bec1.insert();
} catch (Exception e) {
    System.out.println("\n!!! Exception from Cash Account Insert " + e.toString());
}
System.out.println("\n\n" + bec1.toString());

// Update a Cash Account
System.out.println("Updating ...");
bec1.setUtilities("Changed Utilities");
bec1.setType("B");
bec1.setBalance(300);
try {
    bec1.update();
} catch (Exception e) {
    System.out.println("\n!!! Exception from Cash Account Update " + e.toString());
    e.printStackTrace();
}
System.out.println("\n\n" + bec1.toString());

// Delete a Cash Account
System.out.println("Deleting ...");
try {
    bec1.del();
} catch (Exception e) {
    System.out.println("\n!!! Exception from Cash Account Delete " + e.toString());
    e.printStackTrace();
}
key = new BeCashAcctPAOKey();
key.setAccount_ID("00000995");
key.setRes_type("01");

// Retrieve Cash Account information
System.out.println("Retrieving ...");

```

```

        try {
            bec.retrieve();
        } catch (Exception e) {
            System.out.println("\n!!! Expected Exception from Cash
Account Retrieve " + e.toString());
        }
        com.ibm.ivj.communications.Session.endSession(true);
        System.out.println("\nSession ended");
        runtimeContext.close();

    } catch (Exception e) {
        e.printStackTrace();
        System.out.println("Error is " + e);
    }
}

```

## Running the Customer Command Application

To run the application, select the **Execute** class, click the right mouse button and select **Properties**. Select the **class Path** tab. Select the **Compute Now** button to update the Class Path and click **OK** when processing is complete. Click the **run** button (the button with a running person on it found at the top of the workbench). The results will be displayed in the console window.

## Exporting the BeCashAcct Package

After building the Execute class and creating and testing the Component Broker procedural adaptor object within the VisualAge for Java environment, you can run the unit test program outside of the VisualAge for Java environment. This object needs to be imported to Object Builder as a persistent object. Importing this object requires that the procedural adaptor object and its corresponding BeanInfo class is exported outside of VisualAge for Java. To run the sample outside of the VisualAge for Java environment, you must export all classes you created, and modify the CLASSPATH environment variable.

For ease, export the entire package. This package should contain:

- The new procedural adapter object
- Its corresponding BeanInfo class
- All EAB transaction objects

To export the package outside of VisualAge for Java:

1. Select the package to export.
2. From the **VisualAge for Java Workbench** menu, select **File** → **Export**. This opens the Export - wizard. You are going to export all .java and .class files.
3. Select the **Directory** radio button.
4. Click the **Next** button.
5. Type in the directory in which you want to export the classes, for example:

WIN x:\MyProj (where x: represents the drive of your choice)

AIX \$HOME/MyProj This will be your Working Directory for the remainder of this sample.

6. Select the **.java and .class** check boxes.
7. Click the **Finish** button.

When the export completes, the paa\mysamples\cics\eci\acct directory is created under the MyProj directory.

To verify that the package exported correctly, run the unit test program from the command line:

1. Ensure that the directory in which you exported the package, your Working Directory, is in your CLASSPATH.
2. From a command prompt, type one of the following:

```
WIN java -nojit paa.mysamples.cics.eci.acct.Execute
```

```
AIX java paa.mysamples.cics.eci.acct.Execute
```

You should have the same results as when running inside VisualAge for Java.

---

## Developing a CICS-ECI Business Object

This section contains Object Builder and System Management procedures required to create a component named CashAcct. To create this component, perform the procedures in the following sections.

1. "Importing the Bean"
2. "Defining the CashAcct Component" on page 160
3. "Creating Client and Server DLL Files" on page 164
4. "Packaging the Application" on page 165
5. "Building the Application - Client and Server" on page 167
6. "Installing the Application" on page 167
7. "Running the Sample Application" on page 170

### Notes:

1. Before starting Object Builder, ensure that your classpath includes your Working Directory.
2. Specify your Working Directory as the base directory for the project.
3. The procedures contained in this section assume that you have correctly set your classpath to include your Working Directory before starting Object Builder and that you have started Object Builder.

```
AIX To start Object Builder on AIX, type ob on the command line.
```

## Importing the Bean

The bean to import is BeCashAcct from the paa.mysamples.cics.menu.acct package in your Working Directory.

To import this bean:

1. Select the User-Defined PA Schemas folder from the **Object Builder Tasks and Objects** pane.
2. Open the pop-up menu for **User-Defined PA Schemas** and select **Import Bean**. This opens the Import Procedural Adaptor Bean - Bean Selection page wizard.
3. On this page:
  - a. Type paa.mysamples.cics.eci.acct.BeCashAcctPA0 in the **bean name** field.
  - b. Click the **Next** button to accept the remaining defaults and to continue to the Import Procedural Adaptor Bean → Names and Services page.
4. Click the **ECI** radio button in the **Connector Type** box and click the **Next** button to continue to the Import Procedural Adaptor Bean Key Selection page.

5. On this page:
  - a. Select the `res_type` and the `account_ID` properties from the **Properties** list box.
  - b. Click the `>>` button to move these associated keys required to import the bean.
6. Click the **Finish** button.

The bean is imported into Object Builder. The `BeCashAcctPAO` schema and its corresponding persistent object (`BeCashAcctPAOPO`) are now in the tree view of User-Defined PA Schemas.

## Defining the CashAcct Component

This exercise defines the objects required to create a component named "CashAcct". For this component you will:

1. Create a new business object file
2. Define the business object
3. Connect the data object implementation to the persistent object
4. Define the managed object
5. Generate the code

### Creating the Business Object File

To create the `CashAcct` business object file:

1. Select the User-Defined Business Objects folder.
2. Open the pop-up menu for **User-Defined Business Objects**, and select **Add File**. This displays the Name page of the Business Object File - wizard.
3. On this page:
  - a. Type `CashAcct` in the **Name** field.
  - b. Accept the other defaults.
4. Click **Finish**.

The `CashAcct` file is now under the User-Defined Business Objects folder.

### Defining the Business Object

After creating the new business object file, the business object needs to be defined. A fully-configured business object consists of the following:

- A business object interface
- An associated key
- An associated copy helper
- A business object implementation and data object interface

**Defining the Business Object Interface:** To create the `CashAcct` business object interface:

1. From the User-Defined Business Object folder, select **CashAcct**.
2. Open the pop-up menu for **CashAcct**, and select **Add Interface**. This displays the Name page of the Business Object Interface - wizard.
3. On this page:
  - a. Type `CashAcct` in the **Name** field.
  - b. Click **Next** to continue to the Constructs page.
4. Click **Next** to accept the defaults and to continue to the Interface Inheritance page.

5. Click **Next** to accept the defaults and to continue to the Attributes page.
6. Define the user-defined attributes.
  - a. Select **Attributes** from the tree view.
  - b. Open the pop-up menu for **Attributes**, and select **Add**. This displays the Add dialog.
  - c. In this dialog:
    - 1) Type `res_type` in the **Attribute Name** field.
    - 2) Select **string** as the **Type**. This displays the **Size** field.
    - 3) Type 0 in the **Size** field
    - 4) Click **Add Another**.
    - 5) Repeat this step for each attribute of the **CashAcct** interface in `BeCashAcctPAO.java`, but click **Refresh** instead of **Add Another** at the end of the last step.
 

**Note:** For `balance`, use type `long`. For `account_ID`, `acct_type`, and `utilities`, use `string`.
  - d. Click **Finish**.

The CashAcct interface is now under the CashAcct file.

**Defining the Key:** To add the key:

1. From the User-Defined Business Object folder, open the CashAcct interface.
2. Open the pop-up menu of **CashAcct**, and select **Add Key**. This displays the Key - Name and Key Attributes page.
3. Select the `res_type` and the `account_ID` attributes from the **Business Object Attributes** list.
4. Click **>>** to move the attribute to the **Key Attributes** list.
5. Click **Finish**.

The CashAcctKey key is now under the CashAcct interface.

**Defining the Copy Helper:** To add the copy helper:

1. From the User-Defined Business Object folder, open the CashAcct interface.
2. Open the pop-up menu for **CashAcct**, and select **Add Copy Helper**. This displays the Copy Helper - Name and Attributes page.
3. Click the **All>>** button to move the attributes from the **Business Object Attributes** list to the **Copy Helper Attributes** list.
4. Click the **Finish** button.

The CashAcctCopy copy helper is now under the CashAcct interface.

**Defining the Business Object Implementation and Data Object Interface:** To add the Business Object Implementation and Data Object interface:

1. From the User-Defined Business Object folder, open the CashAcct interface.
2. Open the pop-up menu for **CashAcct**, and select **Add Implementation**. This displays the Name and Data Access Pattern page of the Business Object Implementation - wizard.
3. Type `CashAcctB0` in the **File Name** field.
4. Define the implementation.
  - a. Select the **Delegating** radio button from the **Pattern for Handling State Data** group.

- b. Ensure that the **Create a new one now** radio button is selected from the **Data Object Interface** group. This option allows you to define the business object attributes that need to be preserved in the data object.
- c. Deselect **390** in the Select Deployment platform group box.
- d. Click the **Next** button to continue to the Implementation Inheritance page.
5. Click the **Next** button to accept the defaults and to continue to the Implementation Language page.
6. Click the **Next** button to accept the defaults and to continue to the Attributes page.
7. Click the **Next** button to accept the defaults and to continue to the Methods page.
8. Click the **Next** button to accept the defaults and to continue to the Key and Copy Helper page.
9. On this page:
  - a. Verify that the **CashAcctKey** key is selected from the **Key** list.
  - b. Verify that the **CashAcctCopy** copy helper is selected from the **CopyHelper** list.
  - c. Click the **Next** button to continue to the Handle Selection page.
10. Click the **Next** button to accept the defaults and to continue to the Attributes to Override page.
11. Click the **Next** button to accept the defaults and to continue to the Methods to Override page.
12. Click the **Next** button to accept the defaults and to continue to the Data Objects Interface page.
13. Click the **All>>** button to move the attributes in the **Business Object Attributes** list to the **State Data** list.
14. Click the **Finish** button.

The CashAcctBO business object implementation is now under the CashAcct interface, and the CashAcctDO data object interface is now under the CashAcctBO business object implementation.

## Connecting the Data Object Implementation to the Persistent Object

To create the data object implementation and to connect the data object implementation to the persistent object:

1. From the User-Defined Business Object folder, select the CashAcctDO data object interface.
2. Open the pop-up menu for **CashAcctDO**, and select **Add Implementation**. This displays the Data Object Implementation - Name and Platform page.
3. Deselect **390** in the Select Deployment platform group box.
4. Click the **Next** button to continue to the Behavior page.
5. On this page:
  - a. Set the **BOIM with any key** radio button from the **Environment** group box to indicate that the data object is part of a component installed in a business object application adaptor with instances being located by key objects.
  - b. Set the **Procedural Adaptors** radio button from the **Form of Persistent Behavior and Implementation** group box.
  - c. Click the **Next** button to continue to the Implementation Inheritance page.
6. On this page:
  - a. Verify that **IPAAExtLocalToServer::IDataObject** is selected as a parent.
  - b. Click **Next** to continue to the Attributes page.
7. Click **Next** to accept the defaults and to continue to the Methods page.

8. Click **Next** to accept the defaults and to continue to the Key and Copy Helper page.
9. Click **Next** button to accept the defaults and to continue to the Associated Persistent Objects page.
10. On this page:
  - a. Select Persistent Object Instances.
  - b. Open the pop-up menu for Persistent Object Instance, and select **Add**.
  - c. Type `iBeCashAcctPAOPO` in the **Instance Name** field.
  - d. Click **Next** to continue to the Attributes Mapping page.
11. On this page:
  - a. Select `res_type` from the **Attributes** list.
  - b. Open the pop-up menu for **res\_type**, and select **Primitive**.
  - c. Select `iBeCashAcctPAOPO.res_type` from the **Persistent Object Attribute** list.
  - d. Add 1-to-1 mappings for the other attributes in the **Attributes** tree view as you did for `res_type` (`acct_type` will map to `type`).
  - e. Click **Next** to continue to the Methods Mapping page.
12. On this page:
  - a. Select `insert` from the **Special Framework Methods** list.
  - b. Open the pop-up menu for **insert**, and select **Add mapping**.
  - c. Select `BeCashAcctPAOPO.insert` from the **Persistent Object Method** list.
  - d. Add 1-to-1 mappings for the remaining CRUD methods and for the `setConnection()` method in the **Special Framework Methods** tree view as you did for `insert`.
13. Click the **Finish** button.

The `CashAcctDOImpl` data object implementation is now under the `CashAcctDO` interface, and the `BeCashAcctPAOPO` persistent object is now under the `CashAcctDOImpl` data object implementation.

## Defining the Managed Object

To add the managed object:

1. From the User-Defined Business Object folder, select the `CashAcctBO` business object implementation.
2. Open the pop-up menu for `CashAcctBO`, and select **Add Managed Object**. This displays the Name and Services page of the Managed Object - wizard.
3. Type `CashAcctM0` in the **File Name** field.
4. Deselect **390** in the Select Deployment platform group box.
5. Set the **Session Service** radio button.
6. Click the **Next** button to continue to the Implementation Inheritance page.
7. Click the **Finish** button.

## Generating the Code

To generate the application code:

1. From the User-Defined Business Object folder, select `CashAcct`.
2. Open the pop-up menu for **CashAcct**, and select **Generate** → **All**.

Code generation starts. Progress is indicated in the lower-left corner of the window.

## Creating Client and Server DLL Files

The defined objects need to be built into two separate DLL files.

- One that runs on the client and provides access to the business object interface, key, and copy helper.
- One that runs on the server and provides access to the managed object and the rest of the component.

The client DLL file must be defined before the server DLL file. When the server DLL file is defined, it must link to the client DLL file. After defining the objects that comprise each DLL file, these files can be built.

### Defining the Client DLL File

To add the client DLL file:

1. Select the Build Configuration folder.
2. Open the pop-up menu for **Build Configuration**, and select **Add Client DLL**. This displays the Name and Options page of the Add Client DLL - wizard.
3. Type CashAcctC in the **Name** field.
4. Check only the **Applicable Platforms** you want.
5. Click **Next** to continue to the Client Source Files page.
6. Click the **All>>** button to move the client source files to the **Items Chosen** list.
7. Click **Finish**.

The CashAcctC client DLL file is now under the Build Configuration folder.

### Defining the Server DLL File

To add the server DLL.

1. Select the Build Configuration folder.
2. Open the pop-up menu for **Build Configuration**, and select **Add Server DLL**. This displays the Name and Options page of the Server DLL - wizard.
3. Type CashAcctS in the **Name** field.
4. Check only the applicable platforms you want.
5. Click the **Next** button to continue to the Server Source Files page.
6. Click the **All>>** button to move the server source files to the **Items chosen** list.
7. Click the **Next** button to continue to the Libraries to Link With page.
8. Select CashAcctC from the **Items Available** list.
9. Click the **>>** button to move CashAcctC.dll to the **Items Chosen** list.
10. Click the **Finish** button.

The CashAcctS server DLL file is now under the Build Configuration folder.

## Generating the Makefiles

To generate the makefiles to build the configuration:

1. Select the Build Configuration folder.
2. Open the pop-up menu for **Build Configuration**, and select **Generate** → **All** → **All Targets**.

The code generation begins.

## Packaging the Application

Packaging the application consists of the following procedures:

1. Creating the application family
2. Defining the application
3. Creating the container instance
4. Configuring the managed object
5. Generating the application

### Creating the Application Family

To add the application family:

1. Select the Application Configuration folder.
2. Open the pop-up menu for **Application Configuration**, and select **Add Application Family**. This displays the Name page of the Add Application Family - wizard.
3. Type CashAcctApp in the **Name** field.
4. Click the **Finish** button.

The CashAcctApp application family is now under the Application Configuration folder.

### Defining the Application

To add the Application:

1. Select the CashAcctApp application family.
2. Open the pop-up menu for **CashAcctApp**, and select **Add Application**. This displays the Name and Environment page of the Add Application - wizard.
3. Type CashAcct in the **Application Name** field.
4. Click the **Finish** button.

The CashAcct application is now under the CashAcctApp application family.

### Creating the Container Instance

To add the new container instance:

1. Select the Container Definition folder.
2. Open the pop-up menu for **Container Definition**, and select **Add Container Instance**. This displays the Container - wizard.
3. Type CashAcctContainer in the **Name** field.
4. Deselect **390** on the Select deployment platform group box.
5. Click **Next** to continue to the Workload Management page.

6. Click **Next** to continue to the Service page.
7. On the Service page, set the **Use PAA Session Service** radio button.
8. Click **Next** to continue to the Service Details page.
9. On this page:
  - a. Type CICS\_CashAcct\_Server in the **Connection Name** field.
  - b. Set the **ECI Connection** radio button under the **Connector Type** used by a Session group box.
10. Click the **Finish** button.

The CashAcctContainer container is now under the Container Definition folder.

## Configuring the Managed Object

To add the managed object for the Application:

1. Open CashAcctApp under the Application Configuration folder.
2. Select the CashAcct application.
3. Open the pop-up menu for **CashAcct**, and select **Add Managed Object**. This displays the Configure Managed Object - wizard.
4. In this window:
  - a. Verify that CashAcctMO is in the **Managed Object** field.
  - b. Click the **Next** button to continue to the Data Object Implementations page.
5. On this page:
  - a. Select Implementations.
  - b. Open the pop-up menu for Implementations, and select **Add**.
  - c. Select CashAcctDOImpl from the **Data Object Implementation** list.
  - d. Click the **Next** button to continue to the Container page.
6. On this page, select CashAcctContainer from the **Name** list.
7. Click the **Next** button to continue to the Home page.
8. On this page, select BOIMHomeOfRegHomes from the **Home Name** list.
9. Click the **Finish** button.

The CashAcctMO managed object is now under the Application Configuration folder.

## Generating the Applications

To generate the application family:

1. Select the CashAcctApp application under the Application Configuration folder.
2. Open the pop-up menu for CashAcctApp, and select **Generate**.

**Note:**  On Windows NT, if you do not have InstallShield installed on your system, click the **Yes** button when the dialog concerning InstallShield is displayed.

When code generation completes, the Method Implementation pane contains the CashAcctApp.ddl file. You can now close Object Builder.

## Building the Application - Client and Server

All imported and generated files are placed in one of the following subdirectories of your Working Directory:

**WIN** Working\NT.

1. Change the directory to:

