CA Gen

Distributed Processing - Enterprise JavaBeans User Guide

Release 8.5

CA Technologies Product References

This document references the following CA Technologies products:

- CA Gen

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introduction

This guide describes the CA Gen Enterprise JavaBeans Generation and Assemble Option. This option allows you to generate, build, and assemble your Server Procedure Steps as Stateless Session Enterprise JavaBeans (EJB).

This chapter provides an overview of the Java Platform, Enterprise Edition 5 (Java EE) technology and how it is used by CA Gen. Later chapters provide task-related information and reference materials.

Java Enterprise Edition Technology

The Java EE is a related set of specifications that define the entire Web-enabled application, from the browser to the server and all the components in between.

The Java EE Application Model defines a multi-tiered distributed application:

- The Web browser is used for the application user interface.
- A middle tier that may implement the application work flow logic.
- An Enterprise Information Systems tier that may implement the application business rules and database operations.

The Java EE specification is a collection of several related specifications. The following is a partial list of specifications that are encompassed by the Java EE specification:

- Java Platform, Standard Edition (Java SE)
- Common Annotations for the Java Platform
- Java Server Pages (JSP)
- Java Servlet
- Enterprise JavaBeans (EJB)
- Java Database Connectivity (JDBC)
- Java Naming and Directory Interface (JNDI)
- Java Transaction API (JTA)
- Java API for XML-Based Web Services (JAX-WS)

Since these specifications are based on Java technology, Java EE applications are hardware platform and operating system independent.

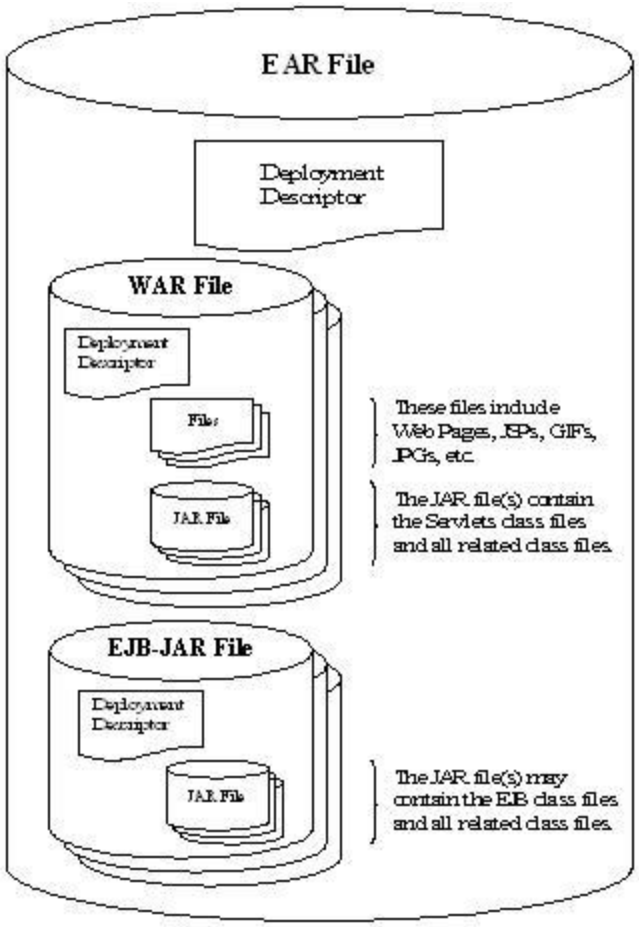
Java EE Terminology

A number of concepts and terms have been introduced by Java EE. The following list defines these items:

- Application Server—An integrated set of software that provides several functions including a Web Server, JSP Server, JNDI Server, and an EJB Server.
- Application Assembly—The process of collecting the various parts of the Java EE application (Web pages, JSPs, EJBs, and so on) and placing them into WAR and/or EAR files for deployment onto an application server.
- Application Deployment—The process of distributing the various parts of the Java EE application (Web pages, JSPs, EJBs, and so on) to the appropriate servers within the application server.
- JAR File—Java Archive. A single compressed file that usually contains many files of different types. Compression saves disk space and network bandwidth.
- Deployment Descriptor—An XML file within a WAR, EJB-JAR, or EAR file that contains the description of the contents of the file. It is used by the application server deployment tool.
- WAR File—Web Archive. A specialized JAR file that contains the portion of the application that will be deployed to a Web Server. The WAR file includes a deployment descriptor, plus all the files (the Web pages, JSPs, Java Servlets, and so on) destined for the Web Server.
- EJB-JAR File—A specialized JAR file that contains the portion of the application that will be deployed to an EJB Server. The EJB-JAR file includes a deployment descriptor, plus all files needed to implement the EJB(s).
- EAR File—Enterprise Archive. A specialized JAR file that usually contains the entire Java EE application. It includes a deployment descriptor, as well as WAR and EJB-JAR files. The deployment descriptor describes the WAR and EJB-JAR files, plus any security and database information specific to the application.
- Resource Adapter—A system-wide driver used by Java EE applications to access resources, such as a database.
- Web Service—A software system designed to support interoperable machine-to-machine interaction over a network.

EAR File Contents

The following diagram illustrates the relationship between JARs, EJB-JARs, WARs, and an EAR file:



CA Gen Applications Before Java Generation

CA Gen has evolved from an application development tool that generates block mode and batch applications into one that generates client/server applications. Client/server applications are divided into the following elements:

- GUI—The graphical user interface.
- Client Procedure Steps—Responsible for handling the GUI events and performing some of the application logic, usually related to the flow of the application, and can include database accesses. Utilizing middleware, Client Procedure Steps invoke Server Procedure Steps to perform the balance of the application's functionality.
- Server Procedure Steps—Responsible for executing the bulk of the business logic and database accesses. Server Procedure Steps do not have a user interface. If allowed by the TP monitor, a Server Procedure Step can invoke other Server Procedure Steps.

For Windows-based GUI applications, the GUI and Client Procedure Steps are generated in the C language for execution on a Microsoft Windows platform. The Server Procedure Steps are generated in COBOL for execution on the IBM mainframe, or in the C language for execution on Microsoft Windows, selected Linux, UNIX, or NonStop platforms.

The GUI Clients can communicate with the servers using one of many communications runtimes, including:

- The CA Gen Client Manager
- TCP/IP
- Oracle Tuxedo (Tuxedo)
- WebSphere MQ
- ECI
- Web Services

Note:

- Not all servers support all the above communications methods.
- For more information about cooperative flows, see the *Distributed Processing – Overview Guide*.

Java Generation

The design goal of the Java code generators is to generate a complete Java EE application from a CA Gen Client/Server model without modification. In practice, you may want to add Java-specific information to your models, such as the Java package names. The generated Java code is designed to be compliant with the Java 2 language specification, ensuring portability to any compliant Java Virtual Machine (JVM).

An action diagram is generated as a class whose name is the CA Gen member name. All import, export, and local views are generated as separate classes. For example, the action block DEMO_AB would be generated as the class DEMOAB. The import, export, and local views would be generated as the classes DEMOAB_IA, DEMOAB_OA, and DEMOAB_LA, respectively. An additional set of Java files are generated by the Window Manager and Server Manager to present the import and export views in a more object-oriented manner and wrapper classes to move the data into and out of the _IA and _OA classes.

JDBC is used for all database accesses in the generated Java code.

Java Runtime

The generated Java code is supported by a CA Gen Java runtime. The runtime is designed and written in Java to ensure performance of the generated applications. For example, the runtime caches objects for reuse whenever possible. In the EJB environment, the runtime uses the facilities provided by the EJB Container to cache objects such as resource management, transaction control, and flow control.

Supported Features

The generated Java code is designed to be functionally equivalent to the generated C and COBOL code. Accordingly, the generated Java code supports:

- CA Gen Referential Integrity
- DBMS-enforced Referential Integrity
- Decimal precision to 18 digits with the use of the Java BigDecimal class for appropriately marked attributes
- All six digits in the microsecond field of a CA Gen timestamp
- Debugging with the CA Gen Diagram Trace Utility
- All relevant user exits and a number of new ones
- Generation of external action blocks

Multiple Database Support

CA Gen Java applications use JDBC and are designed to allow access to one or more databases within a given procedure step. Clients wishing to use this feature must create a database resource for each database to be accessed by the procedure step.

In contrast, CA Gen C applications can access only one database for a given procedure step. This is due to the use of the default database handle for all embedded SQL statements.

Commits and rollbacks for multiple database connections may or may not be performed using a two-phased commit facility. In the Java Web Client environment, two-phased commit is supported only when the deployment options for using JDBC DataSources and JTA transactions are set and properly configured. In the EJB environment, two-phased commit is supported only when utilizing XA-compliant JDBC and JTA drivers.

Java Web Client Generation

CA Gen can generate clients for the Web and Java environment. The application GUI is generated as HTML and JavaScript, and the Client Procedure Steps are generated as Java with a small amount of Java ServerPages code. The Client Procedure Steps generated as Java Web Clients are allowed to invoke Server Procedure Steps generated as C, COBOL, Java code implemented as EJBs.

The Client Procedure Steps may interact with C or COBOL generated Server Procedure Steps using the following communication runtimes:

- TCP/IP
- WebSphere MQ
- ECI
- Tuxedo

Note: For more information about cooperative flows, see the *Distributed Processing – Overview Guide*.

EJB Server Generation

CA Gen can generate Server Procedure Steps as Stateless Session Enterprise JavaBeans. These EJBs are designed to be compliant with the EJB specification.

Taking advantage of built-in support for Web Services in Java EE platform, CA Gen can generate Server Procedure Steps as Stateless Session Enterprise JavaBeans with Web Service annotations.

To support the CA Gen Transaction Retry feature, the generated Server Procedure Steps default to managing their own transactions using the Java Transaction API (JTA). The CA Gen Transaction Retry feature reduces the number of application abends by automatically rolling back and retrying the Server Procedure Step logic in the event of a database deadlock or timeout.

Client-to-Server Flows

Java Web Client and Java Proxy Clients

CA Gen Java Web Clients and CA Gen Java Proxy Clients communicate with CA Gen EJB Servers using the CA Gen Java RMI Communications Runtime. Rather than using the CA Gen Common Format Buffer, the CA Gen Java RMI Communications Runtime uses the data interchange and remote invocation mechanisms native to Java. Specifically, the runtime serializes the import and export views and invokes the target EJB using the Remote Method Invocation (RMI) feature of Java.

CA Gen Java Web Clients and CA Gen Java Proxy Clients communicate with CA Gen EJB Web Service Servers using the CA Gen Java Web Services Middleware Runtime. The CA Gen Java Web Services Middleware Runtime uses SOAP. Specifically, the runtime converts the import and export views as XML and communicates with the target EJB Web Service using HTTP.