```
x:\MyProj\Working\NT
```

2. Enter:

```
nmake -f all.mak cpp java
```

**AIX** Working/AIX.

1. Change the directory to:

```
$HOME/MyProj/Working/AIX
```

2. Enter

```
make -f all.mak cpp java
```

Everything in the sample application is built.

## Installing the Application

Installing an application consists of:

1. Loading the application
2. Configuring the application

These procedures assume that you are currently logged on to DCE and that you are currently using the System Manager User Interface. If not, logon to DCE and start the System Manager User Interface.

## Loading the Application onto System Management

To install the CashAcct server application:

1. Start the System Manager User Interface if it is not already started.
2. Become an Expert user (**View** → **User Level** → **Expert**).
3. Expand Host Images, and select <your host name>.
4. From the pop-menu, select **Load Application**. This opens the Load Application dialog.
5. Browse for and select **CashAcctApp.ddl**.

**WIN** x:\MyProj\Working\NT\CashAcctApp\CashAcctApp.ddl

**AIX** \$HOME/MyProj/Working/AIX/CashAcctApp/CashAcctApp.ddl

## Configuring the Application with System Management

To configure the application:

1. Configure the application.
  - a. Expand Available Applications, and select **CashAcct**.
  - b. Open the pop-up menu for **CashAcct**, and select **Drag**.

- c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select **Sample Configuration**.
  - d. Open the pop-up menu for **Sample Configuration**, and select **Add Application**.
2. Configure the ECI connection.
    - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → ECI Connections, and select **CICS\_CashAcct\_Server**.
    - b. Open the pop-up menu for **CICS\_CashAcct\_Server**, and select **Edit**, which opens the Object Editor.
    - c. Click the **Main** tab.
    - d. Change the **CICS Server name** field to match your CICS region with which you are communicating as specified in your cicscli.ini file.
    - e. Under gateway address:
      - If the Transaction Gateway is local (installed on the same computer), leave the default (local:)
      - If the network Transaction Gateway is required, type one of the following addresses:
 

```
tcp://my.cics.gateway/
OR
tcp://my.other.gateway:8080/
```

Where *my.cics.gateway* represents the name used for the default port number and *my.other.gateway* represents the name used with port 8080. The address format is defined by the Transaction Gateway.
    - f. Click the **OK** button to validate and accept the changes.
  3. Define the server.
    - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select **Sample Configuration**.
    - b. Open the pop-up menu for **Sample Configuration**, and select **New** → **Server (free standing)**. This displays a new dialog box.
    - c. Type CashAcctSvr as the name for the server.
    - d. Click the **OK** button. The CashAcctSvr server is now under Server (free standing).
  4. Associate the application with the server.
    - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Applications, and select **CashAcct**.
    - b. Open the pop-up menu for **CashAcct**, and select **Drag**.
    - c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Server (free standing), and select **CashAcctSvr**.
    - d. Open the pop-up menu for **CashAcctSvr**, and select **Configure Application**.
  5. Associate the iPAAServices with the server.
    - a. Host Images → *myhost* → Application Family Installs → iPAAApplications → Application Installs, and select **iPAAServices**.
    - b. Open the pop-up menu for **iPAAServices**, and select **Drag**.
    - c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select **Sample Configuration**.

- d. Open the pop-up menu for **Sample Configuration**, and select **Add Application**.
  - e. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Applications, and select **iPAAServices**.
  - f. Open the pop-up menu for **iPAAServices**, and select **Drag**.
  - g. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Server (free standing), and select **CashAcctSvr**.
  - h. Open the pop-up menu for **CashAcctSvr**, and select **Configure Application**.
6. Configure the server with the host.
- a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Server (free standing), and select **CashAcctSrv**.
  - b. From the pop-up menu for **CashAcctSrv**, select **Drag**.
  - c. Expand Hosts and select your server.
  - d. From the pop-up menu for your server, select **Configure Server (free standing)**.
7. **Optional:** Enable security services for the server.
- a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Servers (free standing), and select **CashAcctSrv**.
  - b. Open the pop-up menu for **CashAcctSrv**, and select **Edit**, which opens the Object Editor.
  - c. In this notebook:
    - 1) Select the **Security Service** tab.
    - 2) Change the value for the **data system principal** field to the user ID that the server will use when connecting to the CICS system.
    - 3) Change the value for the **data system password** field to the password that the server will use when connecting to the CICS system.
    - 4) Change the value for the **security enabled** field from `no` to `yes`.
    - 5) Click the **OK** button. The changes are applied and the Object Editor closes.
8. **Optional:** Enable security services for the client.
- a. Expand Management Zones → Sample Cell and Work Group Zone → Configuration → Sample Configuration → Client Styles, and select **myClient**.
  - b. Open the pop-up menu for **myClient**, and select **Edit**, which opens the Object Editor.
  - c. In this notebook:
    - 1) Select the **Security Service** tab.
    - 2) Change the value for the **security enabled** field from `no` to `yes`.
    - 3) Click the **OK** button. The changes are applied and the Object Editor closes.
9. Activate the configuration.
- a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select **Sample Configuration**.
  - b. Open the pop-up menu for **Sample Configuration**, and select **Activate**, which automatically starts the application server. Wait for the completion message in the **Action Console** window before continuing.

## Running the Sample Application

Before running this sample, ensure that the IBM Transaction Server for CICS/NT is configured for the CICS region and a single Encina shared file system (SFS). For details, see Appendix A, "Installing the IVPs and CICS-ECI Sample" on page 219.

To run the sample client application, complete one of the following procedures:

### WIN

1. Copy CashAcctcli.mak and CashAcctcli.cpp from:  
x:\CBroker\samples\InstallVerification\PAA\Application\CashAcctcli  
to  
x:\MyProj\Working\NT
2. Change directory to:  
x:\MyProj\Working\NT
3. Enter:  
nmake -f CashAcctcli.ma
4. Enter:  
cashacctcli

### AIX

1. Copy CashAcctcli.mak and CashAcctcli.cpp from:  
\$HOME/samples/PAAsamples/Application/CashAcctcli  
to  
\$HOME/MyProj/Working/AIX
2. Change directory to:  
\$HOME/MyProj/Working/AIX
3. Enter:  
make -f CashAcctcli.mak
4. Enter:  
cashacctcli

---

## Chapter 8. Developing an IMS-APPC Application

This chapter provides information for building a sample Component Broker application with an IMS backend.

This chapter contains the following information.

- “The IMS Sample Application”
- “Enterprise Access Builder Procedures”
- “Developing an IMS-APPC Business Object” on page 183

**Note:** To walk-through this sample, the following software and Component Broker software must be installed on your system:

- The Component Broker samples
- The CICS and IMS Application Adaptor SDK
- IBM VisualAge Java with EAB

**Important Information:** Before walking through this sample, please refer to the *Late Breaking News* provided with Component Broker before performing the exercise in this chapter. This document provides the latest information regarding the CICS and IMS application adaptor samples, that may differ from the instructions for this sample application.

---

### The IMS Sample Application

The IMS-APPC sample application is based on an IMS Installation Verification Procedure (IVP). The IVP is a mock phone book database, where each entry in the phone book contains the following fields:

- Last name
- First name
- Phone number extension
- Internal zip code

This sample application works on an IMS database and permits adding, inquiring, updating, and deleting of phone book entry records through the ADD, DISPLAY, UPDATE, and DELETE transactions.

The sample that you build in this section is included with the product and can be built by following the steps in the HTML file in one of the following:

WIN `CBroker\samples\InstallVerification\PAA\readme.htm`

AIX `$HOME/samples/InstallVerification/PAA/readme.htm`

---

### Enterprise Access Builder Procedures

An overview of the steps is given below:

1. “Importing Prerequisite Features into the Workspace” on page 172
2. “Creating a Project/Package under VisualAge for Java” on page 172
3. “Creating the Procedural Adaptor Object and Key” on page 173
4. “Importing the PhoneBook COBOL File” on page 175
5. “Creating the Record Mapper” on page 176
6. “Creating the Command Classes” on page 178

7. “Modifying the Procedural Adapter Object to call the Commands” on page 181
8. “Developing an IMS-APPC Business Object” on page 183
9. “Running the Sample Application” on page 194

**WIN** If you are using VisualAge for Java on Windows 95 or Windows NT, from the **Start** menu, select **Programs** → **IBM VisualAge for Java for Windows** → **IBM VisualAge for Java**.

**AIX** If you are using Visual Age for Java on AIX, type `vajide` on the command line.

If the VisualAge Quick Start dialog appears, select **Go to the Workbench** and click **OK**. The IDE appears.

From the Window pulldown, select **Options**. Select **Design Time** and un-check **Inherit BeanInfo** of bean superclass. Click **OK**.

## Importing Prerequisite Features into the Workspace

1. Select **File** → **Quick Start**.
2. Select **Features** in the left pane and **Add Feature** in the right pane.
3. Click **OK**.
4. Select the following:
  - IBM Procedural Application Adapter 1.0
  - CICS Connector 3.0
  - IBM Component Broker Connectors 1.0
  - IBM Enterprise Access Builder Library 2.0
  - IBM Component Broker PAA Samples for IMS 1.0
5. Click **OK**.

You may ignore the following expected errors this introduces in the following packages:

- com.ibm.ivj.communications
- com.ibm.ivj.trace
- com.ibm.eNetwork.ECL
- com.ibm.eNetwork.ncod.services.RAS

**Note:** If you do not see all of these features listed, they have been previously installed. To confirm, perform the following steps:

- a. Select **File** → **Quick Start**.
- b. Select **Features - Delete Feature** and see which features are already loaded (then Cancel).

## Creating a Project/Package under VisualAge for Java

1. From the VisualAge for Java list of projects, select **IBM Component Broker PAA Samples for IMS**.
2. Open the pop-up menu of **IBM Component Broker PAA Samples for IMS**, and select **Add Package**. This creates a package for the project.
3. Type `paa.mysamples.ims.appc.pbe` for the new package, and click the **Finish** button.

**Note:** If you are using the default mouse configuration, right-click on the denoted item. You do not have to select the item before opening its menu. You can select the item and open its menu with a right-click.

## Creating the Procedural Adaptor Object and Key

The procedural adaptor object inherits from `com.ibm.ivj.eab.paa.EntityProceduralAdapterObject`, which serves as a base implementation for all procedural adaptor objects. As a subclass of `EntityProceduralAdapterObject`, the procedural adaptor object contains the CRUD methods (create (or insert), retrieve, update, and delete). However, these methods are all empty-bodied. You must define their implementation for your procedural adaptor object.

The attributes defined in the `PhoneBookEntry` interface are essential. Thus, the procedural adaptor object, as the adaptor that connects the Component Broker data object to the backend system, should contain the properties that correspond to these attributes.

1. From the VisualAge for Java desktop under the IBM Component Broker PAA Samples for IMS project, select **paa.mysample.ims.appc.pbe**.
2. Open the pop-up menu for **paa.mysample.ims.appc.pbe**, and select **Add > Class**.
3. In this dialog:
  - a. Type `APhoneBookPAO` in the **Class name** field.
  - b. Click the **Browse** button to select the Superclass:
    - 1) Browse for and select **EntityProceduralAdapterObject** as your Superclass.
    - 2) Click the **OK** button to close the dialog.
4. Click the **Finish** button.

Add the properties for the **APhoneBookPAO** interface:

1. Select the **APhoneBookPAO** interface.
2. Open the pop-up menu for **APhoneBookPAO**, and select **Open**, which opens the Object Editor notebook.
3. In this notebook:
  - a. Select the **BeanInfo** tab.
  - b. From the menu bar, select **Feature** → **New Property Feature**, which opens the wizard - New Property Feature.
  - c. In this wizard type the name of the new property in the **Property name** field. For simplicity, use the same name as used in the `PhoneBookRec` interface. For example, use:

```
LastName  
firstName  
extNumber  
internalZip
```

for the properties as defined in the `pbe.cpp` file. Each of these properties must be defined individually. For this step (first time) type `LastName`. For each subsequent time, type `firstName`, `extNumber`, `internalZip`, respectively.

- 1) For all properties, select **java.lang.String** from the pull down menu of the **Property type** field.
- 2) Accept the other defaults and click the **Next** button.
- 3) Click the **Finish** button.

Define the other properties (**Features** → **New Property Feature**)

4. Close the Object Editor Notebook.

Now the Key for this PAO object must be created:

1. From the VisualAge for Java desktop under the IBM Component Broker PAA Samples for IMS project, select **paa.mysample.ims.appc.pbe**.
2. Open the pop-up menu for **paa.mysample.ims.appc.pbe**, and select **Add** → **Class**.

3. In this dialog:
  - a. Type APhoneBookPAOKey in the **Class name** field.
  - b. Click the **Browse** button to select the Superclass:
    - 1) Browse for and select **BusinessObjectKey** as your Superclass.
    - 2) Click the **OK** button to close the dialog.
  - c. Click the **Finish** button.

Add the properties for the **APhoneBookPAOKey** interface:

1. Select the **APhoneBookPAOKey** interface.
2. Open the pop-up menu for **APhoneBookPAOKey**, and select **Open**, which opens the Object Editor notebook.
3. In this notebook:
  - a. Select the **BeanInfo** tab.
  - b. From the menu bar, select **Features** → **New Property Feature**, which opens the wizard - New Property Feature.
  - c. In this wizard window:
    - 1) Type the name of the new property in the **Property name** field. For example, use:  
LastName  
  
for the property that is going to be the key attribute. You can select **java.lang.String** for the type of the property.
    - 2) Accept the other defaults and click the **Next** button.
    - 3) Click the **Finish** button.
  - d. Close the object editor notebook.

Modify the APhoneBookPAOKey and APhoneBookPAO to tie the PAO and key class together.

## APhoneBookPAOKey

1. Select and expand the **APhoneBookPAOKey** class.
2. Highlight the getPropertyValues() method. This method is used by the Enterprise Access Builder (EAB) run time to calculate a value to key into the EAB cache. It needs to be modified to specifically return just the key values.
3. In the source pane, return an array of Objects that make up the key, by invoking the methods that get the key properties. For example:

```
return new Object[] { this.getLastName() };
```
4. Save the changes to the modified PAO Key class by pressing **Ctrl-S**.

## APhoneBookPAO

1. Select and expand the **APhoneBookPAO** class.
2. Modify the getter for the key property value getLastName() by getting the key class associated with this PAO and returning that value. For example:

```
APhoneBookPAOKey key = (APhoneBookPAOKey) this.getKey();  
return key.getLastName();
```
3. Save the changes to the modified PAO by pressing **Ctrl-S**.

## Importing the PhoneBook COBOL File

### Creating the Input Information Class

1. Select the package that you have created.
2. Open the pop-up menu for the package you are working under, and select **Tools** → **Records** → **Create Cobol Record Type...**, a wizard window will appear.
3. In this wizard window:
  - a. In the **Class Name** field enter APhoneBookInfoInput.
  - b. In the **COBOL File** field, browse through the files to locate the pbe.ccp file. It should be located in one of the following:
    - WIN `CBroker\samples\InstallVerification\PAA\Backend\IMSAPPC\` and select **Open**.
    - AIX `$HOME/samples/InstallVerification/PAA\Backend/IMSAPPC/` and select **Open**.
  - c. Check that the Project and Package names are correct.
  - d. Click **Next** to continue to the next screen.
4. On the next screen:
  - a. In the list of Available level 01 commareas, select **INPUT-MSG** and click → to move it to the **Selected commareas** list.
  - b. Un-check the box beside the **RecordType intended for CICS** field.
  - c. Click **Finish** button when this is complete. A new class named APhoneBookInfoInput will appear in the designated package.
5. Select the APhoneBookInfoInput class
6. Open the pop-up menu for the **APhoneBookInfoInput** class, and select **Tools** → **Records** → **Generate Records...**, the Generate Records wizard will appear.
7. In this wizard window:
  - a. In the **Class Name** field type APhoneBookRecordInput
  - b. Select the **Beans** radio button to generate the records as beans.
  - c. Select the **Direct** radio button to access the record fields directly.
  - d. Select the **Dynamic Records** radio button to generate the records as dynamic records.
  - e. Check that the Project and Package names are correct and click **Next**.
8. In the next window:
  - a. Change Endian, Remote Integer Endian, Code Page and Machine Type to the correct values for the server. (For example, the code page for North American MVS is 037 and for North American Windows NT it is 437).
    - WIN You will have to change all of the values if you are going to a Transaction Server on Windows NT. Also remember to change your endianness to littleEndian if you are running this from Windows NT.
  - b. Click **Finish** when this is complete. Two new classes will appear in your package: **APhoneBookRecordInput** and **APhoneBookRecordInputBeanInfo**.

### Creating the Output Information Class

1. Select the package that you have created.

2. Open the pop-up menu for the package you are working under, and select **Tools** → **Records** → **Create Cobol Record Type...**, a wizard window will appear.
3. In this wizard window:
  - a. In the **Class Name** field type APhoneBookInfoOutput.
  - b. In the **COBOL File** field, browse through the files to locate the pbe.ccp file. It should be located in one of the following:
    - WIN CBroker\samples\InstallVerification\PAA\Backend\IMSAPPC\ and select **Open**.
    - AIX \$HOME/samples/InstallVerification/PAA\Backend/IMSAPPC/ and select **Open**.
  - c. Check that the Project and Package names are correct.
  - d. Click **Next** to continue to the next screen.
4. On the next screen:
  - a. In the list of Available level 01 commareas, select **OUTPUT-AREA** and click → to move it to the **Selected commareas** list.
  - b. Un-check the box beside the **RecordType intended for CICS** field.
  - c. Click **Finish** when this is complete. A new class named **APhoneBookInfoOutput** will appear in the designated package.
5. Select the **APhoneBookInfoOutput** class
6. Open the pop-up menu for the **APhoneBookInfoOutput** class, and select **Tools** → **Records** → **Generate Records...**, the Generate Records wizard will appear.
7. In this wizard window:
  - a. In the **Class Name** field type APhoneBookRecordOutput
  - b. Select the **Beans** radio button to generate the records as beans.
  - c. Select the **Direct** radio button to access the record fields directly.
  - d. Select the **Dynamic Records** radio button to generate the records as dynamic records.
  - e. Check that the Project and Package names are correct and click **Next**.
8. In the next window:
  - a. Change Endian, Remote Integer Endian, Code Page and Machine Type to the correct values for the server. (The code page for North American MVS is 037 and for North American Windows NT it is 437).
    - WIN You will have to change all of the values if you are going to a Transaction Server on Windows NT. Also remember to change your endianness to littleEndian (if you are running this from Windows NT).
  - b. Click **Finish** when this is complete. Two new classes will appear in your package: **APhoneBookRecordOutput** and **APhoneBookRecordOutputBeanInfo**.

## Creating the Record Mapper

### Creating the Input Mapper Class

1. Select the sample package again that you have created and expand it.
2. Select class APhoneBookRecordInput.
3. Open the pop-up menu for the **APhoneBookRecordInput** class, and select **Tools** → **Mapper Editor...**, a Mapper wizard will appear.

4. In this window:
  - a. From the **Code Generation** pulldown, select **Set Target mapper**. A window containing three fields will appear.
  - b. In this window:
    - 1) Enter the project and package name of this sample in the first two fields.
    - 2) In the **Class** field enter `APhoneBookRecordInputMapper`.
    - 3) Click **OK** when this is complete.
5. Select **Change Input bean** from the Code Generation pulldown. A window containing one field will appear.
6. In this window:
  - a. Select **Browse**, and then type `APhoneBookRecordInput` to select the class (corresponding to the sample package) and click **OK**.
  - b. Click **OK** once more to select the Input Bean class. The following message displays:  
All of your connections will be lost. Do you want to proceed?  
Click the **Yes** button.
  - c. You will now see a list of fields available from `APhoneBookRecordInput`.
7. Click the **Add** button located at the bottom left of the wizard window and enter `APhoneBookPAO` in the **Pattern** field.
8. Select the **APhoneBookPAO** class corresponding to the package you are currently using.
9. Click **OK** when this is complete. You will now see a directory named `java.lang.Object` in the Output Beans side of the window.
10. Expand this directory until the first instance of `APhoneBookPAO` is visible (should be able to see `extNumber`, `firstName`, `lastName`, etc.)
11. Select the **extNumber** field of the `APhoneBookPAO` object. Move the cursor to the right hand side of the screen and select **IN\_\_EXTENSION**. At the bottom of the screen, select **<—>** the `extNumber` field should be connected to **IN\_\_EXTENSION** on the input side. Repeat this step to form connections between the rest of the **APhoneBookPAO** fields starting with **IN**. Select **Apply** and **OK** when this is complete. A new class called **APhoneBookRecordInputMapper** will appear in the current package.

### Creating the Output Mapper Class

1. Select the sample package again that you have created and expand it.
2. Select class `APhoneBookRecordOutput`.
3. Open the pop-up menu for the **APhoneBookRecordOutput** class, and select **Tools** → **Mapper Editor...**, a Mapper wizard will appear.
4. In this window:
  - a. From the Code Generation pulldown, select **Set Target mapper**. A window containing three fields will appear.
  - b. In this window:
    - 1) Enter the project and package name of this sample in the first two fields.
    - 2) In the **Class** field enter `APhoneBookRecordOutputMapper`.
    - 3) Click **OK** when this is complete.
5. Select **Change Input bean** from the Code Generation pulldown. A window containing one field will appear.

6. In this window:
  - a. Select Browse. Next, type APhoneBookRecordOutput to select the class (corresponding to the sample package) and click **OK**.
  - b. Click **OK** once more to select the Input Bean class. The following message displays:  
All of your connections will be lost. Do you want to proceed?  
Click the **Yes** button.
  - c. You will now see a list of fields available from APhoneBookRecordOutput.
7. Click the **Add** button located at the bottom left of the wizard window and enter APhoneBookPAO in the **Pattern** field.
8. Select the **APhoneBookPAO** class corresponding to the package you are currently using.
9. Click **OK** when this is complete. You will now see a directory named java.lang.Object in the Output Beans side of the window.
10. Expand this directory until the first instance of APhoneBookPAO is visible (should be able to see extNumber, firstName, lastname, etc.)
11. Select the **extNumber** field of the APhoneBookPAO object. Move the cursor to the right hand side of the screen and select **OUT\_\_EXTENSION**. At the bottom of the screen, select <—> **the extNumber** field should be connected to **OUT\_\_EXTENSION** on the input side. Repeat this step to form connections between the rest of the **APhoneBookPAO** fields starting with **IN**. Select **Apply** and **OK** when this is complete. A new class called **APhoneBookRecordOutputMapper** will appear in the current package.

## Creating the Command Classes

### Input Command

1. Select the package that you have been working in.
2. Open the pop-up menu of the package, and select **Add** → **Class**. A wizard window appears.
3. In this window:
  - a. Enter the project and package names of the sample application into their corresponding fields.
  - b. Select the **Create a new class** radio button and enter APhoneBookCommandInput for the Class name.
  - c. To select the Superclass, click the **Browse...** button and select **CommunicationCommand** from the list.
  - d. Click the **OK** button.
  - e. Ensure that the **Compose the class visually** radio button is NOT selected and click **Finish**.
4. Select the **APhoneBookCommandInput** class
5. From the pop-up menu for the class, select **Open To** → BeanInfo.
6. In this Dialog:
  - a. Select **Features** → **Generate BeanInfo** class. This will generate a new BeanInfo class for your command class.
  - b. Select **Features** → **Add Available Features**.
    - 1) In the **Add Available Features** dialog, select the following features that may appear:
      - class

- communication
  - connectionSpec
  - disconnectCommunication
  - expectedTriggerClass
  - input
  - interactionSpec
  - mappedObjects
  - mappingHelper
  - output
- 2) Click the **OK** button.
- c. Close the Command Class window.
- a. Open up the pop-up menu of the class, and select **Tools** → **Command Editor**. A new dialog is displayed.
  - b. In this dialog:
    - 1) Right-mouse click on the **Communication** task and select **Add InteractionSpec**. A window displays all objects that inherit from InteractionSpec.
    - 2) In this window:
      - a) Select **APPCInteractionSpec**.
      - b) Click **OK** and the window will be closed and an interactionSpec entitled **celInteractionSpec** will be added under the **Communication Task**.
    - 3) Select **celInteractionSpec**, right-mouse click on it and select **Properties**. A window is displayed that allows you to change the bean properties.
    - 4) In this window:
      - a) In the **codepage** field, type 037 (the codepage for an MVS system).
      - b) In the **intEndian** field, type 1.
      - c) In the **machineType** field, type 2.
      - d) In the **mode** field, type 0.
      - e) In the **otherEndian** field, type 0.
      - f) Click the **OK** button to close the properties window.
    - 5) Right-mouse click on the **Input** task and select **Add IByteBuffer Bean**. A window will display all beans in VAJ. Select the **RecordBean** created earlier, **AphoneBookRecordInput**, and click **OK**.
    - 6) Right-mouse click on **celInput** and select **Promote Bean Feature**. Ensure that the **Property** radio button is selected and move **IN\_LAST\_NAME**, **IN\_COMMAND** from the left pane to the right pane by highlighting the property and clicking the >> button.
    - 7) Click **OK** to generate runtime code and the bean info class.
    - 8) Right-mouse click on **celInput** and select **Add Mapper**. A window will display all mapper beans in VAJ. Select **AphoneBookRecordInputMapper** and click **OK**. A **ceMapperCelInput** object should now be created under the celInput object.
  - c. Click **OK**.
  - d. Right-mouse click on the **Output** task and select **Add IByteBuffer Bean**. A window displays all beans in VAJ. Select the **RecordBean** created earlier, **AphoneBookRecordOutput**, and select
  - e. Right-mouse click on **ceOutput1** and select the **Promote Bean Feature**. A window is displayed.
  - f. Ensure that the **Property** radio button is selected and highlight **OUT\_\_MESSAGE** and move it over. (This is key field).

- g. Click **OK**.

### Output Command

1. Select the package that you have been working in.
2. Open the pop-up menu of the package, and select **Add > Class**. A wizard window appears.
3. In this window:
  - a. Type the project and package names of the sample application into their corresponding fields.
  - b. Select the **Create a new class** radio button and enter `APhoneBookCommandOutput` for the Class name.
  - c. To select the Superclass, click the **Browse...** button and select **CommunicationCommand** from the list.
  - d. Click **OK**.
  - e. Ensure that the **Compose the class visually** radio button is NOT selected and click **Finish**.
  - f. Select the **APhoneBookCommandOutput** class
  - g. From the pop-up menu for the class, select **Open To → BeanInfo**.
  - h. In this Dialog:
    - 1) Select **Features → Generate BeanInfo** class. This will generate a new BeanInfo class for your command class.
    - 2) Select **Features → Add Available Features**.
      - a) In the **Add Available Features** dialog, select the following features that may appear:
        - class
        - communication
        - connectionSpec
        - disconnectCommunication
        - expectedTriggerClass
        - input
        - interactionSpec
        - mappedObjects
        - mappingHelper
        - output
      - b) Click the **OK** button.
  - i. Close the Command Class window.
4. Open up the pop-up menu of the class, and select **Tools → Command Editor**. A new dialog is displayed.
5. In this dialog:
  - a. Right-mouse click on the **Communication** task and select **Add InteractionSpec**. A window displays all objects that inherit from InteractionSpec.
  - b. In this window:
    - 1) Select **APPCInteractionSpec**.
    - 2) Click **OK** and the window will close and an interactionSpec entitled **celInteractionSpec** will be added under the Communication Task.
  - c. Select **celInteractionSpec**, right-mouse click on it and select **Properties**. A window is displayed that allows you to change the bean properties.

- d. In this window:
  - 1) In the **codepage** field, type 037 (the codepage for an MVS system).
  - 2) In the **intEndian** field, type 1.
  - 3) In the **machineType** field, type 2.
  - 4) In the **mode** field, type 0.
  - 5) In the **otherEndian** field, type 0.
  - 6) Click **OK** to close the properties window.
6. Right-mouse click on the **Input** task and select **Add IByteBuffer Bean**. A window will display all beans in VAJ. Select the **RecordBean** created earlier, **APhoneBookRecordInput**, and click **OK**.
7. Right-mouse click on **ceInput** and select **Promote Bean Feature**. Ensure that the **Property** radio button is selected and move **IN\_LAST\_NAME, IN\_COMMAND** from the left pane to the right pane by highlighting the property and clicking the >> button.
8. Click **OK** to generate runtime code and the bean info class.
9. Right-mouse click on **ceInput** and select **Add Mapper**. A window will display all mapper beans in VAJ. Select **APhoneBookRecordInputMapper** and click **OK**. A **ceMapperCeInput** object should now be created under the ceInput object.
10. Right-mouse click on the **Output** task and select **Add IByteBuffer Bean**. A window displays all beans in VAJ. Select the **RecordBean** created earlier, **APhoneBookRecordOutput**, and click **OK**.
11. Right-mouse click on **ceOutput1** and select the **Promote Bean Feature**. A window is displayed.
12. Click **OK**.
  - a. Right-mouse click on **ceOutput1** and select **Add Mapper**. A window will display all mapper beans in VAJ. Select **APhoneBookRecordOutputMapper** and click **OK**. A **ceMapperCeOutput1** object should now be created under the ceInput object.
  - b. Click **OK**.

## Modifying the Procedural Adapter Object to call the Commands

1. Select the **APhoneBookPAO** class and expand it.
2. Highlight each of the CRUD methods (insert, retrieve update, and delete).
3. For each method, add in the necessary code to setup and call each Command Object. For the insert() method, the code should look like:

```
public void insert() throws com.ibm.ipaa.IDataKeyAlreadyExistsException {
    APhoneBookCommandOutput pbc = new APhoneBookCommandOutput();
    pbc.setConnectionSpec(this.getConnectionSpec());
    pbc.setCeInputIN_COMMAND("ADD");
    pbc.setCeInputIN_LAST_NAME(this.getLastName());
    pbc.execute();
    String msg = (String) pbc.getCeOutput1OUT_MESSAGE();
    if ((msg != null) && msg.trim().equals("ADDITION OF ENTRY HAS FAILED"))
        throw new com.ibm.ipaa.IDataKeyAlreadyExistsException();
}
```

For the retrieve() method, the code should look like:

```
public void retrieve() throws com.ibm.ipaa.IDataKeyNotFoundException {
    APhoneBookCommandOutput pbc = new APhoneBookCommandOutput();
    pbc.setConnectionSpec(this.getConnectionSpec());
    pbc.setCeInputIN_COMMAND("DISPLAY");
    pbc.setCeInputIN_LAST_NAME(this.getLastName());
    pbc.execute();
}
```

```

String msg = (String) pbc.getCeOutput1OUT__MESSAGE();
if ((msg != null) && msg.trim().equals("SPECIFIED PERSON WAS NOT FOUND"))
    throw new com.ibm.ipaa.IDataKeyNotFoundException();
}

```

For the update() method, the code should look like:

```

public void update() throws com.ibm.ipaa.IDataKeyNotFoundException {
    APhoneBookCommandOutput pbc = new APhoneBookCommandOutput();
    pbc.setConnectionSpec(this.getConnectionSpec());
    pbc.setCeInputIN__COMMAND("UPDATE");
    pbc.setCeInputIN__LAST__NAME(this.getLastName());
    pbc.execute();
    String msg = (String) pbc.getCeOutput1OUT__MESSAGE();
    if ((msg != null) && msg.trim().equals("SPECIFIED PERSON WAS NOT FOUND"))
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}

```

For the del() method, the code should look like:

```

public void del() throws com.ibm.ipaa.IDataKeyNotFoundException {
    APhoneBookCommandOutput pbc = new APhoneBookCommandOutput();
    pbc.setConnectionSpec(this.getConnectionSpec());
    pbc.setCeInputIN__COMMAND("DELETE");
    pbc.setCeInputIN__LAST__NAME(this.getLastName());
    pbc.execute();
    String msg = (String) pbc.getCeOutput1OUT__MESSAGE();
    if ((msg != null) && msg.trim().equals("SPECIFIED PERSON WAS NOT FOUND"))
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}

```

## Exporting the pbe Package

After building the Execute class and creating and testing the Component Broker procedural adaptor object within the VisualAge for Java environment, you can run the unit test program outside of the VisualAge for Java environment. This object must be imported to Object Builder as a persistent object. Importing this object requires that the procedural adaptor object and its corresponding BeanInfo class is exported outside of VisualAge for Java. To run the sample outside of the VisualAge for Java environment, you must export all classes you created and modify the CLASSPATH environment variable.

For ease, export the entire package. This package should contain:

- The new procedural adapter object
- Its corresponding BeanInfo class
- All EAB transaction objects

To export the package outside of VisualAge for Java:

1. Select the paa.mysamples.ims.appc.pbe package to export.
2. From the VisualAge for Java Workbench menu, select **File** → **Export**. Select the **Directory** radio button and click the **Next** button.
3. This opens the Type of Export - wizard. You are going to export all .java and .class files. Select the **Both Java and Class Files** check box.
4. Click the **Next** button.
5. Type one of the following in the **Directory** field.:

```

 x:\MyProj

```

**AIX** \$HOME/MyProj

6. Click the **Finish** button.

When the export completes, the paa.mysamples.ims.appc.pbe package is created under the MyProj directory.

**Note:** There is a process available to verify that the package you exported can run outside of Visual Age for Java. For the latest information on this process, contact your IBM representative.

---

## Developing an IMS-APPC Business Object

This section contains Object Builder and System Management procedures required to create a component named "PhoneBookRec." To create this component, perform the procedures in the following sections.

1. "Importing the Bean" on page 184
2. "Defining the PhoneBookRec Component" on page 184
3. "Creating Client and Server DLL Files" on page 188
4. "Packaging the Application" on page 189
5. "Building the Application - Client and Server" on page 96
6. "Installing the Application" on page 192
7. "Running the Sample Application" on page 194

Before starting Object Builder, complete one of the following procedures:

**WIN**

1. Ensure that your classpath includes the x:\Myproj directory.
2. When you start Object Builder, you must select **Enable Object Builder beta capability** when the Open Project dialog is displayed in Object Builder.
3. Specify x:\MyProj as the base directory for the project.
4. The procedures contained in this section assume that you have correctly set your classpath to include x:\MyProj before starting Object Builder and that you have started Object Builder.
5. Click the **Finish** button.

**AIX**

1. Ensure that your classpath includes the \$HOME/Myproj directory.
2. When you start Object Builder, you must select **Enable Object Builder beta capability** when the Open Project dialog is displayed in Object Builder.
3. Specify \$HOME/MyProj as the base directory for the project.
4. The procedures contained in this section assume that you have correctly set your classpath to include \$HOME/MyProj before starting Object Builder and that you have started Object Builder.
5. Click the **Finish** button.

## Importing the Bean

**WIN** For Windows NT, the bean to import is PBBean from the paa.mysamples.ims.appc.pbe package for the x:\MyProj directory.

**AIX** For AIX, the bean to import is PBBean from the paa.mysamples.ims.appc.pbe package for the \$HOME/MyProj directory.

To import this bean:

1. Select the User-Defined PA Schemas folder from the **Object Builder Tasks and Objects** pane.
2. Open the pop-up menu for **User-Defined PA Schemas**, and select **Import Bean**. This opens the Import Procedural Adaptor Bean - wizard.
3. On this page:
  - a. Type paa.mysamples.ims.appc.pbe.APhoneBookPA0 in the **Class Name** field.
  - b. Click the **Next** button to accept the remaining defaults and to continue to the Names and Services page.
4. On this page:
  - a. Select **LU6.2** for the Connector Type.
  - b. Click the **Next** button to accept the defaults and to continue.
5. On this page:
  - a. Select the **lastName** property from the **Properties** list box.
  - b. Click the >> button to move the associated key required to import the bean.
6. Click the **Finish** button.

The bean is imported into Object Builder. The PBBean schema and its corresponding persistent object (PBBeanPO) are now in the tree view of User-Defined PA Schemas.

## Defining the PhoneBookRec Component

This exercise defines the objects required to create a component named PhoneBookRec. For this component, you will:

1. Create a new business object file
2. Define the business object
3. Connect the data object implementation to the persistent object
4. Define the managed object
5. Generate the code

## Creating the Business Object File

To create the PhoneBookRec business object file:

1. From the **Tasks and Objects** pane, select the User-Defined Business Objects folder.
2. Open the pop-up menu for **User-Defined Business Objects**, and select **Add File**, which opens the Business Object File - wizard to the Name and Attributes page.
3. On this page:
  - a. Type PhoneBookRec in the **Name** field.
  - b. Accept the other defaults.

4. Click the **Finish** button.

The PhoneBookRec file is now under the User-Defined Business Objects folder.

## Defining the Business Object

After creating the new business object file, the business object must be defined. A fully-configured business object consists of the following:

- A business object interface
- An associated key
- An associated copy helper
- A business object implementation

**Defining the Business Object Interface:** To create the PhoneBookRec business object interface:

1. Expand the User-Defined Business Objects folder, and select **PhoneBookRec**.
2. Open the pop-up menu for **PhoneBookRec**, and select **Add Interface**, which opens the Business Object Interface - wizard to the Name and Attributes page.
3. On this page:
  - a. Type PhoneBookRec in the **Name** field.
  - b. Click the **Next** button to continue to the Constructs page.
4. Click the **Next** button to accept the defaults and to continue to the Interface Inheritance page.
5. Click the **Next** button to accept the defaults and to continue to the Attributes page.
6. Define the user-defined attributes.
  - a. Select Attributes from the tree view.
  - b. Open the pop-up menu for Attributes, and select **Add**. This displays the Add dialog.
  - c. In this dialog:
    - 1) Type lastName in the **Attribute Name** field.
    - 2) Select **string** as the **Type**. This displays the **Size** field.
    - 3) Type 0 in the **Size** field.
    - 4) Click the **Add Another** button.
  - d. Repeat the previous step for the remaining attributes of the PhoneBookRec interface. The remaining attributes are:
    - firstName, and click the **Add Another** button.
    - extNumber, and click the **Add Another** button.
    - internalZip, and click the **Refresh** button.
7. Click the **Finish** button.

The **PhoneBookRec** interface is now under the PhoneBookRec file.

**Defining the Key:** To add the key:

1. From the User-Defined Business Object folder, select the **PhoneBookRec** interface.
2. Open the pop-up menu for **PhoneBookRec**, and select **Add Key**, which opens the Key -wizard.
3. Select the **lastName** attribute from the **Business Object Attributes** list.
4. Click the >> button to move this attribute to the **Key Attributes** list.
5. Click the **Finish** button.

The PhoneBookRecKey key is now under the PhoneBookRec interface.

**Defining the Copy Helper:** To add the Copy Helper:

1. From the User-Defined Business Object folder, select the **PhoneBookRec** interface.
2. Open the pop-up menu for **PhoneBookRec**, and select **Add Copy Helper**, which opens the Copy Helper - wizard.
3. Click the **All >>** button to move the attributes from the **Business Object Attributes** list to the **Copy Helper Attributes** list.
4. Click the **Finish** button.

The PhoneBookRecCopy copy helper is now under the PhoneBookRec interface.

**Defining the Business Object Implementation:** To add the business object implementation and data object interface:

1. From the User-Defined Business Object folder, select the **PhoneBookRec** interface.
2. Open the pop-up menu for PhoneBookRec, and select **Add Implementation**, which opens the Business Object Implementation - wizard to the Name and Data Access Pattern page.
3. Define the implementation.
  - a. Select the **Delegating** radio button from the **Pattern for Handling State Data** group box.
  - b. Ensure that the **Create a new one now** radio button is selected from the **Data Object Interface** group box. This option allows you to define the business object attributes that need to be preserved in the data object.
  - c. Click the **Next** button to continue to the Implementation Inheritance page.
4. Click the **Next** button to accept the defaults and to continue to the Implementation Language page.
5. Click the **Next** button to accept the defaults and to continue to the Attributes page.
6. Click the **Next** button to accept the defaults and to continue to the Methods page.
7. Click the **Next** button to accept the defaults and to continue to the Key and Copy Helper page.
8. On this page:
  - a. Verify that the **PhoneBookRecKey** key is selected from the **Key** list.
  - b. Verify that **PhoneBookRecCopy** is selected from the **Copy Helper** list.
  - c. Click the **Next** button to continue to the Handle Selection page.
9. Click the **Next** button to accept the defaults and to continue to the Attributes to Override page.
10. Click the **Next** button to accept the defaults and to continue to the Methods to Override page.
11. Click the **Next** button to accept the defaults and to continue to the Data Object Interface page.
12. Click the **All>>** button to move the attributes in the **Business Object Attributes** list to the **State Data** list.
13. Click the **Finish** button.

The PhoneBookRecBO business object implementation is now under the PhoneBookRec interface, and the PhoneBookRecDO data object interface is now under the PhoneBookRecBO business object implementation.

## Connecting the Data Object Implementation to the Persistent Object

To create the data object implementation and to connect the data object implementation to the persistent object, perform the following procedure.

1. From the User-Defined Business Object folder, select the **PhoneBookRecDO** data object interface.
2. Open the pop-up menu for PhoneBookRecDO, and select **Add Implementation**. This displays the Data Object Implementation - wizard.
3. On this page:
  - a. Set the **BOIM with any key** radio button from the **Environment** group box to indicate that the data object is part of a component installed in a business object application adaptor with instances being located by key objects.
  - b. Set the **Procedural Adaptors** radio button from the **Form of Persistent Behavior and Implementation** group box.
  - c. Click the **Next** button to continue to the Implementation Inheritance page.
4. On this page:
  - a. Verify that **IPAAExtLocalToServer IPAAExtLocalToServer** is selected as a parent.
  - b. Click the **Next** button to continue to the Attributes page.
5. Click the **Next** button to accept the defaults and to continue to the Methods page.
6. Click the **Next** button to accept the defaults and to continue to the Key and Copy Helper page.
7. Click the **Next** button to accept the defaults and to continue to the Associated Persistent Objects page.
8. On this page:
  - a. Select **Persistent Object Instances**.
  - b. Open the pop-up menu for **Persistent Object Instances**, and select **Add**.
  - c. Type APhoneBookPAOPO in the **Instance Name** field.
  - d. Click the **Next** button.
9. On this page:
  - a. Select **lastName** from the **Attributes** list.
  - b. Open the pop-up menu for **lastName**, and select **Primitive**.
  - c. Select APhoneBookPAOPO.lastName from the **Persistent Object Attribute** list.
  - d. Add 1-to-1 mappings for the other attributes under the **Attributes** tree view as you just did for lastName.
  - e. Click the **Next** button.
10. On this page:
  - a. Select **insert** from the **Special Framework Methods** list.
  - b. Open the pop-up menu for insert, and select **Add Mapping**.
  - c. Select **APhoneBookPAOPO.insert** from the **Persistent Object Method** list.
  - d. Add 1-to-1 mappings for the other methods under the **Special Framework Methods** tree view as you just did for insert. In addition, add a mapping for the setConnectin() method.
11. Click the **Finish** button.

The PhoneBookRecDOImpl data object implementation is now under the PhoneBookRecDO interface, and the APhoneBookPAOPO persistent object is now under the PhoneBookRecDOImpl data object implementation.

## Defining the Managed Object

To add the managed object:

1. From the User-Defined Business Object folder, select the **PhoneBookRecBO** business object implementation.
2. Open the pop-up menu for **PhoneBookRecBO**, and select **Add Managed Object**, which opens the Managed Object - wizard to the Name and Application Adaptor page.
3. Set the **Transaction Service** radio button under **Service to Use**.
4. Click the **Next** button to accept the defaults and continue to the Implementation Inheritance page.
5. Click the **Finish** button.

## Generating the Code

To generate the application code:

1. From the User-Defined Business Object folder, select **PhoneBookRec**.
2. Open the pop-up menu for PhoneBookRec, and select **Generate** → **All**.

Code generation starts. Progress is indicated in the lower-left corner of the window.

## Creating Client and Server DLL Files

The defined objects must be built into two separate DLL files;

- One that runs on the client and provides access to the business object interface, key and copy helper.
- One that runs on the server and provides access to the managed object and the rest of the component.

The client DLL file must be defined before the server DLL file. When the server DLL file is defined, it must link to the client DLL file. After defining the objects that comprise each DLL file, these files can be built.

## Defining the Client DLL File

To add the client DLL file:

1. Select the **Build Configuration** folder.
2. Open the pop-up menu for Build Configuration, and select **Add client DLL**. This displays the Name and Option page of the Add Client DLL - wizard.
3. Type apbcC in the **Name** field.
4. Deselect **390**.
5. Click the **Next** button to continue to the Client Source Files page.
6. Click the **All>>** button to move the client source files to the **Items chosen** list.
7. Click the **Finish** button.

The apbcC client DLL file is now under the Build Configuration folder.

## Defining the Server DLL File

To add the server DLL:

1. Select the **Build Configuration** folder.
2. Open the pop-up menu for **Build Configuration**, and select **Add Server DLL**. This displays the Name and Option page of the Server DLL - wizard.
3. Type apbcS in the **Name** field.
4. Deselect **390**.
5. Click the **Next** button to continue to the Server Source Files page.
6. Click the **All>>** button to move the server source files to the **Items chosen** list.
7. Click the **Next** button to continue to the Libraries to Link With page.
8. Click the **All>>** button to move all the files from the **Items Available** list to the **Items chosen** list.
9. Click the **Finish** button.

The apbcS server DLL file is now under the Build Configuration folder.

## Building the DLL Files

To generate the makefiles to build the configuration:

1. Select the **Build Configuration** folder.
2. Open the pop-up menu for Build Configuration, and select **Generate** → **All** → **All Targets**.

The code generation begins.

## Packaging the Application

Packaging the application consists of the following procedures:

1. Creating the application family
2. Defining the application
3. Creating the container instance
4. Configuring the managed object
5. Generating the application

### Creating the Application Family

To add the application family:

1. Select the **Application Configuration** folder.
2. Open the pop-up menu for **Application Configuration**, and select **Add Application Family**. This displays the Name page of the Application Family - wizard.
3. Type apbcAppFam in the **Name** field.
4. Click the **Finish** button.

The apbcAppFam application family is now under the Application Configuration folder.

## Defining the Application

To add the Application:

1. Select the **apbcAppFam** application family.
2. Open the pop-up menu for **apbcAppFam**, and select **Add Application**, which opens the Add Application -wizard to the Name and Environment page.
3. Type apbcApp in the **Application Name** field.
4. Click the **Finish** button.

The apbcApp application is now under the apbcAppFam application family.

## Creating the Container Instance

To add the new container instance:

1. Select the **Container Definition** folder.
2. Open the pop-up menu for Container Definition, and select **Add Container Instance**, which opens the Container - wizard.
3. Type apbcContainer in the **Name** field.
4. Deselect **390**.
5. Click the **Next** button to continue to the Work Load Manager Container page.
6. Click the **Next** button to continue to the Policies and Services page.
7. On this page:
  - a. Set the **Use PAA Transaction Services** radio button.
  - b. Click the **Next** button to continue to the Services page.
8. On this page:
  - a. Set the **Throw an exception and abandon the call** radio button under **Behavior for Methods Called Outside a Session**.
  - b. Type IMS\_pbc\_Connection in the **Connection Name** field.
  - c. Click the **Next** button to continue to the Data Access Patterns page.
9. On this page, ensure that the **Delegating** check box is set under both **Business Object** and **Data Object** blocks.
10. Click the **Finish** button.

The apbcContainer container is now under the Container Definition folder.

## Configuring the Managed Object

To add the managed object for the Application:

1. Select the **apbcApp** application.
2. Open the pop-up menu for **apbcApp**, and select **Add Managed Object**, which opens the Configure Manage Object - wizard.
3. In this window:
  - a. Verify that **PhoneBookRecMO PhoneBookRecMO** is in the **Managed Object** field.
  - b. Click the **Next** button to continue to the Data Object Implementations page.

4. On this page:
  - a. Select **Implementation**.
  - b. Open the pop-up menu for **Implementation**, and select **Add**.
  - c. Select PhoneBookRecDOImpl PhoneBookRecDOImpl from the **Data Object Implementation** list.
  - d. Click the **Next** button to continue to the Container page.
5. Click the **Next** button to continue to the Home page.
6. On this page, select **BOIMHomeOfRegHomes** from the **Home Name** list.
7. On this page, select the **Default Home** radio button under Default Home.
8. Click the **Finish** button.

The PhoneBookRecMO managed object is now under the Application Configuration folder.

## Generating the Applications

To generate the application family:

1. Select the **apbcAppFam** application.
2. Open the pop-up menu for **apbcApp**, and select **Generate**.

**Note:** If you do not have InstallShield installed on your system, click the **Yes** button when the dialog concerning InstallShield is displayed. When code generation completes, the Method Implementation pane contains the apbcApp.ddl file. You can now close Object Builder.

## Building the Application - Client and Server

Perform one of the following procedures: WIN

All imported and generated files are placed in the `x:\MyProj\Working\NT` directory (where `x:\MyProj` is the working directory when Object Builder was started).

1. Change directory to:
 