ASP.NET Internet clients and .Net Proxy Clients

CA Gen ASP.NET Internet clients and .Net Proxy Clients communicate with CA Gen EJB Web Service Servers using the CA Gen .Net Web Services Middleware Runtime. The CA Gen .Net Web Services Middleware Runtime uses SOAP. Specifically, the runtime converts the import and export views as XML and communicates with the target EJB Web Service using HTTP.

MFC GUI Clients, C and COM Proxy Clients

The CA Gen C-language GUI Clients as well as the CA Gen C and COM Proxy Clients communicate with CA Gen EJB Web Service Servers using the CA Gen C Web Services Middleware Runtime. The CA Gen C Web Services Middleware Runtime uses SOAP. Specifically, the runtime converts the import and export views as XML and communicates with the target EJB Web Service using HTTP.

The CA Gen C-language GUI Clients as well as the CA Gen C and COM Proxy Clients cannot communicate directly with EJB Servers using Java RMI. Rather, these clients must use one of two facilities to convert their server requests to use the CA Gen Java RMI Communications Runtime. These facilities are defined below:

- CFB Server—A Java program that can run on any Java Platform in the network. Client systems connect to the CFB Server using the TCP/IP Communications Runtime. The CFB Server converts the client request into a Java RMI call. The CFB Server requires a local copy of the target EJB portable interface definition files.
- C to Java RMI Coopflow—Performs the conversion and Java RMI call on the client system. Accordingly, the client system must have a Java Virtual Machine installed and a local copy the target EJB portable interface definition files.

Note: For more information about cooperative flows, see the *Distributed Processing – Overview Guide.*

In client-to-server flows, the client does not pass its transaction context to the server. Therefore, CA Gen generated EJB Servers start its own transaction context.

Handcrafted Clients

Hand-written Java applications may interoperate with generated CA Gen EJB Servers without utilizing a CA Gen-generated Proxy. The interface of the CA Gen-generated EJB Server will perform all required data validations prior to executing the target procedure step.

Server-to-Server Flows

In server-to-server flows, a CA Gen generated EJB can invoke another CA Gen generated EJB using the Java RMI Communications Runtime. The calling EJB can be in the same EJB Server as the target EJB Server, or it can be located elsewhere on the network (as supported by the EJB Servers). The target EJB can start its own transaction context, or it can extend the transaction context of the calling EJB as specified in the CA Gen model.

A CA Gen generated EJB can also invoke another CA Gen generated Web Services server using the new Java Web Services Middleware Runtime. The calling EJB and the target Web Service do not need to be co-located in the same JVM. The target Web Service cannot extend the transaction context of the calling EJB as specified in the CA Gen model.

Further, a CA Gen generated EJB can invoke a CA Gen generated C or COBOL server using one of the existing communications runtimes (TCP/IP, WebSphere MQ, ECI, or Tuxedo).

Distributed transactions are not supported when servers are executing within different TP systems.

Non-CA Gen EJBs must be called through External Action Blocks.

Limitless Views

There are no design limits for flows between CA Gen Java Web Clients and CA Gen EJB Servers when using the CA Gen Java RMI Communications Runtimes. Likewise, there are no size limits for server-to-server flows between generated EJB components using the Java RMI Communications Runtimes.

Similarly, there are no design limits for flows between CA Gen Java Clients and CA Gen EJB Web Service Servers when using the CA Gen Java Web Service Middleware Runtime. Likewise, there are no size limits for server-to-server flows between generated EJB servers and Web Service Servers using the Java Web Service Middleware Runtime.

Note: Some application servers may still impose their own limits. For more information, see your application server documentation.

References

CA Gen supports three types of references: EJB References, Local References, and Resource References.

- An EJB Reference provides a shortcut to an enterprise bean that is accessed through its remote business interface. Some application servers require EJB references while others require the deployer create a cross-reference XML file to help resolve the EJB references. When an application contains no EJB modules, no EJB references are needed.
- A Local Reference is a high performance mechanism for quickly transferring data between the EJB and its caller. As the name implies, the EJB and its caller must be in the same JVM. The performance improvement originates from the elimination of the serialization and deserialization of classes being passed between the EJB and its caller.
- A Resource Reference provides a shortcut through which an application can look up JDBC pooled connections maintained by the application server. Some servers require resource references while others require the deployer create a cross-reference XML file to help resolve the resource references.

Restrictions when Targeting EJB Components

CA Gen generated applications running as EJBs are subject to certain limitations. The following list describes the types of functions that cannot be executed in the EJB environment:

- All file I/O functions
- Creation of message boxes
- Creation of new threads

Additionally, asynchronous Action Diagram statements are not supported.

Supported Application Servers

For a complete description of the application servers supported by CA Gen, see the Technical Requirements document located at Customer Support.

Model Generation Targeting Java

The following sections describe the process of generating a CA Gen model in Java and deploying it on an application server.

Generation

The generation of Java Web Clients and EJB Servers can take place on the CA Gen Workstation Toolset or the Client/Server Encyclopedia (CSE). If the generation takes place on a CSE, the remote files must be moved to a supported Windows workstation to be built.

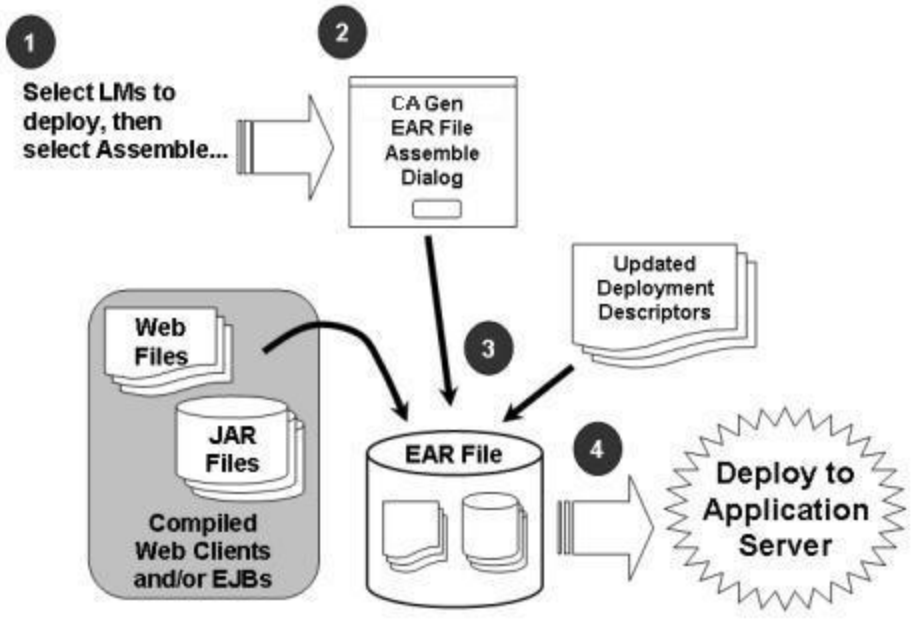
Build

A Java-generated application can only be built on a supported Windows system using the CA Gen Build Tool.

After all the CA Gen Load Modules have been successfully built, the application is ready for assembly and deployment.

Assembly and Deployment

During assembly and deployment, the files that make up a CA Gen Java EE application are placed into an EAR file. The following diagram shows how this is accomplished.



1. After selecting the CASCADE and Load Modules to be assembled into an EAR file, select Action, Assemble or click the Assemble toolbar icon to open the EAR File Assemble Details dialog.

   Note: You must not select the DDL module.

2. The EAR File Assemble Details dialog allows you to specify the properties of the EAR file, including its name, the targeted application server, and so on.

   If a Web Client has been selected for deployment, you can specify the WAR file properties, including its name and default context.

   Note: For a detailed explanation of this dialog, see the *Build Tool User Guide*.

   After the dialog is complete, click OK.

3. The Build Tool executes a script that performs the following tasks:

   - The files of the selected Web Clients, EJBs, or both are placed into the EAR file.
   - The generated deployment descriptors are updated with information from the EAR File Assemble Details dialog.

After completion, the EAR file is ready for deployment on your application server. Use the deployment facilities native to your application server to perform this task.

Testing Recommendations

Java-generated applications should be deployed to a small scale application server on a workstation or similar system for local testing. If Diagram Trace is to be used, it is recommended that it take place in a single user environment.

COBOL Application Considerations

This section addresses retargeting COBOL generated applications to either C, C#, or Java. COBOL has the capability of storing and manipulating numbers with precision. The native C, C#, or Java data types have limitations in either the number of digits or their precision. For example, most numbers with decimal digits will be generated as a double, which has a limit of 15 digits of precision and can lose precision when used in some mathematical operations.

CA Gen has the capability of generating an alternative data type that can achieve 18 digits of precision. In C, CA Gen uses a character array and a special software library. In C#, CA Gen uses the decimal type from the system package. In Java, CA Gen uses the BigDecimal class from the java.math package.

Using these alternative data types consumes more CPU resources, as it uses a software library to perform the mathematical operations, whereas, the double data type uses the hardware instructions to perform the same mathematical operations.

CA Gen does not generate these alternative data types automatically. Rather, you should examine your data model and determine which attributes should be generated in this alternative form and set the appropriate option.

After an attribute has been identified, its properties dialog should be displayed by either double-clicking on the attribute or selecting the attribute and then choosing Properties from the Detail menu. To instruct CA Gen to generate the alternative data type, select the Implement in C, C#, and Java with decimal precision check box.

DBMS Considerations with Decimal Precision

For Java generated code, CA Gen implements all attributes marked with decimal precision as a BigDecimal class and passes them to the JDBC driver. It is the responsibility of the driver to accurately store and retrieve the application data.

Unicode Considerations

Unicode is a method for encoding characters that allows one application to process characters and strings from most of the languages of the world. Nearly one million unique characters can be represented in Unicode. Unicode is neither a single byte character system (SBCS) nor a double byte character system (DBCS).

The C# and Java languages support Unicode only for characters, literals, and strings represented in code. This is true for all C# and Java applications, not just those generated by CA Gen.

It is possible for users of C# or Java generated applications to provide input data that cannot be stored on a database that is not configured to store Unicode.

Customers should evaluate the advantages and disadvantages of converting the applications database to Unicode. If the database is not converted, C# and Java applications could submit unrecognized character that cannot be stored correctly. Alternatively, customers may choose to use only C and COBOL applications with a database that is set to store DBCS characters.

This creates challenges when you attempt to use C# or Java code to access databases. Most existing databases store characters in a codepage that represents a subset of Unicode characters. Thus, many of the characters processed by the C# or Java application cannot be represented on the database. Unsupported characters are usually replaced with a 'sub character' or a '?' depending upon the database translation algorithm.

Converting existing databases to Unicode is possible with most implementations and would allow all characters entered by a user and processed by the code to be stored. However, converting a database to use Unicode could require changes to existing applications and in the way they access the database.