```
x:\MyProj\Working\NT
```
2. Enter:
 

```
nmake -f all.mak cpp java
```

AIX All imported and generated files are placed in the `$HOME/MyProj/Working/AIX` directory (where `$HOME/MyProj` is the working directory when Object Builder was started).

1. Change directory to:
 

```
$HOME/MyProj/Working/AIX
```
2. Enter:
 

```
make -f all.mak cpp java
```

Everything in the sample application is built.

## Installing the Application

Installing an application consists of:

1. Loading the application
2. Configuring the application

These procedures assume that you are currently logged on to DCE, and that you are currently using the System Manager User Interface. If not, logon to DCE and start the System Manager User Interface.

## Loading the Application onto System Management

To install the pbc server application:

1. Start the System Manager User Interface if it is not already started.
2. Become an Expert user (**View** → **User Level** → **Expert**).
3. Expand Host Images, and select your host name.
4. From the pop-menu, select **Load Application**. This opens the Load Application dialog. Do one of the following:

**WIN** For Windows NT, browse for and select **apbcAppFam.ddl**  
(x:\MyProj\Working\NT\apbcAppFam\apbcAppFam.ddl).

**AIX** For AIX, browse for and select **apbcAppFam.ddl**  
(\$HOME/MyProj/Working/AIX/apbcAppFam/apbcAppFam.ddl).

**Note:** A warning may be displayed about iCachedWLMSystemManagedObjects while the DDL is loading. You can ignore this warning.

## Configuring the Application with System Management

Configuring the apbcApp application:

1. Expand Available Applications, and select **apbcApp**.
2. Open the pop-up menu for **apbcApp**, and select **Drag**.
3. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select **Sample Configuration**.
4. Open the pop-up menu for Sample Configuration, and select **Add Application**.
5. Define the server:
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select **Sample Configuration**.
  - b. Open the pop-up menu of **Sample Configuration**, and select **New** → **Server (free standing)**. This displays a new dialog box.
  - c. Type apbcSrv as the name for the server group.
  - d. Click the **OK** button. The apbcSrv is now under Server (free standing).
6. Associate the application with the server.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Applications, and select **apbcApp**.
  - b. Open the pop-up menu of **apbcApp**, and select **Drag**.

- c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Server (free standing), and select **apbcSrv**.
  - d. Open the pop-up menu of **apbcSrv**, and select **Configure Application**.
7. Associate the iPAAServices with the server.
- a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, Applications, and select **iPAAServices**.  
**Note:** If iPAAServices is not found in the Applications branch, you can add it by:
    - 1) Expand Host Images-myhost-Application Family Installs-iPAAApplications-Application Installs, and select **iPAAServices**.
    - 2) Open the pop-up menu for **iPAAServices**, and select **Drag**.
    - 3) Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select **Sample Configuration**.
    - 4) Open the pop-up menu for **Sample Configuration**, and select **Add Application**.
  - b. Open the pop-up menu for iPAAServices, and select **Drag**.
  - c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Server (free standing), and select **apbcSrv**.
  - d. Open the pop-up menu for **apbcSrv**, and select **Configure Application**.
8. Configure the server with the host.
- a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Server (free standing), and select **apbcSrv**.
  - b. From the pop-up menu for **apbcSrv**, select **Drag**.
  - c. Expand Hosts, and select your server.
  - d. From the pop-up menu for your server, select **Configure Server (free standing)**.
9. Configure the APPC connection:
- a. On the System Management GUI main window, expand Host Images → *myhost* → Server Images → apbcSrv → APPC Connection Images.
  - b. Right-click on IMS\_pbe\_Connection.
  - c. In the pop-up menu, click **Edit....** A new dialog opens.
  - d. Click the **Main** tab.
  - e. In this window:
    - 1) Enter your fully-qualified local LU name in the **Fully-qualified Local LU name** field. This is not your CP name, but an LU name under your node Control Point. You can obtain this from your network administrator.
    - 2) Enter your fully-qualified partner LU name in the **Fully-qualified Partner LU name** field (IBM internal users can enter USIBMSTY.STY7IM16).
    - 3) For the **Mode Name**, enter L62MDE01.
    - 4) For the **Remote Procedure Type**, click **IMS**.
    - 5) For the **Transaction Program name (TPN)**, enter IVTN0.
    - 6) For the **Security mechanism**, click **IMS**.
    - 7) For the Transaction type, click **Pessimistic**.

- 8) Accept all other remaining defaults.
10. Activate the configuration.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select **Sample Configuration**.
  - b. Open the pop-up menu for Sample Configuration, and select **Activate**, which automatically starts the application server. Wait for a completion message in the Action Console window before continuing.

## Running the Sample Application

For IVP install instructions for IMS, see *IMS/ESA Version 6 Install Volume 1*. The entire book contains information on installing and configuring the IVP sample. Chapter 11, entitled “Install/IVP Application,” discusses the sample IMS application.

To run the sample client application, perform one of the following procedures:

**WIN**

1. Copy the pbcclient.mak and pbcclient.cpp from:  
x:\CBroker\samples\installVerification\Application\IMSAPPCCli  
to:  
x:\MyProj\Working
2. Change directory to:  
x:\MyProj\Working
3. Enter:  
nmake -f pbcclient.mak
4. Enter:  
pbcclient

**AIX**

1. Copy the pbcclient.mak and pbcclient.cpp from:  
\$HOME/samples/installVerification/Application/IMSAPPCCli  
to:  
\$HOME/MyProj/Working/AIX
2. Change directory to:  
\$HOME/MyProj/Working/AIX
3. Enter:  
make -f pbcclient.mak
4. Enter:  
pbcclient

---

## Chapter 9. Developing a CICS-APPC Application

This chapter provides information for building a sample Component Broker application with a CICS backend. This chapter contains the following information:

- “The CICS-APPC Sample Application”
- “Enterprise Access Builder Procedures” on page 196
- “Developing a CICS-APPC Business Object” on page 205

**Note:** To walk-through this sample, the CICS and IMS Application Adaptor SDK and IBM VisualAge Java with EAB must be installed.

### Important Information

Before walking through this sample, please refer to the *Late Breaking News* provided with Component Broker before performing the exercise in this chapter. This document provides the latest information regarding the CICS and IMS application adaptor samples, which may differ from the instructions for this sample application.

---

## The CICS-APPC Sample Application

The CICS-APPC sample application is a mock account database consisting of the following fields:

- Account Balance
- Account Number
- The type of customer
- The type of account
- Utilities

The ACashAcct interface will be implemented during this exercise. The data object implementation for this business object will leverage a procedural adaptor object that in turn uses CICS through the APPC to provide the state data back to the data object.

Although this sample application is not a full-blown CICS application, it captures the essence of an application involving multiple APPC requests and delivering some amount of business function. This sample application can be extended and customized to explore different CICS-APPC application issues.

The sample that you build in this section is included with the product and can be built by following the steps in the HTML file in one of the following:

WIN

`CBroker\samples\InstallVerification\PAA\readme.htm`

AIX

`$HOME/samples/InstallVerification/PAA/readme.htm`

---

## Enterprise Access Builder Procedures

This exercise defines the classes required to create a Component Broker procedural adapter object (PAO) named "ABeCashAcct". For this object, you will perform the following steps:

1. "Creating a project/package under VisualAge for Java" on page 197
2. "Creating the Procedural Adaptor Object and Key" on page 197
3. "Importing the Customer COBOL File" on page 199
4. "Creating the Record Mapper" on page 200
5. "Creating the ABeCashAcctCommand Class" on page 201
6. "Modifying the Procedural Adaptor Object to call the Commands" on page 204
7. "Exporting the ABeCashAcct Package" on page 205
8. "Running the Sample Application" on page 217

**WIN** If you are using VisualAge for Java on Windows 95 or Windows NT, from the **Start** menu, select **Programs** → **IBM VisualAge for Java for Windows** → **IBM VisualAge for Java**.

**AIX** If you are using VisualAge for Java on AIX, type `vajide` on the command line.

If the VisualAge Quick Start dialog appears, select **Go to the Workbench** and click **OK**. The IDE appears.

From the Window pull down, select **Options**. Select **Design Time** and un-check **Inherit BeanInfo** of bean superclass. Click **OK**.

## Importing Pre-requisite Features into the Workspace

1. Select **File** → **Quick Start**.
2. Select **Features** in the left pane, **Add Feature** in the right pane.
3. Click **OK**.
4. Select the following:

- IBM Procedural Application Adapter 1.0
- CICS Connector 3.0
- IBM Component Broker Host On Demand 1.0
- IBM Component Broker Connectors 1.0
- IBM Enterprise Access Builder Library 2.0
- IBM Enterprise CICS Access Builder Library 1.0
- IBM Component Broker PAA Samples for CICS 1.0

Click **OK**.

•

You may ignore the following expected errors this introduces in the following packages:

- `com.ibm.ivj.communications`
- `com.ibm.ivj.trace`
- `com.ibm.eNetwork.ECL`
- `com.ibm.eNetwork.ncod.services.RAS`

**Note:** If you do not see all of these features listed, they have been previously installed. To confirm, perform the following steps:

- a. Select **File** → **Quick Start**.
- b. Select **Features** → **Delete Feature** and see which features are already loaded (then Cancel).

## Creating a project/package under VisualAge for Java

1. Create the Component Broker Samples project, if it does not already exist:
  - a. Right-click on the VisualAge for Java desktop.
  - b. From the pop-up menu, select **Add** → **Project**.
  - c. Type `CBSamples` in the **Project Name** field, and press the **Enter** key. The **CBSamples** project should be under the VisualAge for Java list of projects.
2. From the list of projects, select **CBSamples**.
3. Open the pop-up menu of **CBSamples**, and select **Add Package**. This creates a package for the project.
4. Type `paa.mysamples.cics.appc.acct` for the new package, and click **Finish**.

**Note:** If you are using the default mouse configuration, right-click on the denoted item. You do not have to select the item before opening its menu. You can select the item and open its menu with a right-click.

## Creating the Procedural Adaptor Object and Key

The procedural adaptor object inherits from `com.ibm.ivj.eab.paa.EntityProceduralAdapterObject`, which serves as a base implementation for all procedural adaptor objects. As a subclass of `EntityProceduralAdapterObject`, the procedural adaptor object contains the CRUD methods (create (or insert), retrieve, update, and delete). However, these methods are all empty-bodied. You must define their implementation for your procedural adaptor object.

The attributes defined in the `ABeCashAcct` interface are essential. Thus, the procedural adaptor object, as the adaptor that connects the Component Broker data object to the backend system, should contain the properties that correspond to these attributes.

1. From the VisualAge for Java desktop under the `CBSamples` project, select **paa.mysamples.cics.appc.acct**.
2. Open the pop-up menu for **paa.mysamples.cics.appc.acct**, and select **Add** → **Class**.
3. In this dialog:
  - a. Type `ABeCashAcctPAO` in the **Class Name** field.
  - b. Click **Browse** to select the Superclass.
    - 1) Browse for and select **EntityProceduralAdapterObject** as your Superclass.
    - 2) Click **OK** to close the dialog.
4. Click **Finish**.

Add the properties for the **ABeCashAcctPAO** interface.

1. Select the **ABeCashAcctPAO** interface.
2. Open the pop-up menu for **ABeCashAcctPAO**, and select **Open**, which opens the Object Editor notebook.
3. In this notebook:

- a. Select the **BeanInfo** tab.
- b. From the menu bar, select **Features** → **New Property Feature**, which opens the wizard - New Property Feature.
- c. In this wizard:
  - 1) Type the name of the new property in the **Property name** field. For simplicity, use the same name as used in the ACashAcct interface. For example, use:
 

```
balance
res_type
account_ID
type
utilities
```

for the properties as defined in the ABeCashac.cpp file. Each of these properties must be defined individually. For this step (first time) type `balance`. For each subsequent time, type `res_type`, `account_ID`, `type` or `utilities`, respectively.
  - d. For all properties except `balance`, select **java.lang.String** from the pull down menu of the **Property type** field. For the `balance` property, select **int** (not `int[]`).
  - e. Accept the other defaults and click **Finish**.
4. Close the object editor.
5. Define the other properties (**Features** → **New Property Feature**)

Next, create the Key for the PAO object by completing the following steps:

1. From the VisualAge for Java desktop under the CBSamples project, select **paa.mysamples.cics.appc.acct**.
2. Open the pop-up menu for **paa.mysamples.cics.appc.acct**, and select **Add** → **Class**.
3. In this dialog:
  - a. Type `ABeCashAcctPAOKey` in the **Class Name** field.
  - b. Click **Browse** to select the Superclass.
    - 1) Browse for and select **BusinessObjectKey** as your Superclass.
    - 2) Click **OK** to close the dialog.
  - c. Click **Finish**.

Add the properties for the **ABeCashAcctPAOKey** interface:

1. Select the **ABeCashAcctPAOKey** interface.
2. Open the pop-up menu for **ABeCashAcctPAOKey**, and select **Open**, which opens the Object Editor notebook.
3. In this notebook:
  - a. Select the **BeanInfo** tab.
  - b. From the menu bar, select **Features** → **New Property Feature**, which opens the wizard - New Property Feature.
  - c. In this wizard:
    - 1) Type the name of the new property in the **Property name** field. For example, use:
 

```
res_type
account_ID
```

for the properties that are going to be the key attributes. You can select **java.lang.String** for the type of the two properties.

- 2) Accept the other defaults and click the **Next** button.
  - 3) Click **Finish**.
  - 4) Close the object editor.
4. Modify the **ABeCashAcctPAOKey** and **ABeCashAcctPAO** to tie the PAO and key class together.

### **ABeCashAcctPAOKey**

1. Select and expand the **ABeCashAcctPAOKey** class.
2. Highlight the **getPropertyValues()** method. This method is used by the Enterprise Access Builder (EAB) run time to calculate a value to key into the Enterprise Access Builder (EAB) cache. It needs to be modified to specifically return just the key values.
3. In the source pane, return an array of Objects that make up the key, by invoking the methods that get the key properties. For example:

```
return new Object[] { this.getRes_type(), this.getAccount_ID()};
```

4. Save the changes to the modified PAO Key class by pressing **Ctrl-S**.

### **ABeCashAcctPAO**

1. Select and expand the **ABeCashAcctPAO** class.
2. Modify the getters for the key property values (**getRes\_type()** and **getAccount\_ID()**) by getting the key class associated with this PAO and returning that value.

In **getAccount\_ID()**

```
ABeCashAcctPAOKey key = (ABeCashAcctPAOKey) this.getKey();  
return key.getAccount_ID();
```

In **getRes\_type()**