For a CA Gen application that creates a new database, CA suggests that the new database always support Unicode encoding. There are two major Unicode encoding schemes supported by most databases: UTF-8 and UCS-16. UCS-16 Unicode Coding Sequence 16-bit is the simplest because every character in the database is represented as an unsigned 16-bit number, which is the same representation as characters in a C# and Java application. UCS-16 uses greater amounts of storage when most of the characters represented in a database are indicated by code point numbers less than 128 (0x80). UTF-8 (Unicode Transfer Format 8-Bit) solves this problem because all code points less than 128 require only a single 8-bit byte of storage (code points between 128 and 2047 require two bytes, and code points between 2048 and 65535 require 3 bytes, and so on).

Further, a database created to support UTF-8 makes more assumptions about the type and size of characters in each text column. For example, a text column that is fixed ten characters (bytes) in length correctly stores ten characters when all ten code points are less than 128. But any higher character code points would require more than ten bytes in the column. When the type of data is unknown, all character columns in a UTF-8 database should be defined as variable and three times the length of the maximum string written by the C# or Java application (in this example, 3*10 or thirty).

Chapter 2: Software Installation

This chapter describes the tasks that should be performed before, during, and after the installation of CA Gen, with the intent of generating Server Procedure Steps as EJBs. This chapter also describes the configuration of third-party software.

Prerequisite Software

The following are the third-party and CA software products that are used with the CA Gen EJBs.

Construction Workstation

The following are the software products that must be present on the Windows system being used to build, assemble, and deploy the CA Gen generated EJB Servers.

- Windows Operating System
- Java SE (also known as JDK). It contains the Java compiler and tools such as the Java Archive (JAR) utility used during the build of the EJBs and the EAR file.
- Java EE SDK—Java EE 5 is a free integrated development kit that you can use to build, test, and deploy Java EE 5-based applications.

Note: For the supported versions of software, see the *Technical Requirements* document.

Client/Server Encyclopedia

The Client/Server Encyclopedia has no special requirements for EJB generation.

Runtime Environment

The CA Gen-generated EAR files are designed to deploy to any application server that conforms to the Java EE 5 specification.

Note: For the supported versions of software, see the *Technical Requirements* document.

CA Gen Installation

This section explains the installation of the CA Gen software with the intent of generating EJB Servers.

Windows Workstation

This section explains the pre-installation procedures and the tools to install CA Gen.

Pre-Installation Procedures

The following tasks should be performed before installing CA Gen:

Follow these steps:

1. Obtain the CA License file for your system.

   Note: For more information, see the *Distributed Systems Installation Guide*.

2. Install the Java SE kit.

3. Test the default version of the JVM. Follow the steps below:

   a. Open a Command Prompt window.

   b. Enter the following command:

      ```
      java -version
      ```

   c. Examine the output. If text similar to the following is displayed:

      ```
      The name specified is not recognized as an internal or external command,
      operable program or batch file.
      ```

      This indicates that Java (the JRE, JDK, or Java SE kit) is not in the environment variable PATH. To correct this issue, add the disk and directory name of the Java SE's bin directory to the PATH variable.

      If text similar to the following is displayed:

      ```
      java version "#.#.#_##"
      Java(TM) SE Runtime Environment
      ```

      A version of Java has been installed and is ready to use.

      For the supported versions of software, see the *Technical Requirements* document. If the correct version is not displayed, check the environment variable PATH. In most cases, an older version of Java (the JRE, JDK, or Java SE kit) appears before the desired version.

4. Install Java EE SDK.

Tools to Install

During the CA Gen Custom Installation, a list of available CA Gen components is displayed. The components that should be installed depend upon the type of work to be performed on the Windows workstation system.

You can generate, build, and assemble the application's EAR file from a Windows system. Or, you can generate EJBs on a Client/Server Encyclopedia, transfer the remote files to a Windows system, then build and assemble the application's EAR file. The following table lists the components of EJB creation:

Component	Notes
Workstation Construction Toolset	Provides the generators and Windows Build Tool needed to create a generated application.
Implementation Toolset	Provides the tools needed to build EJBs that were generated elsewhere.
Encyclopedia Construction Server	Can generate EJBs. However, they must be built by the Workstation Construction Tool or Windows Implementation Toolset.
Enterprise JavaBeans	A separately purchased option that adds the ability to generate EJBs to the Workstation Construction Tool or the Encyclopedia Construction Server.
Java RMI Middleware	This component is needed only at runtime. The cooperative flow runtime that used Java RMI to allow generated clients to interoperate with EJB Servers.
Java Web Services Middleware	This component is needed only at runtime. This is the cooperative flow runtime that uses Java SOAP Web Services to allow generated clients to interoperate with generated Web Service Servers.
CFB Converter Services to EJBs	This component is needed only at runtime. It provides support for non-Java clients such as GUI C or C and COM Proxies to interact with CA Gen EJBs. Must also install the Java RMI middleware.

Client/Server Encyclopedia (Windows or UNIX)

This section explains the pre-installation procedures and the tools to install the CA Gen Client/Server Encyclopedia.

Pre-Installation Procedures

Before you install CA Gen, be sure to obtain the CA License file for your system.

Note: For more information, see the *Distributed Systems Installation Guide*.

Installation Procedures

During the CA Gen installation, a list of available CA Gen components is displayed. The following components should be installed to generate EJBs from the Client/Server Encyclopedia:

Component	Notes
CSE Construction Server	Provides the generators needed to generate the application and remote files.
Encyclopedia Server Cross Generation Option to EJBs	Provides support for generating EJBs.

Application Server Setup

The application server should be completely configured and tested to ensure it is ready to accept EAR files for deployment and execution. Most application servers have some sort of installation verification procedure. This procedure should be successfully completed before generating or deploying an application.

There may be additional setup tasks for your application server. For example, the default port for HTTP may not be port 80. Other application servers may not have hot deployment enabled by default. Review the installation documentation for your application server and consider these and similar settings.

Install the JDBC Driver

The JDBC Driver is a set of classes provided by the DBMS vendor that allow Java applications to access the target DBMS. Most application servers are delivered with a JDBC driver for select DBMSs. To determine if the JDBC Driver required by your application is preinstalled, see the application server documentation.

If necessary, install the JDBC driver for your target DBMS into the application server.

Chapter 3: DDL Generation

CA Gen-generated Java applications do not require a Java-specific DDL Generation. All generated Java applications use JDBC to access their database and have no special requirements on how the DDL was generated or installed.

In general, customers who are retargeting their application to Java should continue to use the Technical Design of their existing application. For example, if the existing application is generated in C and uses Oracle, then the CA Gen-generated Java application should be generated with Oracle as the DBMS. This implies that existing CA Gen-generated databases may be reused by CA Gen-generated Java applications.

The following section describes how to use the JDBC Technical Design and how to generate the DDL defined and specialized within it. If you are reusing an existing technical design, continue to use the DDL generation and installation methods currently in use.

How to Install DDL Using the JDBC Technical Design

The JDBC Technical Design can be used when there is no existing technical design to reuse. The steps required to install DDL using the JDBC Technical Design are outlined next:

Follow these steps:

1. Set up the DBMS.
2. In the CA Gen Build Tool, define the User ID, password, and other information needed to access the database.
3. Perform DDL Generation with Installation.

Set Up the DBMS

Install and configure the DBMS software. Perform all post-installation tests to ensure the DBMS is ready for use.

Create the physical database and tablespaces required to host the tables and indexes.

If necessary, set the security within the DBMS so that the CA Gen user or application may access the database.

Set Up the Build Tool

On your Windows system, start the CA Gen Build Tool to define the following tokens to set up the Build Tool.

Follow these steps:

1. Open the profile in the Profile Manager.
2. Expand DBMS.
3. Select JDBC.
4. Set OPT.DBUSER and OPT.DBPSWD as appropriate to access the target database.
5. The values for the next three tokens vary by JDBC driver. To determine the values to be entered into the Build Tool, see the documentation for your JDBC driver.
   - Set OPT.DBCONNECT to the JDBC connection string for the target database. For example:

     jdbc:oracle:thin:@myhost:1521:database
   - Set LOC.JDBCDRIVERS to the driver manager class name to be loaded by the Build Tool before connecting to the JDBC DBMS for DDL installation. For example:

     oracle.jdbc.driver.OracleDriver
   - Set LOC.JDBCDRIVERSCLASSPATH to the disk, directory, and ZIP or JAR file name where the JDBC Driver is located. For example:

     c:\oracle\jdbc\lib\classes12.zip

Perform DDL Generation

After you set up the DBMS and the Build Tool, you can perform the DDL generation and installation.

Follow these steps:

1. Start the CA Gen Workstation Toolset.
2. Open the model.
3. Open the Technical Design Diagram.
4. Select and open the JDBC Technical Design.
5. If necessary, perform transformation or retransformation to synchronize the logical and physical data models.

6. Open the Generation Diagram.
7. Open the Generation Defaults dialog. Set the generation parameters:
   - Operating System: JVM
   - DBMS: JDBC

   Perform the DDL Generation and Installation.

Chapter 4: Pre-Generation Tasks

This chapter describes pre-generation tasks under the following topic headings:

- CA Gen Java Runtime/User Exits
- Model Considerations
- Build Tool Settings
- Configuring the Database Resource Adapters

CA Gen Java Runtime/User Exits

CA Gen provides a Java Runtime and User Exits to support the generated Java code. There are two ways of deploying the Java Runtime and User Exits:

- Place the CA Gen Java Runtime and User Exits in each EAR file. This allows each application to have its own copy of the runtime and user exits. However, should the runtime need to be PTFed, or if a change is made to the user exits, then each application must have its EAR file updated and redeployed.

  Note: Some application servers do not support this option.

- Place the CA Gen Java Runtime and User Exits in a common location on the application server, such as the ...\lib\ext directory. This simplifies the process of PTFing the runtime or updating a user exit for all applications on an application server. However, this option usually requires a restart of the application server in order for the change to take effect.

In either case, the CA Gen Java Runtime would be PTFed or the User Exits would be updated on a Windows workstation and then moved into the application's EAR file or the application server. Therefore, you should put controls in place to manage the configuration of the CA Gen Java Runtime and User Exits over the life of the application.

Deploy the CA Gen Java Runtimes

If you choose to deploy the CA Gen Java Runtime to the application server rather than in the application's EAR file, the following procedure shows how to perform the deployment.

Follow these steps:

1. Install and configure the application server in accordance with the documentation provided by the vendor.
2. On the Windows system, open a Command Prompt window.

3. Set the environment variable JAVA_HOME to the disk and directory where Java SE is located:

   ```
   set JAVA_HOME C:\Program Files\Java\jre#.#.#_##\
   ```

4. Change to the <CA-Gen-root>\GEN\CLASSES directory:

   ```
   cd "%Gen xx %"
   cd gen\classes
   ```

   Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

5. Execute the batch file MKJAVART with the appropriate case-sensitive parameters exactly as described below.

   Note: It is good practice to put the CA Gen product version number in the JAR file's name.

   - If the application server executes only CA Gen generated EJBs, enter the following command:

     ```
     MKJAVART genrt.xx.jar EJB EJBRMI WS
     ```

   - If the application server executes both CA Gen generated Java Web Clients and CA Gen generated EJBs, enter the following command:

     ```
     MKJAVART genrt.xx.jar WCE EJB EJBRMI WS
     ```

     Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

   - If CA Gen generated EJBs are to participate in flows to and from CA Gen generated non-EJBs servers, additional communication types are needed. Use the USAGE parameter to display the full list of supported parameters and cooperative flow runtimes:

     ```
     MKJAVART USAGE
     ```

6. When the batch file has completed, copy the JAR file to the appropriate directory for the target application server. The directory is different for each application server. Review the documentation for your application server to determine the best location for additional JAR files to be installed.

Model Considerations

This section details the following areas of concern when generating the Server Procedure Steps as EJBs:

- Marking attributes to use Decimal Precision
- Adding Java Package Names to the CA Gen model
- Packaging considerations
- Setting EJB/RMI-specific properties for Server Managers

Decimal Precision

If you are re-targeting your COBOL application to Java, you may want to set some numeric attributes to use Decimal Precision. For more information, see COBOL Application Considerations.

Add Java Package Names to a Model

Package names are used to allow similarly named methods to be properly referenced. For example, CA could produce a class named Address. There is nothing to prevent you from creating a class by the same name. But by declaring a package name, such as com.ca.gen, when the class is defined, the effective class name becomes com.ca.gen.Address.

Although most situations do not require that you set the Java package name for your application(s) in your CA Gen model, you may find it useful to do so. By convention, the package name is the reverse of the customer's internet domain name. For example, www.ca.com would become com.ca.<facility-name>.

Note: If you use CBD techniques, you must set the Java package names in your models.

Package names can be specified at the business system and model levels. If no package name is specified at the business system level, the package name at the model level is used. If no package name is specified at either level, a version of the short model name that conforms to the rules listed below is used.

The following list describes the rules for package names:

- Only ASCII letters, numbers, and periods are allowed.
- Mixed-case text is allowed.
- A number cannot begin any word in the name.
- Words are separated by periods.
- No leading or trailing periods are allowed.

Package Names for CBD Models

To allow components to be easily consumed in the Java environment, the following package name settings are recommended:

- In the implementation model, define a Package Name at the Model Level.
- In the consuming model, import the spec model into its own Business System and set its package name to that of the implementation model.

Set the Model's Package Names

Perform the following steps to set the model's package names.

Follow these steps:

1. With the model open in the Toolset, open the Environment Diagram by using one of the following methods:
   - On the main menu bar, select Construction, Environment.
   - On the main menu bar, select Tool, Construction, Environment.
   - On the tree control, select the Diagram tab, expand Construction, then double-click Environment.
2. On the Main Menu bar, click Options, then Model Generation Properties.
3. Enter the Java package name in the field provided and click OK.

Set the Business System Package Names

Perform the following steps to set package name at the business system level.

Follow these steps:

1. With the model open in the Toolset, open the Environment Diagram by using one of the following methods:
   - On the main menu bar, select Construction, Environment.
   - On the main menu bar, select Tool, Construction, Environment.
   - On the tree control, select the Diagram tab, expand Construction, then double-click Environment.
2. Select the target business system.
3. On the main menu bar, select Detail, Properties.
4. Enter the Java package name in the field provided and click OK.

   Note: If the field is disabled, change the Operating System to JVM.

Cooperative Packaging

When generating for Java, the organization of load modules and procedure steps is not as important as for other target environments. For Java generation, a CA Gen Server Manager becomes an EJB-JAR file (a collection of EJBs) rather than an executable load module where total size is a consideration. Therefore, it may be advantageous to place all the Server Procedure Steps into a single Server Manager. This simplifies many processes, including deployment.

Server Manager Properties

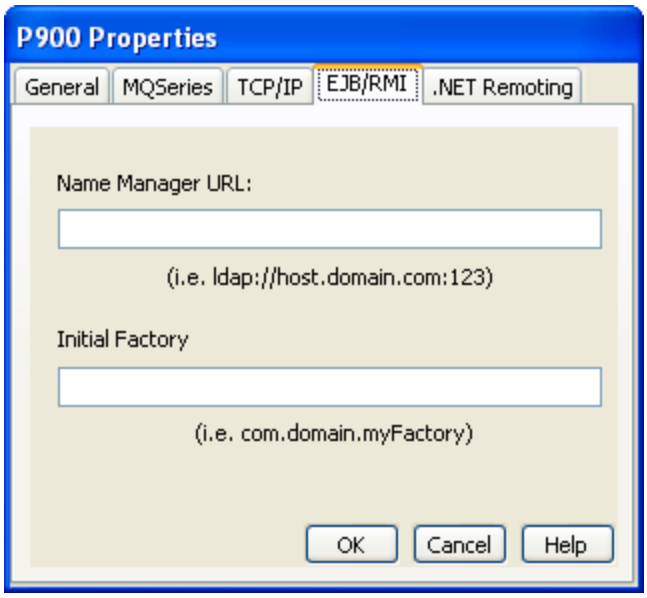
Setting the Server Manager EJB/RMI properties is required only when the application is going to be deployed in more than one EAR file. Some examples of this situation include:

- Deploying the clients in one EAR file and the servers in another EAR file
- When the servers are being deployed in more than one EAR file

If the application is being deployed in exactly one EAR file, these properties need not be set.

The Server Manager properties dialog allows you to specify EJB/RMI-specific information. Supplying this information simplifies the deployment. However, the values that are entered in the fields tend to be application-server dependent. Therefore, the EAR file created by CA Gen will contain application server-specific information.

To open the Server Manager properties dialog, open either the Cooperative Packaging or the Cooperative Generation dialog. Then either double-click the desired Server Manager, or select the Server Manager and select Properties from the Detail menu. The Server Manager properties dialog appears.



The values entered in the properties dialog are application-server dependent. Review the documentation for your application server to determine the values for the following fields:

- Name Manager URL—Enter the URL of the JNDI server the EJB will be defined within. For example:

  `localhost:1099`

  where localhost can be in IPv4 or IPv6 format.

- Initial Factory Class—Enter the name of the factory class the EJB Reference will invoke. For example:

  `org.jnp.interfaces.NamingContextFactory`

Define Build Tool Tokens

You must define the Build Tool tokens required to build and assemble CA Gen EJB Servers.

Follow these steps:

1. Start the CA Gen Build Tool on your Windows system.
2. Open the profile in the Profile Manager.
3. Select JAVA.
4. Set LOC.JDK_HOME to the disk and root directory where the Java SE is located.
5. Set LOC.JAVAEE_HOME to the disk and root directory where the Java EE is located.
6. Expand DBMS and select JDBC.
7. Set OPT.DBUSER and OPT.DBPSWD as appropriate to access the target database.

   The values for the next three tokens vary by JDBC driver. To determine the values to be entered into the Build Tool, see the documentation for your JDBC driver.
8. Set OPT.DBCONNECT to the JDBC connection string for the target database. For example:

   `jdbc:oracle:thin:@myhost:1521:database`
9. Set LOC.JDBCDRIVERS to the driver manager class names to be loaded by the Build Tool before connecting to the JDBC DBMS. For example:

   `oracle.jdbc.driver.OracleDriver`
10. Set LOC.JDBCDRIVERSCLASSPATH to the disk, directory, and ZIP or JAR file name where the JDBC Driver is located. For example:

    `c:\oracle\jdbc\lib\classes12.zip`

Configure the Database Resource Adapters

Resource Adapters are part of the Java EE Connector Architecture. They allow application servers to interact more easily with enterprise information system resources such as a DBMS.

Configuring a resource adapter varies by DBMS and application server. Review the documentation for your application server to determine the appropriate procedures for defining resource adapters.

Chapter 5: Construction and Deployment

This chapter describes the construction and deployment of CA Gen Enterprise JavaBeans application.

Prerequisites

The following tasks should be performed before generating the application in Java:

- The resource adapter for the application's database should be defined in the application server.
- The CA Gen Java Runtime and User Exits should be copied to a common area of the application server if that deployment option is been chosen.

Generation

This section describes the actions required to generate CA Gen Server Procedure Steps as EJBs. The generation can take place on the Windows Workstation Toolset or the Client Server Encyclopedia. The construction of the generated code can only be performed with the Build Tool running on Windows. Therefore, the remote file created by the CSE must be copied to a Windows system with the Build Tool to construct the application.

Target Environment Definition

When generating the CA Gen Server Procedure Steps as EJBs, set the following generation parameters in the Server Environment Parameters dialog:

Parameter	Value
Operating System	JVM
DBMS	JDBC or any supported DBMS (Technical Design)
Language	Java
TP Monitor	EJB or EJB Web Services
Communications	JavaRMI

Set the DBMS Generation Parameter

When setting the DBMS Generation parameter, JDBC or any valid DBMS that supports JDBC can be chosen. Regardless of what is selected, all database access is generated as JDBC calls.

The only advantage to choosing a specific DBMS over JDBC is that choosing a specific DBMS uses the table names, index names, and other user-modified names that have been defined under the Technical Design dialogs. Therefore, it may be better to choose a specific DBMS when the application will access an existing database. Applications accessing a new database may want to choose JDBC.

Note: ODBC is not a valid value for the DBMS generation parameter.

Set the TP Monitor

Selecting the TP Monitor parameter as an EJB Web Service, results in generation of the CA Gen Server Procedure Steps as not only Stateless Session Enterprise JavaBeans but also as Web Services. Selecting the TP Monitor parameter as an EJB, results in generation of the CA Gen Server Procedure Steps as Stateless Session Enterprise JavaBeans only.

Construction

Building EJBs is no different from building any other generated application. However, the construction of the generated application can only be performed with the Build Tool running on a Windows system.

If the application was generated on the Windows workstation, the Build Tool will be invoked to build all generated cascade and load modules.

If the application was generated by a CSE, copy the remote file(s) to a Windows system with the Build Tool. Then start the Build Tool, select the directory where the remote files are located, select all cascade and load modules to be built, and then click Actions, Build.

The results of the construction process are placed in the <model-directory>\java\classes directory. The following files are created by the build of a Load Module and are found in the classes directory:

- <load-module>.jar—Contains the classes that implement the Server Procedure Step as a Stateless Session bean. The EJB Server for execution loads this file.
- <Server Procedure Step>_dpy.jar—Contains the portable interface definitions for the EJB. This file is used by the clients (Java Web Client, Java Proxy, and so on) to call the EJB.

Assembly

After all code has been successfully built, the application is ready for assembly and deployment. The assembly process can only be performed by the Build Tool running on a Windows system. After the EAR file has been created, it is ready for deployment.