```
ABeCashAcctPAOKey key = (ABeCashAcctPAOKey) this.getKey();  
return key.getRes_type();
```

3. Save the changes to the modified PAO by pressing **Ctrl-S**.

## **Importing the Customer COBOL File**

1. Select the package that you have created
2. Open the pop-up menu for the package you are working under, and select **Tools** → **Records** → **Create Cobol Record Type...**, a wizard window will appear.
3. In this wizard window:
  - a. In the **Class Name** field, type **ABeCashAcctInfo**.
  - b. In the **COBOL File** field, browse through the files to locate the **BeCashAcct.ccp** file. It should be located in one of the following:

**WIN** *CBroke*\samples\InstallVerification\PAA\Backend\CashAcct\

**AIX** \$HOME/samples/InstallVerification/PAA/Backend/ACashAcct and select **Open**.

- c. Check that the Project and Package names are correct
- d. Select **Next** to continue to the next screen.

4. On the next screen:
  - a. In the list of Available level 01 commareas select **WS-COMMAREA-BUFFER** and click > to move it to the Selected commareas list.
  - b. Check the box beside the **RecordType intended for CICS** field.
  - c. Click **Finish** when this is complete. A new class named **ABeCashAcctInfo** appears in the designated package.
5. Select the **ABeCashAcctInfo** class.
6. Open the pop-up menu for the **ABeCashAcctInfo** class, and select **Tools** → **Records** → **Generate Records...**, the Generate Records wizard will appear.
7. In this wizard window:
  - a. In the **Class Name** field enter **ABeCashAcctRecord**.
  - b. Select the **Beans** radio button to generate the records as beans.
  - c. Select the **Direct** radio button to access the record fields directly.
  - d. Select the **Dynamic Records** radio button to generate the records as dynamic records.
  - e. Check that the Project and Package names are correct and click **Next**.
8. In the next window:
  - a. Change Endian, Remote Integer Endian, Code Page and Machine Type to the correct values for the CICS server. (ex. the code page for North American MVS is 037, for North American NT is 437).
 

**Note:**  WIN You will have to change all of the values if you are going to a Transaction Server on Windows NT. Also remember to change your endianness to littleEndian (if you are running this from Windows NT).
  - b. Click **Finish** when this is complete. Two new classes appear in your package: **ABeCashAcctRecord** and **ABeCashAcctRecordBeanInfo**.

## Creating the Record Mapper

1. Select the sample package again that you have created and expand it.
2. Select class **ABeCashAcctRecord**.
3. Open the pop-up menu for the **ABeCashAcctRecord** class, and select **Tools** → **Mapper Editor...**, a Mapper wizard will appear.
4. In this window
  - a. From the Code Generation pulldown, select **Set Target mapper**. A window containing three fields appears.
  - b. In this window:
    - 1) Enter the project and package name of this sample in the first two fields.
    - 2) In the **Class** field, enter: **ABeCashAcctRecordMapper**.
    - 3) Click **OK**.
  - c. Select **Change Input bean** from the Code Generation pulldown. A window containing one field appears.
  - d. In this window:
    - 1) Click the **Browse** button beside the field and enter **ABeCashAcct** in the **Pattern** field, a list of matching classes appear.

- 2) Select the **ABeCashAcctRecord** class (corresponding to the sample package) and click **OK**.
- 3) Click **OK** once more to select the Input Bean class. You will be advised "All of your connections will be lost. Do you wish to proceed?", click the **Yes** button.
- 4) You will now see a list of fields available from **ABeCashAcctRecord**.
- e. Select the **Add** button located at the bottom left of the wizard window and enter **ABeCashAcct** in the **Pattern** field, a list of classes will appear.
- f. Select the **ABeCashAcctPAO** class corresponding to the package you are currently using.
- g. Click **OK** when this is complete. You will now see a directory named `java.lang.Object` in the Output Beans side of the window.
- h. Expand this directory until the first instance of **ABeCashAcctPAO** is visible (should be able to see **account\_ID**, **res\_type**, **balance**, etc.)
- i. Select the **account\_ID** field of the **ABeCashAcctPAO** object. Move the cursor to the right hand side of the screen and select **COMM\_\_ACCOUNTID**. At the bottom of the screen, select the button to connect the two fields. Note that the **ABeCashAcctPAO account\_ID** field should be connected to **COMM\_\_ACCOUNTID** on the input side. Repeat this step to form connections between the rest of the **ABeCashAcctPAO** fields. Click **Apply** and **OK** when this is complete. A new class called **ABeCashAcctRecordMapper** will appear in the current package.

## Creating the ABeCashAcctCommand Class

1. Select the package that you have been working in.
2. Open the pop-up menu of the package, and select **Add** → **Class**. A wizard appears.
3. In this window:
  - a. Enter the project and package names of the sample application into their corresponding fields.
  - b. Choose the **Create a new class** radio button and enter **ABeCashAcctCommand** for the Class Name.
  - c. To choose the Superclass, select the **Browse** button and select **CommunicationCommand** from the list.
  - d. Click **OK**.
  - e. Ensure that the **Compose the class visually** radio button is NOT selected and click **Finish**.
4. Select the **ABeCashAcctCommand** class
5. From the pop-up menu for the class, select **Open To** → **BeanInfo**.
6. In this Dialog:
  - a. Select **Features** → **Generate BeanInfo** class. This will generate a new **BeanInfo** class for your command class.
  - b. Select **Features** → **Add Available Features**.
    - 1) In the **Add Available Features** dialog, select the following features that may appear:
      - class
      - communication
      - connectionSpec
      - disconnectCommunication
      - expectedTriggerClass
      - input
      - interactionSpec
      - mappedObjects

- mappingHelper
  - output
- 2) Click on the **OK** button.
- c. Close the Command Class.
7. Open up the pop-up menu of the class, and select **Tools** → **Command Editor**. A new dialog will be displayed.
  8. In this dialog:
    - a. Right-mouse click on the Communication task and select **Add ConnectionSpec**. A window will be displayed displaying all objects that inherit from ConnectionSpec.
    - b. In this window:
      - 1) Choose **APPCConnectionSpec**.
      - 2) Select **OK** and the window will be closed and a connectionSpec entitled **ceConnectionSpec** will be displayed under the Communication task.
    - c. Select **ceConnectionSpec**, right-mouse click on it and select **Properties**. A window will be displayed that will allow you to change the bean properties.
    - d. In this window:
      - 1) In the **CICSProgramName** field, enter the name of the CICS program, BECASHAC.
      - 2) In the **LocalLUName** field, enter your local LU name. e.g., PAA01001:
      - 3) In the **modeName** field, enter the mode name. e.g., LU62PS.
      - 4) In the **partnerLUName** field, enter the partner LU Name. e.g., USIBMZP.CICS4
      - 5) In the **remoteProcType** field, enter one of the following (the sample uses 2 for CICS\_DTP):
 

|          |              |
|----------|--------------|
| <b>0</b> | for unknown  |
| <b>1</b> | for IMS      |
| <b>2</b> | for CICS DTP |
| <b>3</b> | for CICS DPL |
      - 6) In the **securityType** field, enter one of the following (the sample uses 0 for unknown):
 

|          |             |
|----------|-------------|
| <b>0</b> | for unknown |
| <b>1</b> | for IMS     |
| <b>2</b> | for CICS    |
      - 7) In the **transactionProgramName** field, enter BDPL or BDTP. This is the transaction program used by the APPC support. It should correspond to the remoteProcType above.
      - 8) In the **transactionType** field, enter one of the following (the sample uses 2 for optimistic)α
 

|          |                       |
|----------|-----------------------|
| <b>0</b> | for unknown           |
| <b>1</b> | for non-transactional |
| <b>2</b> | for optimistic        |
| <b>3</b> | for pessimistic       |
      - 9) Click **OK** to close the property window to save the changes.
    - e. Right-mouse click on the Communication task and select **Add InteractionSpec**. A window displays all objects that inherit from InteractionSpec.

- f. In this window:
- 1) Choose **APPCInteractionSpec**.
  - 2) Select **OK** and the window closes and an interactionSpec entitled **celInteractionSpec** is added under the Communication Task.
- g. Select **celInteractionSpec**, right-mouse click on it and select **Properties**. A window displayed that allows you to change the bean properties.
- h. In this window:
- 1) In the **codepage** field, enter 037 (the codepage for an MVS system).
  - 2) In the **intEndian** field, enter 0.
  - 3) In the **machineType** field, enter one of the following (the sample uses 0 for MVS):
 

|          |         |
|----------|---------|
| <b>0</b> | for MVS |
| <b>1</b> | for OS2 |
| <b>2</b> | for NT  |
| <b>3</b> | for AIX |
  - 4) In the **mode** field, enter 0.
  - 5) In the **otherEndian** field, enter 1.
  - 6) In the **progName** field, enter BECASHAC.
  - 7) Click **OK** to close the properties window.

## Inbound Side of Command

1. Right-mouse click on the Input task and select **Add IByteBuffer Bean**. A window displays all beans in VisualAge for Java. Choose the RecordBean created earlier, **ABeCashAcctRecord**, and click **OK**.
2. Right-mouse click on **celInput** and select **Promote Bean Feature**. Ensure that the **Property** radio button is selected and move **COMM\_\_RES\_\_TYPE**, **COMM\_\_ACCOUNTID** and **COMM\_\_REQUEST\_\_TYPE** from the left pane to the right pane by highlighting those properties and clicking the >> button.
3. Click **OK** to generate run time code and the bean info class.
4. Right-mouse click on **celInput** and select **Add Mapper**. A window displays all mapper beans in VisualAge for Java. Select **ABeCashAcctRecordMapper** and select **OK**. A **ceMapperCelInput** object should now be created under the celInput object.

## Outbound Side of Command

1. Right-mouse click on the Output task and select **Add IByteBuffer Bean**. A window displays all beans in VisualAge for Java. Select the RecordBean created earlier, **ABeCashAcctRecord**, and click **OK**.
2. Right-mouse click on **ceOutput1** and choose the **Promote Bean Feature**. A window is displayed.
3. Ensure that the **Property** radio button is selected and highlight **COMM\_\_ACCOUNTID** and move it over. Do the same for **COMM\_\_RES\_\_TYPE** (These are the two key fields)
4. Click the **Method** radio button and move the key attribute getters over: **getCOMM\_\_ACCOUNTID()** and **getCOMM\_\_RES\_\_TYPE()**. Move **getCOMM\_\_RETURN\_\_VALUE\_\_1()** over as well. Click **OK** when finished.
5. Select **ceOutput1** and **Add Mapper**. Select **ABeCashAcctRecordMapper** and click **OK**. A **ceMapperCeOutput1** object should now be created under the ceOutput1 object.

6. Click **OK**

## Modifying the Procedural Adaptor Object to call the Commands

1. Select the **ABeCashAcctPAO** class and expand it.
2. Highlight each of the CRUD methods (insert, retrieve update, and delete).
3. For each method, add in the necessary code to set up and call each Command Object. For the **insert()** method, the code should look like:

```
public void insert() throws com.ibm.ipaa.IDataKeyAlreadyExistsException {
    ABeCashAcctCommand bec = new ABeCashAcctCommand();
    bec.setConnectionSpec(this.getConnectionSpec());
    bec.setCeInputCOMM__REQUEST__TYPE((short)1);
    bec.setCeInputCOMM__ACCOUNTID(this.getAccount_ID());
    bec.setCeInputCOMM__RES__TYPE(this.getRes_type());
    bec.execute();
    if (bec.ceOutput1GetCOMM__RETURN__VALUE__1().equals("00000014"))
        throw new com.ibm.ipaa.IDataKeyAlreadyExistsException();
}
```

For the **retrieve()** method, the code should look like:

```
public void retrieve() throws com.ibm.ipaa.IDataKeyNotFoundException {
    ABeCashAcctCommand bec = new ABeCashAcctCommand();
    bec.setConnectionSpec(this.getConnectionSpec());
    bec.setCeInputCOMM__REQUEST__TYPE((short)2);
    bec.setCeInputCOMM__ACCOUNTID(this.getAccount_ID());
    bec.setCeInputCOMM__RES__TYPE(this.getRes_type());
    bec.setCeOutput1COMM__ACCOUNTID(this.getAccount_ID());
    bec.setCeOutput1COMM__RES__TYPE(this.getRes_type());
    bec.execute();
    if (bec.ceOutput1GetCOMM__RETURN__VALUE__1().equals("00000013" )
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}
```

For the **update()** method, the code should look like:

```
public void update() throws com.ibm.ipaa.IDataKeyNotFoundException {
    ABeCashAcctCommand bec = new ABeCashAcctCommand();
    bec.setConnectionSpec(this.getConnectionSpec());
    bec.setCeInputCOMM__REQUEST__TYPE((short)3);
    bec.setCeInputCOMM__ACCOUNTID(this.getAccount_ID());
    bec.setCeInputCOMM__RES__TYPE(this.getRes_type());
    bec.execute();
    if (bec.ceOutput1GetCOMM__RETURN__VALUE__1().equals("00000013" )
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}
```

For the **del()** method, the code should look like:

```
public void del() throws com.ibm.ipaa.IDataKeyNotFoundException {
    ABeCashAcctCommand bec = new ABeCashAcctCommand();
    bec.setConnectionSpec(this.getConnectionSpec());
    bec.setCeInputCOMM__REQUEST__TYPE((short)4);
    bec.setCeInputCOMM__ACCOUNTID(this.getAccount_ID());
    bec.setCeInputCOMM__RES__TYPE(this.getRes_type());
    bec.execute();
    if (bec.ceOutput1GetCOMM__RETURN__VALUE__1().equals("00000013" )
        throw new com.ibm.ipaa.IDataKeyNotFoundException();
}
```

## Exporting the ABeCashAcct Package

After building the Execute class and creating and testing the Component Broker procedural adaptor object within the VisualAge for Java environment, you can run the unit test program outside of the VisualAge for Java environment. This object needs to be imported to Object Builder as a persistent object. Importing this object requires that the procedural adaptor object and its corresponding BeanInfo class is exported outside of VisualAge for Java. To run the sample outside of the VisualAge for Java environment, you must export all classes you created and modify the CLASSPATH environment variable.

For ease, export the entire package. This package should contain:

- The new procedural adapter object
- Its corresponding BeanInfo class
- All Enterprise Access Builder (ESAB) transaction objects

To export the package outside of VisualAge for Java:

1. Select the package to export.
2. From the VisualAge for Java Workbench menu, select **File** → **Export**. This opens the Export - wizard. You are going to export all .java and .class files.
3. Click the **Next** button.

Type one of the following in the **Directory** field.

WIN x:\MyProj

AIX \$HOME/MyProj

4. Select the **.java and .class** check boxes.
5. Click the **Finish** button.

When the export completes, the paa.mysamples.cics.appc.acct directory is created under the MyProj directory.

**Note:** There is a process available to verify that the package you exported can run outside of VisualAge for Java. For the latest information on this process, contact your IBM representative.

---

## Developing a CICS-APPC Business Object

This section contains Object Builder and System Management procedures required to create a component named "ACashAcct". To create this component, perform the procedures in the following sections.

1. "Importing the Bean" on page 206
2. "Defining the ACashAcct Component" on page 207
3. "Creating Client and Server DLL Files" on page 210
4. "Packaging the Application" on page 212
5. "Building the Application - Client and Server" on page 213
6. "Installing the Application" on page 214
7. "Running the Sample Application" on page 217

**Note:** Before starting Object Builder, ensure that your classpath includes one of the following:

## Notes:

1. Before starting Object Builder, ensure that your classpath includes one of the following:

**WIN** x:\MyProj

Specify x:\Myproj as the base directory for the project.

The procedures contained in this section assume that you have correctly set your classpath to include x:\MyProj before starting Object Builder and that you have started Object Builder.

**AIX** \$HOME/MyProj

Specify \$HOME/MyProj as the base directory for the project.

The procedures contained in this section assume that you have correctly set your classpath to include \$HOME/MyProj before starting Object Builder and that you have started Object Builder.

## Importing the Bean

The bean to import is ABeCashAcct from the paa.mysamples.cics.menu.acct package in one of the following directories:

**WIN** x:\MyProj

**AIX** \$HOME/MyProj

To import this bean:

1. Select the User-Defined PA Schemas folder from the Object Builder Tasks and Objects pane.
2. Open the pop-up menu for User-Defined PA Schemas, and select **Import Bean**. This opens the Import Procedural Adaptor Bean - wizard.
3. On this page:
  - a. Type paa.mysamples.cics.appc.acct.ABeCashAcctPA0 in the **Class Name** field.
  - b. Click **Next** to accept the remaining defaults and to continue to the Import Procedural Adaptor Bean — Names and Services page.
4. On this page:
  - a. Select **LU.6.2** for the Connector Type.
  - b. Click **Next** to accept the defaults and continue to the Procedural Adaptor Bean Key Selection page.
5. On this page:
  - a. Select the res\_type and the account\_ID properties from the **Properties** list box.
  - b. Click **>>** to move these associated key required to import the bean.
6. Click **Finish**.

The bean is imported into Object Builder. The **ABeCashAcctPAO** schema and its corresponding persistent object (**ABeCashAcctPAOPO**) are now in the tree view of User-Defined PA Schemas.

## Defining the ACashAcct Component

This exercise defines the objects required to create a component named "ACashAcct". For this component you will:

1. Create a new business object file
2. Define the business object
3. Connect the data object implementation to the persistent object
4. Define the managed object
5. Generate the code

### Creating the Business Object File

To create the ACashAcct business object file:

1. Select the User-Defined Business Objects folder.
2. Open the pop-up menu for User-Defined Business Objects, and select **Add File**. This displays the Name page of the Business Object File - wizard.
3. On this page:
  - a. Type ACashAcct in the **Name** field.
  - b. Accept the other defaults.
4. Click the **Finish** button.

The ACashAcct file is now under the User-Defined Business Objects folder.

### Defining the Business Object

After creating the new business object file, the business object needs to be defined. A fully-configured business object consists of the following:

- A business object interface
- An associated key
- An associated copy helper
- A business object implementation and data object interface

**Defining the Business Object Interface:** To create the ACashAcct business object interface:

1. From the User-Defined Business Object folder, select ACashAcct.
2. Open the pop-up menu for ACashAcct, and select **Add Interface**. This displays the Name and Attributes page of the Business Object Interface - wizard.
3. On this page:
  - a. Type ACashAcct in the **Name** field.
  - b. Click the **Next** button to continue to the Constructs page.
4. Click the **Next** button accept the defaults and to continue to the Interface Inheritance page.
5. Click the **Next** button to accept the defaults and to continue to the Attributes page.
6. Define the user-defined attributes.
  - a. Select Attributes from the tree view.
  - b. Open the pop-up menu for Attributes, and select **Add**. This displays the Add dialog.
  - c. In this dialog:
    - 1) Type res\_type in the **Attribute Name** field.

- 2) Select string as the **Type**. This displays the Size field.
- 3) Type 0 in the **Size** field
- 4) Click the **Add Another** button.
- 5) Repeat this step for each attribute of the ACashAcct interface, but click the **Refresh** button instead of the **Add Another** button at the end of the step.

**Note:** For balance, use type long. For account\_ID, acct\_type, and utilities, use string.

- d. Click the **Finish** button.

The ACashAcct interface is now under the ACashAcct file.

**Defining the Key:** To add the key:

1. From the User-Defined Business Object folder, select the ACashAcct interface.
2. Open the pop-up menu of ACashAcct, and select **Add Key**. This displays the Key - wizard.
3. Type ACashAcctKey in the **File Name** field.
4. Select the res\_type and the account\_ID attributes from the **Business Object Attributes** list.
5. Click the **>>** button to move the attribute to the **Key Attributes** list.
6. Click the **Finish** button.

The ACashAcctKey key is now under the ACashAcct interface.

**Defining the Copy Helper:** To add the copy helper:

1. From the User-Defined Business Object folder, select the ACashAcct interface.
2. Open the pop-up menu for ACashAcct, and select **Add Copy Helper**. This displays the Copy Helper - wizard.
3. Type ACashAcctCopy in the **File Name** field.
4. Click the **All>>** button to move the attributes from the **Business Object Attributes** list to the **Copy Helper Attributes** list.
5. Click the **Finish** button.

The ACashAcctCopy copy helper is now under the ACashAcct interface.

**Defining the Business Object Implementation and Data Object Interface:** To add the Business Object Implementation and Data Object interface:

1. From the User-Defined Business Object folder, open the ACashAcct interface.
2. Open the pop-up menu for ACashAcct, and select **Add Implementation**. This displays the Name and Data Access Pattern page of the Business Object Implementation - SmartGuide.
3. Type ACashAcctB0 in the **File Name** field.
4. Define the implementation.
  - a. Select the **Delegating** radio button from the **Pattern for Handling State Data** group box.
  - b. Ensure that the **Create a new one now** radio button is selected from the **Data Object Interface** group box. This option allows you to define the business object attributes that need to be preserved in the data object.
  - c. Deselect **390** in the Select deployment platform group box.
  - d. Click the **Next** button to continue to the Implementation Inheritance page.
5. Click the **Next** button to accept the defaults and to continue to the Implementation Language page.
6. Click the **Next** button to accept the defaults and to continue to the Attributes page.

7. Click the **Next** button to accept the defaults and to continue to the Methods page.
8. Click the **Next** button to accept the defaults and to continue to the Key and Copy Helper Selection page.
9. On this page:
  - a. Verify that the ACashAcctKey key is selected from the **Key** list.
  - b. Verify that the ACashAcctCopy copy helper is selected from the **Copy Helper** list.
  - c. Click the **Next** button to continue to the Handle Selection page.
10. Click the **Next** button to accept the defaults and to continue to the Attributes to Override page.
11. Click the **Next** button to accept the defaults and to continue to the Data Object Interface page.
12. Type ACashAcctDO in the **Data Object File Name** field.
13. Click the **All>>** button to move the attributes in the **Business Object Attributes** list to the **State Data** list.
14. Click the **Finish** button.

The ACashAcctBO business object implementation is now under the ACashAcct interface, and the ACashAcctDO data object interface is now under the ACashAcctBO business object implementation.

## Connecting the Data Object Implementation to the Persistent Object

To create the data object implementation and to connect the data object implementation to the persistent object:

1. From the User-Defined Business Object folder, select the ACashAcctDO data object interface.
2. Open the pop-up menu for ACashAcctDO, and select **Add Implementation**. This displays the Data Object Implementation - wizard.
3. Deselect **390** in the Select deployment platform group box and click **Next** to continue to the Behavior page.
4. Set the environment.
  - a. Set the **BOIM with any key** radio button from the **Environment** group box to indicate that the data object is part of a component installed in a business object application adaptor with instances being located by key objects.
  - b. Set the **Procedural Adaptors** radio button from the **Form of Persistent Behavior and Implementation** group box.
  - c. Click the **Next** button to continue to the Implementation Inheritance page.
5. On this page, verify that IPAAExtLocalToServer is selected as parents.
6. Click the **Next** button to continue to the Attributes page.
7. Click the **Next** button to continue to the Methods page.
8. Click the **Next** button to continue to the Key and Copy Helper page.
9. Click the **Next** button to accept the defaults and to continue to the Associated Persistent Objects page.
10. On this page:
  - a. Select Persistent Object Instances.
  - b. Open the pop-up menu for Persistent Object Instance, and select **Add**.
  - c. Type iABeCashAcctPA0P0 in the **Instance Name** field.
  - d. Click the **Next** button to continue to the Attributes Mapping page.

11. On this page:
  - a. Select *Res\_type* from the **Attributes** list.
  - b. Open the pop-up menu for *Res\_type*, and select **Primitive**.
  - c. Select *ABeCashAcctPAOPO.Res\_type* from the **Persistent Object Attribute** list.
  - d. Add 1-to-1 mappings for the other attributes in the **Attributes** tree view as you did for *Res\_type*.
  - e. Click the **Next** button to continue to the Methods Mapping page.
12. On this page:
  - a. Select insert from the **Special Framework Methods** list.
  - b. Open the pop-up menu for insert, and select **Add Mapping**.
  - c. Select *iABeCashAcctPAOPO.insert* from the **Persistent Object Method** list.
  - d. Add 1-to-1 mappings for the other methods, retrieve, update, delete, and setConnection in the **Special Framework Methods** tree view as you did for insert.
13. Click **Finish**.

The ACashAcctDOImpl data object implementation is now under the ACashAcctDO interface, and the ABeCashAcctPAOPO persistent object is now under the ACashAcctDOImpl data object implementation.

## Defining the Managed Object

To add the managed object:

1. From the User-Defined Business Object folder, select the ACashAcctBO business object implementation.
2. Open the pop-up menu for ACashAcctBO, and select **Add Managed Object**. This displays the Name and Services page of the Managed Object - SmartGuide.
3. Type ACashAcctM0 in the **File Name** field.
4. Select the **Transaction Service** radio button in the **Service to Use** group.
5. Deselect **390** in the Select deployment platform group box.
6. Click the **Next** button to accept the defaults and continue to the Implementation Inheritance page.
7. Click the **Finish** button.

## Generating the Code

To generate the application code:

1. From the User-Defined Business Object folder, select ACashAcct.
2. Open the pop-up menu for ACashAcct, and select **Generate** → **All**.

Code generation starts. Progress is indicated in the lower-left corner of the window.

## Creating Client and Server DLL Files

The defined objects need to be built into two separate DLL files.

- One that runs on the client and provides access to the business object interface, key, and copy helper.
- One that runs on the server and provides access to the managed object and the rest of the component.

The client DLL file needs to be defined before the server DLL file. When the server DLL file is defined, it needs to link to the client DLL file. After defining the objects that comprise each DLL file, these files can be built.

## Defining the Client DLL File

To add the client DLL file:

1. Select the Build Configuration folder.
2. Open the pop-up menu for Build Configuration, and select **Add client DLL**. This displays the Name and Option page of the Add Client DLL - wizard.
3. Type ACashAcctC in the **Name** field.
4. Check only the Applicable Platforms you want.
5. Click **Next** to continue to the Client Source Files page.
6. Click **All>>** to move the client source files to the **Items chosen** list.
7. Click **Finish**.

The ACashAcctC client DLL file is now under the Build Configuration folder.

## Defining the Server DLL File

To add the server DLL.

1. Select the Build Configuration folder.
2. Open the pop-up menu for Build Configuration, and select **Add Server DLL**. This displays the Name and Option page of the Server DLL - wizard.
3. Type ACashAcctS in the **Name** field.
4. Click only the Applicable Platforms you want.
5. Click **Next** to continue to the Server Source Files page.
6. Click **All>>** to move the server source files to the **Items chosen** list.
7. Click **Next** to continue Libraries to Link With page.
8. Select ACashAcctC from the **Items Available** list.
9. Click **>>** to move ACashAcctC.dll to the **Items Chosen** list.
10. Click **Finish**.

The ACashAcctS server DLL file is now under the Builder Configuration folder.

## Generating the Makefiles

To generate the makefiles to build the configuration:

1. Select the Build Configuration folder.
2. Open the pop-up menu for Build Configuration, and select **Generate** → **All** → **All Targets**.

The code generation begins.

## Packaging the Application

Packaging the application consists of the following procedures:

1. Creating the application family
2. Defining the application
3. Creating the container instance
4. Configuring the managed object
5. Generating the application

### Creating the Application Family

To add the application family:

1. Select the Application Configuration folder.
2. Open the pop-up menu for Application Configuration, and select **Add Application Family**. This displays the Name page of the Application Family - wizard.
3. Type ACashAcctApp in the **Name** field.
4. Click the **Finish** button.

The ACashAcctApp application family is now under the Application Configuration folder.

### Defining the Application

To add the Application:

1. Select the ACashAcctApp application family.
2. Open the pop-up menu for ACashAcctApp, and select **Add Application**. This displays the Name and Environment page of the Add Application - wizard.
3. Type ACashAcct in the **Application Name** field.
4. Click the **Finish** button.

The ACashAcct application is now under the ACashAcctApp application family.

### Creating the Container Instance

To add the new container instance:

1. Select the Container Definition folder.
2. Open the pop-up menu for Container Definition, and select **Add Container Instance**. This displays the Container - wizard.
3. Type ACashAcctContainer in the **Name** field.
4. Click the **Next** button to continue to the Workload Management Container page.
5. Click the **Next** button to continue to the Services page.
6. On this page:
  - a. Set the **Use PAA Transaction Services** radio button.
  - b. Click the **Next** button to continue to the Services Details page.
7. On this page, type APPC\_ACashAcct\_Server in the **Connection Name** field.
8. Select the **Throw an exception and abandon the call** radio button.

9. Click the **Finish** button.

The ACashAcctContainer container is now under the Container Definition folder.

## Configuring the Managed Object

To add the managed object for the Application:

1. Select the ACashAcct application.
2. Open the pop-up menu for ACashAcct, and select **Add Managed Object**. This displays the Configure Manage Object - wizard.
3. In this window:
  - a. Verify that ACashAcctMO is in the **Managed Object** field.
  - b. Click the **Next** button to continue to the Data Object Implementations page.
4. On this page:
  - a. Select Implementation.
  - b. Open the pop-up menu for Implementation, and select **Add**.
  - c. Select ACashAcctDOImpl ACashAcctDOImpl from the **Data Object Implementation** list.
  - d. Click the **Next** button to continue to the Container page.
5. On this page, select ACashAcctContainer from the **Name** list.
6. Click the **Next** button to continue to the Home page.
7. On this page, select BOIMHomeOfRegHomes from the **Home Name** list.
8. Click the **Finish** button.

The ACashAcctMO managed object is now under the Application Configuration folder.

## Generating the Applications

To generate the application family:

1. Select the ACashAcctApp application.
2. Open the pop-up menu for ACashAcctApp, and select **Generate**.

**Note:** If you do not have **InstallShield** installed on your system, Click the **Yes** button when the dialog concerning **InstallShield** is displayed.

When code generation completes, the Method Implementation pane contains the ACashAcctApp.ddl file. You can now close Object Builder.

## Building the Application - Client and Server

WIN

All imported and generated files are placed in the `x:\MyProj\Working\NT` directory (where `x:\MyProj` is the working directory when Object Builder was started).

1. Change directory to:

```
x:\MyProj\Working\NT
```
2. Type:

```
nmake -f all.mak cpp java
```
3. Everything in the sample application is built.

AIX

All imported and generated files are placed in the `$HOME/MyProj/Working/AIX` directory (where `$HOME/MyProj/Working/AIX` is the working directory when Object Builder was started).

1. Change directory to:  
`$HOME/MyProj/Working/AIX`
2. Type:  
`make -f all.mak cpp java`
3. Everything in the sample application is built.

## Installing the Application

Installing an application consists of:

1. Loading the application
2. Configuring the application

These procedures assume that you are currently logged on to DCE and that you are currently using the System Manager User Interface. If not, logon to DCE and start the System Manager User Interface.

## Loading the Application onto System Management

To install the ACashAcct server application:

1. Start the System Manager User Interface, if it is not already started.
2. Become an Expert user (**View** → **User Level** → **Expert**).
3. Expand Host Images, and select **<your host name>**.
4. From the pop-menu, select **Load Application**. This opens the Load Application dialog.

**WIN** Browse for and select ACashAcctApp.ddl (`x:\MyProj\Working\NT\ACashAcct\ACashAcctApp.ddl`).

**AIX** Browse for and select ACashAcctApp.ddl  
(`$HOME/MyProj/Working/AIX/ACashAcct/ACashAcctApp.ddl`).

## Configuring the Application with System Management

1. To configure the application:
  - a. Expand Available Applications, and select ACashAcct.
  - b. Open the pop-up menu for ACashAcct, and select **Drag**.
  - c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select Sample Configuration.
  - d. Open the pop-up menu for Sample Configuration, and select **Add Application**.
2. Configure the APPC connection.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → APPC Connection Installs, and select APPC\_ACashAcct\_Server.
  - b. Open the pop-up menu for APPC\_ACashAcct\_Server, and select **Edit** to open the Object Editor.
  - c. Click the **Main** tab.
  - d. Change the **Fully-qualified Local LU name** field to match the local LU6.2 LU that you will be using to communicate with your CICS/IMS system (for example, PAA01001).

- e. Change the **Fully-qualified Partner LU name** field to match the partner LU6.2 LU that you will be using to communicate with your CICS/IMS system (for example, CICS4).
  - f. Change the **Mode Name** field to match the mode name that you will be using to communicate with your CICS/IMS system (for example, LU62PS).
  - g. Change the **Remote Procedure Type** field to match the type of program with which you will be communicating (for example, CICS\_DPL or CICS\_DTP). The CICS\_DPL flavor appends eight bytes (converted to the target code page) that correspond to the CICS application to which the DTP program should EXEC CICS LINK.
  - h. Change the **Transaction Program Name** field to match the CICS transaction program that you will run (for example, BDPL or BDTP).
  - i. Change the **CICS Program Name** field to match the CICS program name that you will be running under the transaction program (for example, BECASHAC).
  - j. Change the **transaction type** field to be either optimistic or pessimistic. Pessimistic will talk initiate the conversation as sync-level 2 for the entire while optimistic will only talk sync-level 2 during the prepare and commit parts of the transaction.
  - k. Click the **OK** button to validate and accept the changes.
3. Define the server.
    - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select Sample Configuration.
    - b. Open the pop-up menu for Sample Configuration, and select **New** → **Server (free standing)**. This displays a new dialog box.
    - c. Type ACashAcctSvr as the name for the server.
    - d. Click the **OK** button. The ACashAcctSvr server is now under Server (free standing).
  4. Associate the application with the server.
    - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Applications, and select ACashAcct.
    - b. Open the pop-up menu for ACashAcct, and select **Drag**.
    - c. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Server (free standing), and select ACashAcctSvr.
    - d. Open the pop-up menu for ACashAcctSvr, and select **Configure Application**.
  5. Associate the iPAAServices with the server.
    - a. If the data directory has been refreshed, add the iPAAServices application to Sample Configurations, otherwise skip to step 5b:
      - 1) Expand Host Images → *myhost* → Application Family Installs → iPAAApplications → Application Installs, and select iPAAServices.
      - 2) Open the pop-up menu for IPAAServices, and select **Drag**.
      - 3) Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select Sample Configuration.
      - 4) Open the pop-up menu for Sample Configuration, and select **Add Application**.
    - b. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Applications, and select iPAAServices.
    - c. Open the pop-up menu for iPAAServices, and select **Drag**.

- d. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Sample Configuration → Server (free standing), and select ACashAcctSvr.
  - e. Open the pop-up menu for ACashAcctSvr, and select **Configure Application**.
6. Configure the server with the host.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations → Server (free standing), and select ACashAcctSrv
  - b. From the pop-up menu for ACashAcctSrv, select **Drag**.
  - c. Expand Host, and select your host.
  - d. From the pop-up menu for your server, select **Configure Server (free standing)**.
7. **Optional:** Enable security services for the server.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configuration → Sample Configuration → Server (free standing), and select ACashAcctSrv.
  - b. Open the pop-up menu for ACashAcctSrv, and select Edit, which opens the Object Editor.
  - c. In this notebook:
    - 1) Select the **Security Service** tab.
    - 2) Change the value for the **data system principal** field to the user ID that the server will use when connecting to the CICS system.
    - 3) Change the value for the **data system password** field to the password that the server will use when connecting to the CICS system.
    - 4) Change the value for the **security enabled** field from **no** to **yes**.
    - 5) Click the **OK** button. The changes are applied and the Object Editor closes.
8. **Optional:** Enable security services for the client.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configuration → Sample Configuration → Client Styles, and select *myClient*.
  - b. Open the pop-up menu for *myClient*, and select **Edit**, which opens the Object Editor.
  - c. In this notebook:
    - 1) Select the **Security Service** tab.
    - 2) Change the value for the **security enabled** field from **no** to **yes**.
    - 3) Click the **OK** button. The changes are applied and the Object Editor closes.
9. Activate the configuration.
  - a. Expand Management Zones → Sample Cell and Work Group Zone → Configurations, and select **Sample Configuration**.
  - b. Open the pop-up menu for Sample Configuration, and select **Activate**, which automatically starts the application server. Wait for the completion message in the Action Console window before continuing.

## Running the Sample Application

Before running this sample ensure that the IBM Transaction Server for CICS/NT is configured for the CICS region and a single Encina shared file system (SFS). For details, see Appendix B, “Installing the CICS DTP Sample Programs” on page 223.

To run the sample client application, complete one of the following procedures:

### WIN

1. Copy ACashAcctcli.mak and ACashAcctcli.cpp from:

```
x:\CBroker\samples\InstallVerification\PAA\Application\ACashAcctcli
```

TO:

```
x:\MyProj\Working\NT
```

2. Change directory to:

```
x:\MyProj\Working\NT
```

3. Enter:

```
nmake -f ACashAcctcli.mak
```

4. Enter:

```
Acashacctcli
```

### AIX

1. Copy ACashAcctcli.mak and ACashAcctcli.cpp from:

```
$HOME/samples/InstallVerification/PAA/Application/ACashAcctcli
```

TO:

```
$HOME/MyProj/Working/AIX
```

2. Change directory to:

```
$HOME/MyProj/Working/AIX
```

3. Enter:

```
make -f ACashAcctcli.mak
```

4. Enter:

```
Acashacctcli
```



---

## Appendix A. Installing the IVPs and CICS-ECI Sample

This appendix contains information on installing the CICS and IMS Installation Verification Programs (IVPs), installing the CICS-ECI sample, and the contents of the t3-trans subdirectory

---

### IVP Install Instructions

This section contains information on where to find IVP install instructions for the CICS-HOD IVP and the IMS-HOD IVP. This information may be useful for understanding how to install the CICS Transaction Server.

- For Windows NT, in the *IBM Transaction Server for Windows NT Installation Guide, Version 4* Chapter 6, entitled “Performing the Installation Verification Procedures” discusses how to load and run the IVP programs for CICS.
- For MVS, in the *IBM CICS Transaction Server for OS/390 CICS Installation Guide* Section 2.6 discusses installing and running the IVP jobs.
- For IMS, in *IMS/ESA Version 6 Install Volume 1*, the entire book contains information on installing and configuring the IVP sample. Chapter 11, entitled “Install/IVP Application,” discusses the sample IMS application.

---

### CICS-ECI Sample Install Instructions

To use the Component Broker CICS and IMS application adaptor CICS-ECI sample program, you must install the tier-3 portion of the sample to an IBM Transaction Server for NT CICS system. The IBM Transaction Server is available from your IBM representative or third-party reseller.

Perform the following procedure to install the CICS/NT and the tier-3 sample.

1. Install the Transaction Server for CICS/NT from the IBM Transaction Server compact disc and configure a single CICS region with a single Encina shared file system (SFS). See the *CICS Installation Guide* or the *CICS Quick Beginnings* for details.

**Note:** If you already installed the Transaction Server, you should create a new region for this sample.

2. Start the Encina SFS and SFS server, but do not start the CICS region. See the *CICS Administration Guide* for details.
3. Copy all files (except the DLL files) for the sample programs from the x:\CBroker\samples\InstallVerification\PAA\Businessobjects\CashAcct\t3-trans directory (where x:\CBroker is the directory where you installed Component Broker) to a subdirectory on the system running the CICS region (for example, x:\usercics).
4. Change directory to your samples directory (x:\usercics).
5. Run BOSSTOWN\_DEF\_TX to add the definitions to the CICS region and Encina SFS. This command required the following four parameters:
  - The name of the CICS region.
  - The directory where the files are located.
  - The name of the Encina SFS.
  - The short name of the Encina SFS.

These parameters are user-supplied configuration parameters, which you provided during the setup of the CICS region and the Encina SFS and are part of the script itself. During the first run, it is normal to receive delete errors, because these objects being deleted do not yet exist.

6. Create the appropriate listeners (LD stanza). See the *CICS Administration Guide* for details.
7. Create the transient data queue. The BECASHAC program uses a transient data queue for tracing purposes, and this queue must exist to run the program. To create this queue, perform the following procedure:
  - a. Right-click your CICS region.
  - b. From the pop-up menu, select **Resources** → **Transient Data Queue**.
  - c. From the menu bar, select **Transient** → **New**.
  - d. In the resulting dialog:
    - 1) Click the **General** tab.
    - 2) Enter BOSS in the **TDQ name** field and select a **Queue Type** of extrapartition.
    - 3) Click the **Extrapartition** tab.
    - 4) Enter a name under **Queue File** to specify the file where you want your transient data queue output to go. Select **line oriented** as the **Record type** and **250** as the **Record length**.
    - 5) Click the **Security** tab.
    - 6) Select **Public** as the **Resource level security key**.
    - 7) Click the **Permanent** button.
8. Copy the eight (8) DLL files from the  
x:\CBroker\samples\Install\Verification\PAA\Businessobjects\CashAcct\t3-trans directory to a directory in your PATH statement (for example, the lib subdirectory).
9. Start the CICS region.
10. Start the CICS terminal to the server and run the “BBCA” transaction to get a screen-based interface for the database.

You are now ready to run the Component Broker CICS-ECI sample.

---

## Content of the t3-trans Directory

The t3-trans directory contains the following files.

### **README**

Basically, the contents of this section.

### **BOSSTOWN\_DEF\_TX.CMD**

A DOS command script to setup the sample.

### **BSCASH.MAP**

A map file for BECASHAC. This file is required at run time.

### **BSCASH**

A map file for BECASHAC. This file is required at run time.

### **BBCASHAC.IBMCOB**

CICS BMS program that accesses BECASHAC from a CICS terminal. This file is required at run time.

**BECASHAC.IBMCOB**

The CICS backend ECI program for the Cash Account. This file is required at run time.

**BBCASH.BMS**

The BMS source file for the BECASHAC program.

**BBCASHAC.CCP**

The source file for BBCASHAC.IBMCOB.

**BECASHAC.CCP**

The source file for BECASHAC.IBMCOB.

**Library Files**

These files should be copied to a directory in your PATH statement (for example, the lib subdirectory).

- ARZLITE.DLL
- IWZODBC.DLL
- IWZRFBTR.DLL
- IWZRFSTL.DLL
- IWZRFVSA.DLL
- IWZRLIB.DLL
- IWZRLIBM.DLL
- IWZRMSTL.DLL

**Note:** The BS\* and BB\* files are required for the CICS terminal interface to the CashAcct database table, which is useful for debugging.



---

## Appendix B. Installing the CICS DTP Sample Programs

This appendix provides procedures to install, set up, and configure the two CICS DTP sample programs on a CICS region. Procedures included are:

- “Installing on CICS/ESA”
- “Installing on a CICS Transaction Server (NT or AIX)”

---

### Installing on CICS/ESA

1. Transfer the files from your Component Broker installation to an appropriate MVS system using, for example, ftp or the send option of IBM Personal Communications. Put both sets of programs as members of a Cobol source library dataset.
2. Translate, compile, and link-edit the programs using standard CICS jobs. The programs must be compiled with the TRUNC(BIN) compiler option. Make sure the resulting executables end up in a load library that can be accessed by CICS.
3. Logon to the CICS region and add definitions for the transactions and programs using the CEDA transaction as follows:

```
CEDA DEF PROGRAM(CICSDPL) GROUP(BOSSTOWN) LANG(COBOL)
CEDA DEF PROGRAM(CICSDTP) GROUP(BOSSTOWN) LANG(COBOL)
```

```
CEDA DEF TRANS(BDPL) GROUP(BOSSTOWN) PROGRAM(CICSDPL)
CEDA DEF TRANS(BDTP) GROUP(BOSSTOWN) PROGRAM(CICSDTP)
```

Add descriptions as appropriate. Install the new definitions with the CEDA transaction as follows:

```
CEDA IN GROUP(BOSSTOWN)
```

4. Add the following DCT entry for the CICS region:

```
BOSS  DFHDCT TYPE=INTRA,
        DESTID=BOSS,
        DESTFAC=FILE,
        DSCNAME=BOSSMSG,
        TRIGLEV=0
```

On earlier versions of CICS/ESA this would be done by compiling the above entry in a load library. On later version of CICS such as CICS Transaction Server, CEDA can be used for adding DCT entries.

The two programs are now ready to use.

---

### Installing on a CICS Transaction Server (NT or AIX)

1. Copy the two files to a subdirectory on the system running the CICS region (for example, x:\USERCICS).
2. Change directory to the CICS directory (x:\USERCICS).
3. Compile the programs:

WIN Run the supplied command file:  
bosstown\_dtp.cmd

AIX Issue the following commands:

```
- cicsctl -e -d -libmcob cicsdtp.ccp
- cicsctl -e -d -libmcob cicsdpl.ccp
```

4. Add the definitions to the CICS region:

**WIN** Run the supplied command file:

```
bosstown_def_dtp.cmd
```

**AIX** Issue the following commands:

```
cicsadd -c pd -r %r -P "CICSDTP" GroupName="BOSSTOWN" PathName="x:\USERCICS\CICSDTP"
cicsadd -c pd -r %r -P "CICSDPL" GroupName="BOSSTOWN" PathName="x:\USERCICS\CICSDPL"
```

```
cicsadd -c td -r %r -P "BDTP" GroupName="BOSSTOWN" ProgName="CICSDTP"
cicsadd -c td -r %r -P "BDPL" GroupName="BOSSTOWN" ProgName="CICSDPL"
```

```
cicsadd -c tdd -r %l -P "BOSS" GroupName="BOSSTOWN" DestType=extrapartition
      ExtrapartitionFile="x:\usercics\bosstrace.out" RecordType=line_oriented
      RecordLen=250
```

where *%r* is the name of the CICS region.

5. Start (or restart if already running) the CICS region.

The two programs are now ready for use.

---

## Appendix C. Interchange Files within VisualAge for Java

With VisualAge for Java, there is more information required to represent the visually built navigations than .java and .class files. If you only export the .java and .class files from VisualAge for Java and import these files into another version of VisualAge for Java, some required information for visual representation is lost. To correctly populate all the information required, you must create a repository (.dat) file from the VisualAge for Java sample. You may need to Version or Edition the appropriate projects, packages, and classes to create this repository file.

To export a file:

1. Select the package or project you want to export.
2. From the VisualAge for Java Workbench menu, select **File** → **export**.
3. Select an export destination by clicking the **Repository** radio button and then clicking **Next**.
4. In the **Directory** text entry field, type the file name of your repository file (e.g. c:\filename.dat) and then click **Finish**.

| When you need to import a package that was exported or import a package that was previously created, perform the following steps:

1. Create a new project (if you are importing a brand new project).
2. You can only import interchange (.dat) files directly into the repository. From the VisualAge for Java Workbench menu, select **File** → **import**.
3. Select the interchange file, and click the **Next** button.
4. Type the name of the interchange file, and click the **Update** button.
5. Set the radio button for package or project depending on whether you exported a package or a project.
6. Select the package you want to import, and click **Finish**. This adds the package to the repository.
7. Open the pop-up menu for the project, and select **Add Package**, which opens the Add Package - SmartGuide.
8. For this SmartGuide:
  - Select **Add package(s) from the repository** check box.
  - Click **Browse**.
  - Browse for and select the correct packages.

You should see the package you just added to the repository under the specific project. If you do not see this package, it could be associated with another project. If it is associated with another project, go to that project and bring in the new version.



---

## Appendix D. IMS Configuration

This appendix contains useful information for setting up IMS to be accessed from Component Broker.

Two idle IMS Message Processing Regions (MPRs) must be available for use by the IMS transactions used by that DO. If a Component Broker application uses 10 DOs that are in IMS, then 20 IMS regions (MPRs) are used by the Component Broker-initiated IMS transactions. Failure to have enough MPRs available could cause some Component Broker DO requests to remain queued for execution within IMS, preventing successful execution of the Component Broker application.

The IMS system definition (IMS gen stage 1) must include a TRANSACT macro defining the transaction code used for the Component Broker DO request and this must have the MAXREGN and PARLIM operands specified appropriately. MAXREGN must be greater than twice the number of DOs in IMS for the Component Broker application and PARLIM must equal 0. You can also specify MAXREGN=0 which allows an arbitrary number of regions to be used. The SCHDTYPE=PARALLEL operand must also be specified on the corresponding APPLCTN definition.

The Component Broker application uses a distributed syncpoint. This causes the IMS transactions representing the DOs to be idle but still scheduled in their dependent regions until the commit point is reached for the entire application. The requirement to have a sufficient number of IMS MPRs available stems from the need for all IMS transactions that are initiated by the single Component Broker application instance to be concurrently executing. Having too few MPRs causes the Component Broker application instance to never complete, because it must wait for a new MPR to start executing the DO. The application programs making up the DO transactions in IMS are very light in resource usage compared to most IMS transactions -- they are waiting much of the time. These new MPRs that must be added to the IMS system (or existing ones reserved for use by the Component Broker applications) are less of a resource impact on the performance and capacity of the IMS system than other MPRs. Their primary impact is in the use of Real and Virtual storage with much less impact on CPU and IO workload. Storage tends to be plentiful on newer computers, thus minimizing the impact of the additional storage being used.

The IMS transaction program implementing the Data Object access to IMS controlled data must be defined with a suitable PROCOPT in the PSBGEN and must issue suitable locking for the data segments being processed. This is very important for distributed sync point to work correctly as the IMS data retrieval and data update transactions need to sync together. IMS has added special processing to recognize all the transactions that are part of the same distributed logical unit of work and automatically propagates the locks from the first transaction to the second one in order to have a consistent unit of work and to prevent deadlocks. This simplifies the application program responsibility for data access control but still leaves two responsibilities:

- The processing option (PROCOPT parameter in PSBGEN)
- Locking of data by the retrieval transaction to prevent other updates

The exclusive processing option (PROCOPT=E) must never be specified for Component Broker transactions. This option specifies that only one transaction accessing the database can run at a time, thus preventing the second Component Broker transaction from being concurrently processed and reaching the coordinated sync point. Almost always you should specify PROCOPT=A for both transactions (this allows all update, insert, and delete activity). Using other processing options, particularly PROCOPT=G0x can result in deadlock or waiting for locks held by other (non-Component Broker) transactions, and should be done only after careful analysis of the impact. It is valid in some special situations but usually has undesirable results.

The first transaction doing the IMS data retrieval should lock the data for update by issuing a GHU or GHN call. The update is done by the second Component Broker update transaction and the locks obtained

by the GHx call are transferred to this second transaction as part of the special IMS support for distributed sync points. This is a new function in IMS and is used only for transactions using distributed sync point (LU 6.2 sync level of syncpt). Other transactions doing data retrieval continue to be prohibited from using the hold form of get calls if you want to avoid lock interference and its adverse performance impact.

There is a possibility of deadlock or lock conflict for inserted data. This varies widely by database type and definition (both logical and physical). In general, inserting data next to updated data is safe (the locking for update will also implicitly lock the insert) but conflicts in free space access can still occur. Normal IMS tuning and lock conflict diagnosis techniques apply with no special considerations needed for Component Broker.

---

## Appendix E. Notices

This information was developed for products and services offered in the U.S.A. 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 that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right 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 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  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

**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 information 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 information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
Department LZKS  
11400 Burnet Road  
Austin, TX 78758  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

---

## Trademarks

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

CICS  
AIX  
DB2  
IBM  
MVS/ESA  
OS/2  
PowerPC  
VisualAge

AFS and DFS are trademarks of Transarc Corporation in the United States, or other countries, or both.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Oracle and Oracle8 are registered trademarks of Oracle Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.





Part Number: C092703

Printed in U.S.A.