Select Load Modules for Assembly

After all cascade and load modules have been successfully built, they are ready for assembly into an EAR file.

Follow these steps:

1. In the CA Gen Build Tool on the Windows system, select the Cascade, Java Web Clients, EJB load modules, and OPLIBs to be assembled into an EAR file.

   Note:

   - Do not select the application's Database (that is, DDL).
   - If as a result of work done with the PStep Interface Designer in CA Gen Studio you generated a Web Service Definition module, make sure you include it with the rest of the application modules during assembly. Web Service Definition module names are of the form <*WSD-name*>.WSD.icm, where <*WSD-name*> is the web service definition name. For information about the PStep Interface Designer and CA Gen Studio, see the *Gen Studio Overview Guide*.

2. After all the desired load modules have been selected, click the Assemble button or select Actions and Assemble from the main menu. The EAR File Assemble Details dialog box opens.

   The EAR File Assemble Details dialog has a selection tab on the left and a panel on the right.

   Note: For a description of all fields in this dialog box, see the *Build Tool User Guide*.

   After the wizard has completed, click OK to build the application's EAR file.

Review Assemble Status

After the CA Gen Build Tool has completed processing, the log of the assemble process should be reviewed. In the Build Tool, select the assemble entry on the modules panel and then choose Actions, Review from the main menu or select the Review icon.

The log file is located in the following directory:

```
<model-directory>\java\assemble.EAR.out
```

Deployment

After the EAR file has been created, it is ready for deployment onto the application server. The deployment process is unique for each application server. To determine how to deploy the EAR, see the documentation for your application server.

Chapter 6: Running Diagram Trace in the EJB Environment

This chapter explains how to run the application to be traced using the Diagram Trace Utility in the EJB environment.

How to Run Diagram Trace in the EJB Environment

Running Diagram Trace in the EJB environment requires the generated Server Procedure Step to connect to the Diagram Trace Utility executing on a Windows system.

Follow these steps:

1. Generate the Procedure Steps and Action Blocks with Trace.
2. Assemble the application and specify the Diagram Trace Utility host and port number.
3. Start the Diagram Trace Utility.
4. Run the application to be traced.

EJB Generation for Trace

There are two ways to generate the EJB for Trace:

- Specify Generate source code with Trace in the Generation Options dialog.
- Check the TRCE flag for all code to be traced.

Assemble the Application

EJBs do not have a user interface. Therefore, it can only be traced with the Diagram Trace Utility. This program executes on a Windows system. The EJB connects to the Diagram Trace Utility using TCP/IP.

During assembly, the EAR File Assemble Details dialog appears.

Follow these steps:

1. On the EAR File Assemble Details dialog, select Tracing from the tree view on the left.
2. In the details area on the right, select the Enable Diagram Trace check box.

3. Specify the Trace Server Name and Trace Server port number of the Diagram Trace Utility. For example:

   localhost

   4567

   where localhost can be in IPv4 or IPv6 format.

Start the Diagram Trace Utility

You may start the Diagram Trace Utility by using the Start menu. Click Start, All Programs, CA, Gen <version>, Diagram Trace Utility.

The Diagram Trace Utility will "autostart" and listen on port 4567. You may change the default port from within the Diagram Trace Utility.

Note: For more information about the Diagram Trace Utility, see the *Diagram Trace Utility User Guide*.

Run the Application with Trace

After the Diagram Trace Utility is running and ready to accept connections, the EJB can be started. Access the EJB as it would normally be accessed. The connection to the Diagram Trace Utility takes place shortly after the EJB has been initialized.

Note: For more information about running and debugging a Java application with the Diagram Trace Utility, see the *Diagram Trace Utility User Guide*.

Chapter 7: Converter Services

This chapter details the cooperative flows between generated clients and servers implemented in different languages. These include:

- CA Gen MFC GUI Clients as well as CA Gen C and COM Proxy Clients to CA Gen EJB Servers using either of the following:
  - **The CFB Server**—The CFB Server is a Java program that can run on any Java Platform in the network. Client systems connect to the CFB Server, which converts the client request into a Java RMI call.

    The CFB Server requires only a small change to the COMMCFG.INI file on the client system. For more details, see Configuring the CFB Server.
  - **The C to Java RMI Coopflow**—The C to Java RMI Coopflow is a software component that performs the conversion and Java RMI call on the Windows system. This facility is easier to configure, but may be difficult to manage for a large number of client systems.

    The client system must have a Java Virtual Machine installed, plus additional files from the target application server and target EJBs.
- CA Gen EJB Servers to CA Gen C and COBOL Servers using any of the supported cooperative flows.

More information:

How to Configure C to the Java RMI Coopflow (see page 53)

How to Configure the CFB Server

The following tasks need to be performed to use the CFB Server:

1. Ensure that the prerequisites are met.
2. Unjar the CFB Server.
3. Edit CFBServer.properties.
4. Edit jndi.properties.
5. Create a consolidated CA Gen Java Runtime JAR file.
6. Copy components from the application server.
7. Copy files from the EJB build machine.

8. Create a startup batch/script file.
9. Change COMMCFG.INI on the client systems.

These tasks are detailed in the following sections.

Prerequisites

To successfully execute the CFB Server, the following prerequisites must be satisfied:

1. Ensure that the system running the CFB Server has sufficient computing power and network capacity to handle the anticipated traffic.
2. Ensure that the CA Gen Converter Services software has been installed.
3. Ensure that the system has a valid JVM installed. Use the following procedure to test the validity of the JVM:
   a. Open a Command Prompt window.
   b. Enter the following command:

      ```
      java -version
      ```

   c. Examine the output. If text similar to the following is displayed, the Java (the JRE, JDK, or Java SE) is not in the environment variable PATH.

      ```
      The name specified is not recognized as an internal or external command,
      operable program or batch file.
      ```

      To correct this issue, add the disk and directory name of the JRE or Java SE's bin directory to the PATH variable:

      If text similar to the following is displayed, a version of Java has been installed and is ready to use:

      ```
      java version "#.#.#_##"
      Java(TM) SE Runtime Environment
      ```

      Note: For the supported versions of software, see the *Technical Requirements* document.

      If the correct version is not displayed, check the environment variable PATH. In most cases, an older version of Java (the JRE, JDK, or Java SE kit) appears before the desired version.

Un-JAR the CFB Server

The CFB Server is installed as a single JAR file which must be unpacked into its own directory, as detailed next:

Follow these steps:

1. Create a top-level directory named CFBServer.
2. Locate the jcfb##.jar file in the <CA-Gen-root>\classes directory.
3. Unjar the contents of the file into the directory. For example:

   ```
   %JAVA_HOME%/bin/jar xvf <CA-Gen-root>\classes\jcfb##.jar
   ```

4. Customize the properties and policies files as described in the following sections.

Edit CFBServer.properties File

The CFBServer.properties file specifies the runtime characteristics of the CFB Server. The directory containing the CFBServer.properties file must be placed in the class path before the directory containing the CFB Server class files. After the file has been edited, the CFB Server must be restarted for the changes to take effect.

The file contains several properties in the form of name=value. The available properties are listed next:

- **listen.ports**—The port numbers that the CFB Server will monitor for client connections. This is a space-separated list of port numbers between 2000 and 65534 (inclusive). The list is limited to 16 port numbers. For example:

  ```
  listen.ports=8901 8902
  ```

- **control.timer**—The number of milliseconds between scans for timed-out connections. After a timed-out connection is found, the port is closed. For example:

  ```
  control.timer=5000
  ```

  In this example, a scan will be conducted every five seconds.

- **control.port**—The port number that the CFB Server will monitor for control connections. This should be set to a single port number between 2000 and 65534 (inclusive). For example:

  ```
  control.port=8900
  ```

- **trace.mask**—A bit mask to enable debug tracing of the CFB Server. The value of zero should be used at most times. Enter a value of –1 to trace all events. For example:

  ```
  trace.mask=128
  ```

- **trace.file**—Specifies the output trace file name and location. Note that forward slashes (/) are used in the path name. For example:

  ```
  trace.file=../log/CFBServer.log
  ```

- **control.allowed.inet**—To increase security, use this entry to specify a space-separated list of the IP address that are allowed to connect to the CFB Server's command port. Note that host names are not allowed. The localhost IP address of ::127.0.0.1 can be used. For example:

  ```
  control.allowed.inet = ::128.247.234.39 ::128.247.234.42
  ```

  The IP address can also be in IPv6 format. For example:

  ```
  control.allowed.inet = ::fed0:0000:0000:0002:0218:b9ff:fe45:0a0a ::fcc0:0000:
  0003:0003:0319:f9bf:fe45:0a0a
  ```

Edit jndi.properties File

The jndi.properties file specifies the JNDI connection information the CFB Server uses to find the target EJBs. The directory containing the file must be placed in the class path before the directory containing the class files. This file can be edited as needed. After the file has been edited, the CFB Server must be restarted for the changes to take effect.

The JNDI connection information varies for each application server. Review the documentation for your application server to find the settings for the following fields.

The file contains several properties in the form of name=value. The available properties are listed next:

- **java.naming.factory.initial**—Specifies the initial context factory of the target JNDI server. For example:

  ```
  java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
  ```

- **java.naming.provider.url**—Specifies the name manager URL of the target JNDI server. For example:

  ```
  java.naming.provider.url=localhost:1099
  ```

  where localhost can be in IPv4 or IPv6 format.

Create a Consolidated CA Gen Java Runtime JAR File

The CFB Server uses the CA Gen Java Runtime to interact with Server Procedure Steps generated as EJBs. The CA Gen Java Runtime is made up of a number of CLASS and JAR files.

A batch file—MKJAVART.BAT—has been provided to consolidate the runtime into a single file. For more information about the MKJAVART.BAT file, see the Deploying the CA Gen Java Runtimes in the chapter "Pre-Generation Tasks". Note that parameters to the MKJAVART.BAT file are case-sensitive, as follows:

```
cd <CA-Gen-root>\gen\classes
mkjavart genrt.xx.jar EJB EJBRMI
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

It is a good practice to put the CA Gen product version number in the name of the JAR file. If necessary, copy the consolidated CA Gen Java Runtime JAR file to the system running the CFB Server.

Copy Components from the Application Server

For the CFB Server to access EJBs on the application server, a number of the application server classes must be copied to the client system.

Follow these steps:

1. On the system running the CFB Sever, create a directory with the name of the target application server. For example:

   ```
   mkdir c:\weblogic
   ```

2. Copy the required components from the application server directory structure to the newly created directory. These components vary for each application server. Review the documentation for your application server to determine the JAR(s) that make up the client components.

Copy Files from the EJB Build Machine

For the CFB Server to access EJBs on the application server, the portable interface definition files of the EJB must be accessed by the client. On the system running the CFB Server, create a directory to house the portable interface definition files. For example:

```
mkdir c:\app1
```

During the construction of the generated EJBs, a file (one per Server Procedure Step) containing the EJB portable interface definition is created. The file is named <Server Procedure Step>_dpy.jar.

Copy all <Server Procedure Step>_dpy.jar files from the <model-root>\java\classes directory to the directory on the CFB Server system.

Create a Startup Batch/Script File

To easily start the CFB Server, a batch or script file should be created.

Follow these steps:

1. Change the current directory to the CFB Server root directory.
2. Define the private class path in an environment variable.
3. Invoke the CFB Server using the private class path.

The following sections describe how to create the batch file. Although the examples used are specific to Microsoft Windows, these procedures apply to all systems where the CFB Server can execute.

Create the File

The first step is to create a batch or script file.

The first command within the file should change the current directory to the root directory of the CFB Server.

```
cd c:\CFBServer
```

Define the Class Path

To facilitate the setup of the CFB Server, set a private version of the class path, so that the application server JAR file(s), the CA Gen runtime JAR file, and the <Server Procedure Step>_dpy.jar file(s) can access the generated EJBs.

The class path is a semicolon-separated list of directories containing CLASS files and/or a semicolon-separated list of JAR files. For example:

```
CP=.;c:\MyApp\classes;c:\OtherApp\sample.jar
```

Follow these steps:

1. Set the class path to the current directory:

   ```
   SET CP=.
   ```

2. Append the list of JAR files copied from the target application server to the client system:

   ```
   SET CP=%CP%;<app-server-dir>\<file1>.jar; <app-server-dir>\<file2>.jar
   ```

3. Append the consolidated CA Gen Java Runtime JAR file to the CLASSPATH:

   ```
   SET CP=%CP%;<directory>\genrt.xx.jar
   ```

   Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

4. Append the list of <Server Procedure Step>_dpy.jar files copied from the EJB build machine to the client system:

   ```
   SET CP=%CP%;<model-dir>\c\<Server Procedure
   Step-1>_dpy.jar;<model-dir>\c\<Server Procedure Step-2>_dpy.jar
   ```

   Remember to separate each directory or file with a semicolon.

Invoke the CFB Server

With the private class path completed, add a command to invoke the CFB Server software:

```
java -cp "%CP%"
-Djava.security.policy=CFBServer.policy
com.ca.genxx.jcfbserv.server.CFBServer
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

This is the final command for the CFB Server startup batch/script file. Close the file.

The CFB Server should then be started. A successful invocation of the CFB Server will produce the following messages:

```
CFBServer Initialized.
CFBServer Started.
```

Change COMMCFG.INI on the Client Systems

The COMMCFG.INI file on the client system must be changed to allow the clients to interact with Server Procedure Steps generated as EJBs via the CFB Server.

The COMMCFG.INI file is located in the <CA-Gen-root> directory. The general format of the COMMCFG.INI entry is as follows:

```
<trancode> TCP <host> <service/port>
```

For example:

```
TCP myserver 8901
```

When this entry is placed at the top of COMMCFG.INI, all transactions use the TCP/IP coopflow to connect to the CFB Server on myserver using port 8901. The <host> can be in IPv4 or IPv6 format.

Invoke the Client

After all the preceding tasks have been completed, execute the client as before.

CFB Server Control Client

The CFB Server Control Client allows the system administrator to interactively see what is happening in any CFB Server. The Control Client is an interactive task that connects to any CFB Server through a published (or well known) socket to display connection status and control active connections.

The default configuration of the CFB Server accepts connections only from the local host IP address. The CFB Server properties can be changed to accept connections from other IP addresses, allowing the Control Client to connect to a Server at a remote IP address.

The Control Client uses a command line interface. The localhost can be in IPv4 or IPv6 format. To start the Control Client, open a command window and enter the following command:

```
java -cp <CA-Gen-Root>\classes\jcfb##.jar
    com.ca.gen xx .jcfbserv.control.ControlServer
    localhost:8900
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

The controlCommand.jar file must be in the current directory.

Localhost:8900 is the control port on the CFB Server in this example.

The basic format of the command is:

```
Dis[play] [port]
```

The Control Client cannot modify the state of any resource in the CFB Server.

The following commands are available for the Control Client.

- Help—lists all available commands.
- Quit—terminates the Control Client.

- Display—shows two or more lines for each listening port. The first line is the name of the server, the port number, and number of active connections.
  - Control—the server that handles the display command request.
  - Listener—the server that handles TCP/IP CFB requests from clients.
  - Request Processor—the server that connects to the EJB server for each request.

  The second line displays server statistics of connections (successful and failed). One additional line is displayed for each active session. It contains:
  - Connection ID—listening port requesting port pair.
  - Status—COK for active connections.
  - Bytes sent and received for connections on listening ports.
  - Time connection has been alive.
  - The name of the connection.

  Request Processor EJB connections do not show performance statistics.
- Display port—shows the status of the identified resource where the server is listening. Examples of what is displayed follow:
  - Command Reader OK.
  - Control[8900] serving 1 active connections.
  - Since host start 1 connections succeeded. 0 failed.
  - 8900:1360 COK 0 0 0 0 0sec XC[8900:1360]
  - Listener[8901] serving 0 active CFB connections.
  - 0 connections succeeded. 0 connections failed.
  - Listener[8902] serving 0 active CFB connections.
  - 0 connections succeeded. 0 connections failed.
  - Request Processor Thread-2 active 10sec. idle 300sec.
  - Processed 0 request, of which 0 failed, processed 0 responses and 0 errors.

How to Configure C to the Java RMI Coopflow

The following tasks need to be performed to use the C to Java RMI Coopflow:

1. Ensure that the prerequisites are met.
2. Create a consolidated CA Gen Java Runtime JAR file.
3. Copy components of the application server to the client system.

4. Copy the target portable interface definition files from the EJB build machine to the client system.
5. Set the CLASSPATH for the client.
6. Change the COMMCFG.INI file to point to the target application server.

These tasks are detailed in the following sections.

Prerequisites

To successfully execute the C to Java RMI Coopflow, the following prerequisites must be satisfied:

1. Ensure that the CA Gen Converter Services software has been installed.
2. Ensure that the client system has a valid JVM installed. Use the following procedure to test the validity of the JVM:
   a. Open a Command Prompt window.
   b. Enter the following command:

      ```
      java -version
      ```

   c. Examine the output. If text similar to the following is displayed, Java (the JRE, JDK, or Java SE kit) is not in the environment variable PATH:

      ```
      The name specified is not recognized as an internal or external command,
      operable program or batch file.
      ```

      To correct this issue, add the disk and directory name to the PATH variable.

      If text similar to the following is displayed, a version of Java has been installed and is ready to use:

      ```
      java version "#.#.#_##"
      Java(TM) SE Runtime Environment
      ```

      Note: For the supported versions of software, see the *Technical Requirements* document.

      If the correct version is not displayed, check the environment variable PATH. In most cases, an older version of Java (the JRE, JDK, or Java SE kit) appears before the desired version.

Create a Consolidated CA Gen Java Runtime JAR File

The CA Gen Java Runtime is used by the client application to interact with Server Procedure Steps generated as EJBs. It is made up of a number of CLASS and JAR files.

A batch file—MKJAVART.BAT—has been provided to consolidate the runtime into a single file. For more information about the MKJAVART.BAT file, see the Deploying the CA Gen Java Runtimes in the chapter "Pre-Generation Tasks". Note that parameters to the MKJAVART.BAT file are case-sensitive, as follows:

```
cd <CA-Gen-root>\gen\classes
mkjavart genrt.xx.jar EJB EJBRMI
```

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

It is a good practice to put the CA Gen product version in the JAR file name.

Copy Components from the Application Server

In order for the client system to access EJBs on the application server, a number of the application server classes must be copied to the client system.

Follow these steps:

1. On the client system, create a new directory with the name of the target application server. For example:

   ```
   mkdir c:\weblogic
   ```

2. Copy the required components from the application server directory structure to the newly created directory. These components vary for each application server. Review the documentation for your application server to determine the JAR(s) that make up the client components.

Copy Files from the EJB Build Machine

For the client system to access EJBs on the application server, the portable interface definition files must be accessed by the client.

During the construction of the generated EJBs, a file (one per load module) containing the EJB's portable interface definition is created. The file is named <Server Procedure Step>_dpy.jar.

Copy all <Server Procedure Step>_dpy.jar files from the <model-root>\java\classes directory to a directory on the client system, such as the <model-root>\c directory.

Set the CLASSPATH

For the client to successfully access the generated EJBs, the CLASSPATH must be set so that the application server JAR files, the CA Gen runtime JAR file, and the <Server Procedure Step>_dpy.jar files are in the CLASSPATH.

CLASSPATH is a semicolon-separated list of directories containing class files and/or a semicolon-separated list of JAR files. For example:

```
CLASSPATH=c:\MyApp\classes;c:\OtherApp\sample.jar
```

The CLASSPATH can be set in a variety of ways. In some cases, creating a small batch file may be a good choice. In other environments, manually setting the system environment variable is the only option.

Either way, the following tasks must be performed:

Follow these steps:

1. Set the CLASSPATH to the list of JAR files copied from the target application server to the client system:

   ```
   <app-server-dir>\<file1>.jar; <app-server-dir>\<file2>.jar
   ```

2. Append the consolidated CA Gen Java Runtime JAR file to the CLASSPATH:

   ```
   <CA-Gen-root>\gen\classes\genrt.xx.jar
   ```

   Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

3. Append the CLASSPATH with the list of <Server Procedure Step>_dpy.jar files copied from the EJB build machine to the client system:

   ```
   <model-dir>\c\<Server Procedure Step-1>_dpy.jar;<model-dir>\c\<Server Procedure Step-2>_dpy.jar
   ```

   Remember to separate each directory or file with a semicolon.

Change COMMCFG.INI File

In general, the COMMCFG.INI file must be changed to let the clients interact with Server Procedure Steps generated as EJBs. In doing so, the client uses a different set of cooperative flows to interact with the application server.

The only exception is when the client has been generated after the Server Manager Java RMI properties have been set. In this case, the generated files already target the desired application server.

The COMMCFG.INI file is located in the <CA-Gen-root> directory. The general format of the COMMCFG.INI entry is as follows:

```
<trancode> EJBRMI <initial-context-factory> <name-manager-url>
```

For example:

```
EJBRMI org.jnp.interfaces.NamingContextFactory myhost:1099
```

When this entry is placed at the top of COMMCFG.INI, all transactions use the C to Java RMI coopflow to connect to myhost using port 1099 to invoke the Initial Context Factory org.tnp.interfaces.NamingContextFactory to perform the EJB lookup.

The Initial Context Factory and Name Manager URL vary for each application server. Review the documentation for your application server to determine the values of these parameters.

Invoke the Client

After all the above tasks have been completed, execute the client as before.

CA Gen Cooperative Flow to CA Gen C and COBOL Servers

The CA Gen EJB Servers may interoperate with CA Gen C and COBOL Servers using the any of the supported cooperative flows.

Configuration

There are a few steps that must be performed to have the CA Gen EJB Servers to interoperate with the CA Gen-generated C and COBOL servers, as described in the following sections.

Configure the Cooperative Flow

Configuring the cooperative flow can be accomplished by either one of the two following methods:

- Change or define the server's properties and regenerate, ensure the server environment parameters are set to the C or COBOL environment with the correct communications setting and regenerate the CA Gen EJB Servers.

  OR

- Modify the COMMCFG.PROPERTIES file to specify the new target CA Gen C or COBOL Server.

Each of these options is described in detail in the following sections.

Option 1: Change Your Model and Generate

During generation, the generated servers know the servers they will be utilizing at runtime and the default cooperative flow to use for that server. The cooperative flow information comes from the server properties dialog and server environment parameters dialog.

Follow these steps:

1. Open your model. On the main menu bar, select Construction, Packaging or Construction, Generation.
2. Open the cooperative packaging diagram by selecting Diagram, Open, Cooperative Code.
3. For each server that is generated for either C or COBOL:
   a. Open the server manager properties dialog by either double clicking on the server, select the server and then selecting Details, Properties from the main menu bar or right-clicking the server and selecting Details, Properties from the pop-up menu.
   b. In the server manger dialog, select the tab for the cooperative flow you will be using and enter the appropriate information.
   c. Click OK to save your changes.
   d. With the server manger still selected, open the server environment parameters dialog by either selecting Detail, Server Environment from the main menu bar or right-clicking on the server and selecting Detail, Server Environment from the pop-up menu.
   e. Set the server's Operating System and Communications to the desired values.
   f. Click OK to save your changes.
4. After all the servers have been configured, generate and build the CA Gen EJB Server. It will now contain data that will default to interoperating with the CA Gen C or COBOL Servers.
5. Before assembling and deploying the application, ensure the COMMCFG.PROPERTIES file does not contain an entry that will override the information generated in the server. For more information, see the next section.

Option 2: Modify COMMCFG.PROPERTIES

CA Gen EJB Servers can be redirected to CA C and COBOL Servers by override the cooperative flow information generated in the server. The COMMCFG.PROPERTIES file can be read at run time to direct the EJB server to send the cooperative flow to another system. Note that the COMMCFG.PROPERTIES file will only use accessible o the EJB servers when it is included in the EAR file. The COMMCFG.PROPERTIES is placed in the EAR file only when the option Include optional runtime property files with runtime is selected.

Follow these steps:

1. Locate the COMMCFG.PROPERTIES file under the CA Gen directory.
2. In the editor of your choice, open the file COMMCFG.PROPERTIES

   `write commcfg.properties`
3. Create an entry for the trancode(s) to be redirected to the CA Gen-generated C or COBOL servers:

   `<trancode> = TCP <host> <service/port>`

   where <host> can be in IPv4 or IPv6 format.
4. Reassemble and redeploy the application. Ensure that the option Include optional runtime property files with runtime is selected.

Execute the Application

After all the preceding tasks have been completed, execute the application as before.

Chapter 8: User Exits

User exits let you to change the behavior of the CA Gen runtimes. User exits are Java classes that are shipped with the product and contain specific routines that are called by the runtimes. You can modify the source code to change the default behaviors of the runtimes.

All Java user exits are located under the %GEN*xx*%Gen\classes directory.

Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Java requires that the files and directory structure mirror the package name structure defined in the Java source code. The base package name is:

`com.ca.gen`*xx*`.exits`

The base package name is then appended with a sub-package name, which specifies the component within the runtime the module is a part of. For example, the file CompareExit.java located in the %GEN*xx*%Gen\classes\com\ca\gen*xx*\exits\common directory has the package name com.ca.gen*xx*.exits.common.

Note: For more information about user exits, see the *User Exit Reference Guide*.

User Exit Locations

The following exits are located in the %GEN*xx*%Gen\classes\com\ca\gen*xx*\exits\common directory:

- CompareExit.java
- DataConversionExit.java
- LowerCaseExit.java
- UpperCaseExit.java

Note:

- *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.
- For more information about the purpose and use of DataConversionExit.java method, see the *User Exit Reference Guide*.

The following exit is located in the %GENxx%Gen\classes\com\ca\gen*xx*\exits\coopflow\ejbrmi directory:

- EJBRMIContextExit.java
- EJBRMIDynamicCoopFlowExit.java
- EJBRMISecurityExit.java

The following exit is located in the %GENxx%Gen\classes\com\ca\gen*xx*\exits\coopflow\tcpip directory:

- TCPIPDynamicCoopFlowExit.java

The following exit is located in the %GENxx%Gen\classes\com\ca\genxx\exits\coopflow\ws directory:

- WSDynamicCoopFlowExit.java

The following exits are located in the %GENxx%Gen\classes\com\ca\gen*xx*\exits\fmrt directory:

- WindowManagerCfgExit.java

The following exit is located in the \classes\com\ca\gen*xx*\exits\jndi directory:

- ContextLookupExit.java

The following exits are located in the %GENxx%Gen\classes\com\ca\gen*xx*\exits\msgobj\cfb directory:

- CFBDynamicMessageDecryptionExit.java
- CFBDynamicMessageEncodingExit.java
- CFBDynamicMessageEncryptionExit.java
- CFBDynamicMessageSecurityExit.java

The following exits are located in the %GENxx%Gen\classes\com\ca\gen*xx*\exits\smrt directory:

- DefaultYearExit.java
- LocaleExit.java
- RetryLimitExit.java
- SessionIdExit.java
- SrvrErrorExit.java
- UserExit.java

After the desired user exit source file has been located, make a backup copy of the file, then edit the file and save it back to its original name and location.

Note: For more information about the purpose and use of each method, see the comments in the exit source or see the *User Exit Reference Guide*.

Important! Do not modify the package name. Doing so will cause the exit to be unrecognized at runtime.

Since the user exit source files are saved to their original locations, it may be useful to save the original and modified sources in a configuration management system to prevent the changes from being lost should a new installation overwrite the source file.

Redistribute the Exits

After the exit has been compiled, it is ready for redistribution. The procedure for redistribution depends on the type of application and the manner the original exits were distributed.

- For Java Web Clients or EJBs, the action depends on whether the EAR file contains a copy of the CA Gen runtime. If the EAR file was deployed with a copy of the CA Gen runtime, then the application should be re-deployed. This causes all updated exits to be pulled into the runtime JAR file and inserted into the EAR file.
- If the runtime is being distributed separately from the EAR files, or if it is being used in a Java Proxy, then the original distribution procedures must be repeated. This may involve rerunning the mkjavart.bat file to rebuild a runtime JAR file.

Chapter 9: External Action Blocks in Java

External Action Blocks (EABs) let your CA Gen application access logic that is not modeled in or generated by CA Gen.

An external action block stub is a code skeleton created when you generate an EAB using a Construction toolset. They are designed to be generated once and then modified and maintained outside of CA Gen. You are responsible for writing and compiling the external subroutine and making it available to the application.

The stub contains the definitions necessary for the skeleton to be called by other generated action blocks and comments that indicate where user code may be added. You may either add a call to other classes and methods or you may add the logic directly into the EAB stub.

This chapter describes the tasks related to External Action Blocks and CA Gen Java-generated applications.

EABs in the Java Environment

All modified and completed EABs must be placed into a separate JAR file and specified to the CA Gen Build Tool for inclusion into the application's EAR file.

If the EABs are not resolved, the application fails at runtime with a 'class not found' exception.

Java requires all references to external classes be resolved at compile time. Consequently, the CA Gen Build Tool will compile all EAB stubs and place them in the JAR files. During application assembly, these classes are removed from the JAR files. This is required since many application servers will search the JAR containing the calling action block before looking to an external JAR file where the completed EAB class file is actually located.

The replacement of completed EAB (for the EAB skeletons) in the original generation directory is not supported due to the likelihood of subsequent generations overwriting the completed EABs.

User Action Outside of CA Gen

Copy the generated EAB stub and associated view files (<class>_IA, <class>_OA) to a separate location before modifying the stub. This prevents the stub, and your code, from being overwritten when the load module is regenerated. The generated stub is located in the java\<package-directory> under the model directory. For example, if DEMO_EAB has a package name of com.mycompany.accounting, the Java file DEMOEAB.JAVA would be located in the <model-directory>\java\com\mycompany\accounting. The file must be copied to a similar directory structure reflecting the package name. For example, c:\my eabs\com\mycompany\accounting.

Add the logic needed to meet the specific requirements of your application. You may use the modified stub to call an existing external method or you may include the method's logic in the stub.

Access Individual Class Members

The import and export views are placed into separate classes in their own source files. For example, the action block DEMO_EAB will be generated as the class DEMOEAB; its import and export views are generated as the classes DEMOEAB_IA and DEMOEAB_OA, respectively.

Within each class are the individual class members that hold the view data. Each view, entity and attribute name is concatenated together and generated as a single flattened variable without underscores for spacing. The single variable may contain non-ASCII characters since Java allows them in variable names. The variable follows the uppercase and lowercase rules typically used in Java programming. Specifically, the first letter of each word is in uppercase and all remaining characters are in lowercase, the first character in the variable name is always in lowercase.

For example:

Entity View: IMPORT_CUSTOMER
Entity: ADDRESS
Attribute: POSTAL_CODE

becomes

importCustomerAddressPostalCode

Each member of the class may be defined as one of the following native Java data types:

char
short
int
double

Further, some members of the class are defined as one of the following native Java classes:

BigDecimal
String

Within a view, each attribute may be accessed as a standard Java property on an object. For example:

```
int id = wIa.importCustomerAddressId;
wOa.exportCustomerAddressId = id;
```

Create a CA Gen Runtime JAR File

The Java EAB makes references to various CA Gen runtime classes. In order for the Java compilation to succeed, a JAR file containing the CA Gen classes must be referenced.

Follow these steps:

1. On the Windows system, open a Command Prompt window.
2. Set the environment variable JAVA_HOME to the disk and directory where Java SE is located:

   ```
   set JAVA_HOME=C:\Program Files\Java\jre#.#.#_##\
   ```

3. Change to the <CA-Gen-root>\GEN\CLASSES directory:

   ```
   cd "%Gen xx %"
   cd gen\classes
   ```

   Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

4. Execute the batch file MKJAVART with the appropriate case-sensitive parameters exactly as follows:

   Note: It is good practice to put the CA Gen version number in the JAR file's name.

   ```
   MKJAVART genrt.xx.jar EJB
   ```

   Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Alternatively, the CA Gen runtime JAR file is created during the build of the generated application. The file, named genrt.*xx*.jar where *xx* is the CA Gen version number, is located in the <model-directory>\java\deploy.j2ee. This file can be copied to the directory where EAB sources have been copied and may be referenced directly.

Compile the EABs

The javac command is used to compile the EAB source .java file into a .class file. Note the following:

- The command must be executed from the directory above the package name directory. For example, if the EAB has a package name of com.mycompany.accounting and the Java file itself is located in c:\my eabs\com\mycompany\accounting, the javac command must be executed from the c:\my eabs directory.
- Specify to the javac command the source and target versions to ensure compatibility with the compiled generated code.
- Specify a classpath that includes the CA Gen runtime JAR file.
- Specify the files to compile with the wildcard asterisk (*). The asterisk is used to cause the EAB file plus its import and export view class files to be compiled as a single unit.

  For example:

  ```
  javac -source 1.5 -target 1.5 –classpath "c:\my eabs\genrt.xx.jar"
  com\mycompany\account\demoeab*.java
  ```

  Note: *xx* refers to the current release of CA Gen. For the current release number, see the *Release Notes*.

Deploy the EABs in the EAR

After the EABs have been compiled, they are ready for packaging into a JAR file and included into the EAR file.

To package the class files into a JAR, use the JAR command.

To insert the JAR file into the EAR file, specify the JAR files name and location to the CA Gen Build Tool's EAR File Assembly Details dialog. From the items listed on the left, choose Additional File. In the panel on the right, click the Add button to display a file chooser dialog. Navigate to the directory holding the JAR file, select the JAR file and click Open. Repeat the process until all required JARs have been specified.

Continue with the assembly and deployment process.

Chapter 10: Component Based Development

In component based development (CBD), logic for an application can be separated into separate functional elements or components.

Each component is represented by an implementation model and a specification model. An implementation model contains the actual programming logic of the component. The specification model contains interface definition for the component. Specifically, this model contains the procedure step and action block signatures as well as data model elements needed for users of the component.

The final player in CBD is the consuming model. This model imports the specification model and uses the interface definitions to make calls to the components. During the assembly process, the executable portions of the implementation model are combined with the built consuming model to create a complete application. A consuming model may consume more than one component. The consuming model could be a component and be consumed itself.

Each public operation in the component may be transactional or subtransactional. Transactional public operations are modeled as procedure steps. Subtransactional public operations are modeled as action blocks in the implementation model and external action blocks in the specification model.

This chapter discusses additional items of interest to customers who are using CBD techniques.

Restrictions

The consumption of UI-bearing components (client procedure steps) is not supported at this time.

Package Name Settings

To allow components to be easily consumed, the following package name settings are recommended:

- In the implementation model, define a Package Name at the Model Level.
- In the consuming model, import the spec model into its own Business System and set its package name to that of the implementation model.

Consume Subtransactional Components

Subtransactional public operations are modeled as action blocks in the implementation model and external action blocks in the specification model. The subtransactional public operations should be delivered to you as a JAR file. The sections below describe the tasks that need to be performed to successfully consume subtransactional public operations.

Assembly

During the assembly process, you will specify the JAR files containing the subtransactional public operations that will be included in the EAR file.

Follow these steps:

1. Specify the JAR file's name and location to the CA Gen Build Tool's EAR File Assembly Details dialog
2. Choose Additional File from the items listed on the left
3. Click the Add button on right panel
   The file chooser dialog is displayed
4. Navigate to the directory holding the JAR file, select the JAR file and click Open
5. Repeat the process until all required JARs have been included

You can now continue with the assembly and deployment process.

Consume Transactional Components

Transactional public operations are modeled as procedure steps.

In the consuming model, the transactional public operations should be located in separate business systems. *Do not* generate these procedure steps.

During deployment, specify the assemblies that implement the transactional public operations to the CA Gen Build Tool so the assemblies may be included in the JAR file.

To insert the JAR file into the EAR file, specify the JAR files name and location to the CA Gen Build Tool's EAR File Assembly Details dialog. From the items listed on the left, choose Additional File. In the panel on the right, click the Add button to display a file chooser dialog. Navigate to the directory holding the JAR file, select the JAR file and click Open. Repeat the process until all required JARs have been specified.

Continue with the assembly and deployment process.

Chapter 11: Handcrafted Clients

Customers may access the CA Gen EJB Servers from handcrafted clients. The goal is to allow the CA Gen EJB Server to be as easy to access as handcrafted EJB Servers. The CA Gen EJB Servers are called using Java RMI.

View Definitions

CA Gen EJB Servers have an import and export view, which sends data to and from the server respectively. The next sections describe the data representation and layout of the generated servers.

View Objects

The view data is generated into Import and Export Objects. Some of the benefits of the functionality built into the view objects are:

- Comprehensive—View data and system data together.
- Hierarchical—More intuitive access paths to the data.
- Object-oriented—Allows more sophisticated manipulation of the data.
- Serializable—Allows object state to be saved and restored.
- No Runtime—Enables easier transmission/distribution/serialization.
- Data Validation—(Immediate) Always keeps view in a consistent state.
- Clone able—Built-in support for cloning.
- Reset able—Multi-level reset ability gives greater control.
- GroupView Support—Allows row-level operations or manipulations.

Import Object

The Import objects are generated in classes called <EJB name>Import based on the import view designed in the model for the CA Gen EJB Server. It contains the actual import data for the execution of the server. The Import objects are also responsible for all data validation that is done on the attributes on the export views.

Even if the server does not contain any import views, the Import object is still generated since the system level data (Command, NextLocation, and so on) are always present. The developer must instantiate an Import object and populate it with the data. The instance is then passed onto the execution methods as a parameter.

The Import objects are stateful. In fact, they exist to hold a state. Therefore, the developer must take care not to use a particular instance of one of these classes between different threads in a multi-threaded application.

Export Object

The Export objects are generated in classes called <EJB name>Export based on the export view designed in the model for the CA Gen EJB Server. It contains the actual export data for the execution of the server. The Export objects are also responsible for all data validation that is done on the attributes on the export views.

Even if the server does not contain any export views, the Export object is still generated since the system level data (Command, ExitState, and so on) are always present. The server object execution methods create and populate the Export objects.

The Export objects are stateful. In fact, they exist to hold a state. Therefore, the developer must take care, not to use a particular instance of one of these classes between different threads in a multi-threaded application.

View Example

The easiest way to understand view objects is to look at an example. Later sections in this chapter describe how particular mappings occur.

The following example shows how generators map a given import view in the model to a view object:

```
ADDR_SERVER

IMPORTS:
    Entity View in address
        line1
        line2

    Group View inGV (2)
        Entity View ingroup address
          line1
          line2
```

From the given import view, five EJB classes are generated. The classes and their properties are shown in the following table:

Class	Attributes
AddrServerImport	■ Command ■ NextLocation ■ ClientId ■ ClientPassword ■ Dialect ■ ExitState ■ InEVAddress—Gets AddrServer.InAddress object ■ IngvGV—Gets AddrServer.Ingv object
AddrServer.InAddress	■ Line1Fld ■ Line2Fld
AddrServer.Ingv	■ Capacity (Read-only constant) ■ Length ■ Rows—Gets a AddrServer.IngvRow object ■ this[]—Implicit indexer same as Rows
AddrServer.IngvRow	■ IngroupEVAddress—Gets AddrServer.IngroupEVAddress object
AddrServer.IngroupAddress	■ Line1Fld ■ Line2Fld

The number of classes increases in proportion with the number of entity views, work sets, and group views.

To finish the sample, it is beneficial to look at how a programmer gets and sets the various pieces of data in the views. The following code sample is written in Java.

Instantiate the Import View object:

```
AddrServerImport importView = new AddrServerImport();
```

Set the command system level data:

```
importView.Command = "SEND";
```

Access the InEVAddress line2 attribute:

```
String value = importView.InEVAddress.Line2Fld;
```

Get the maximum capacity of the IngvGV group view:

```
for (int i = 0; i < importView.IngvGV.Capacity; i++)
```

Set the current IngvGV number of rows:

```
importView.IngvGV.Length = 2;
```

Set the line1 attribute of the IngroupEVAddress entity view:

```
importView.Ingv[2].IngroupEVAddress.Line1Fld = "ABC";
```

Reset the InAddress Entity View back to defaults:

```
importView.InAddress.Reset();
```

System Level Properties

The import and export objects contain a set of properties at their top level that is best described as system-level properties. Not all the servers make use of these properties, but they are always present on each cooperative server call. The following table shows the properties of import and export objects.

Objects	Properties
Import Objects	■ Command ■ NextLocation ■ ExitState ■ Dialect ■ ClientId ■ ClientPassword
Export Objects	■ Command ■ ExitState ■ ExitStateType ■ ExitStateMessage

GroupView Objects

Special objects are generated for each group view. The first is the group view object itself. This object holds the Capacity or the maximum number of rows specified in the model. It also holds the Length or the current number of populated rows in the group view. The Length cannot exceed the Capacity or be negative. If you try to do so, it throws IndexOutOfRangeException or ArgumentOutOfRangeException.

To store row data, an object is generated. It represents a row and contains references to all the entity and work set views within the GroupView. This group view object then stores an array of row objects to hold the data. An individual row can be accessed by indexing the Rows property on the group view, or by using the indexer property built into the group view object itself.

Since a row of data is an object, it lets certain row-level operations to be performed, such as looping, cloning, resetting, and tostring.

Data Validation

The import and export views validate the attribute data when it is set. The validation checks performed throws ArgumentExceptions if the new value is not valid. The views perform the following validation checks: permitted values, length/precision, and mandatory/optional.

Default Values

Attribute view data in the import and export objects enforce the default values, as defined in the model. The default value is applied when the object is created or the attribute is reset programmatically.

Data Type Mappings

Since the import and export objects are designed to have no runtime dependencies, the attribute data types must be native data types supported by the EJB Framework. The following table shows the mappings applied:

CA Gen Attribute Definition	EJB Framework Data Type
Text/Mixed Text/DBCS Text	java.lang.String
Number (no decimals, =< 4 digits)	short
Number (no decimals, =< 9 digits)	int
Number (no decimals, >9 digits)	double

CA Gen Attribute Definition	EJB Framework Data Type
Number (with decimals)	double
Number (with precision turned on)	java.math.BigDecimal
Date	java.sql.Date (null represents optional data)
Time	java.sql.Time (null represents optional data)
Timestamp	java.sql.Timestamp (null represents optional data)

Code to Invoke the EJB Server

The following shows a section of code that would be used to invoke the EJB server.

```
...
<Instantiate the importView object>
<EJB name>Import importView = new <EJB name>Import();
<define the importView properties as desired>
...
<Create the InitialContext to resolve the EJB JNDI lookup.>
...
<Using JNDI lookup, resolve “EJB name” and obtain the remote object>
<EJB name>_Remote server = (<EJB name>_Remote) context.lookup("<EJB name>");
...
<Invoke the call method on the EJB>
<EJB name>Export exportView = server.<EJB name>call(importView);
...
<retrieve exportView data>
...
```

Index

F

I

J

L

M

P

S

T

U

V

W