WebSphere® Application Server

# Building Business Solutions with WebSphere

*Version 4.0*

SC09-4432-03

**Note**

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 149.

**Fourth Edition (June 2001)**

This edition replaces SC09-4432-02.

Order publications through your IBM representative or through the IBM branch office serving your locality.

# Contents

# Figures

# Tables

# About this book

This book describes how WebSphere Application Server™ can be used to create an infrastructure for doing business over the Internet and the World Wide Web. It includes discussions of the architecture and features of WebSphere Application Server Advanced and Enterprise Editions, and covers the technologies supported by each edition. Finally, it describes a sample application that serves as an example of how WebSphere Application Server can be used to craft a business solution.

## Who should read this book

This document is written for software engineers, architects and administrators responsible for designing and building end-to-end e-business systems. There is no presumed knowledge about any aspect of WebSphere Application Server. Readers of this book are assumed to be familiar with traditional programming concepts, object-oriented programming, and components.

## Document organization

This document has the following organization:

Part 1: *Understanding WebSphere Application Server* discusses the various WebSphere Application Server editions and the technologies supported by each edition.

- "Chapter 1. WebSphere Application Server Advanced Edition" on page 5 describes the features and architecture of the three versions of the Advanced Application Server: WebSphere Application Server Advanced Single Server Edition, WebSphere Application Server Advanced Developer Edition, and WebSphere Application Server Advanced Edition.
- "Chapter 2. Using JavaServer Pages" on page 17 describes the WebSphere Application Server implementation of the Sun JavaServer Pages™ (JSP) specification.
- "Chapter 3. Using servlets" on page 25 describes the WebSphere Application Server implementation of servlets.
- "Chapter 4. Using enterprise beans" on page 33 describes the WebSphere Application Server implementation of the Sun Enterprise JavaBeans™ specification.
- "Chapter 5. Developing Web applications" on page 65 describes how JSP pages, enterprise beans, and servlets can be used to create business applications that can be accessed over the World Wide Web.

- "Chapter 6. WebSphere Application Server Enterprise Edition" on page 73 describes the features and architecture of WebSphere Application Server Enterprise Edition.
- "Chapter 7. Using TXSeries" on page 79 describes IBM TXSeries™.

Part 2: *Using WebSphere Application Server* discusses a sample application that illustrates different ways that WebSphere Application Server can be used to implement a business system.

- "Chapter 8. Overview of the sample application" on page 109 describes the sample application scenario and goals.
- "Chapter 9. Sample application design" on page 113 describes how the sample application is designed.
- "Chapter 10. Implementing the sample application" on page 119 describes how the sample application is implemented using various features of WebSphere Application Server Advanced and Enterprise Editions.
- "Chapter 11. Technical details of the sample application" on page 125 describes technical details of each component of the sample application.
- "Chapter 12. Extending the sample application" on page 145 describes some ways to extend the sample application using additional WebSphere products and features.

## Related information

For further information on the topics discussed in this manual, see the following documents:

- *Getting Started with WebSphere Application Server*
- *Writing Enterprise Beans in WebSphere*
- The Enterprise Edition InfoCenter
- The TXSeries Encina® and CICS® product documentation
- The IBM VisualAge™ for Java™ product documentation

## Conventions used in this book

WebSphere Application Server Enterprise Edition documentation uses the following typographical and keying conventions.

*Table 1. Conventions used in this book*

| Convention | Meaning |
|---|---|
| **Bold** | Indicates command names. When referring to graphical user interfaces (GUIs), bold also indicates menus, menu items, labels, and buttons. |
| `Monospace` | Indicates text you must enter at a command prompt and values you must use literally, such as commands, functions, and resource definition attributes and their values. Monospace also indicates screen text and code examples. |

*Table 1. Conventions used in this book (continued)*

| Convention | Meaning |
|---|---|
| *Italics* | Indicates variable values you must provide (for example, you supply the name of a file for *fileName*). Italics also indicates emphasis and the titles of books. |
| Ctrl-*x* | Where *x* is the name of a key, indicates a control-character sequence. For example, Ctrl-c means hold down the Ctrl key while you press the c key. |
| Return | Refers to the key labeled with the word Return, the word Enter, or the left arrow. |
| % | Represents the UNIX command-shell prompt for a command that does not require **root** privileges. |
| # | Represents the UNIX command-shell prompt for a command that requires **root** privileges. |
| C:\> | Represents the Windows NT® command prompt. |
| > | When used to describe a menu, shows a series of menu selections. For example, "Click **File > New**" means "From the **File** menu, click the **New** command." |
| | When used to describe a tree view, shows a series of folder or object expansions. For example, "**Expand Management Zones > Sample Cell and Work Group Zone > Configuration**" means:<br>1. Expand the Management Zones folder<br>2. Expand the management zone named Sample Cell and Work Group Zone<br>3. Expand the Configurations folder<br>**Note:** An object in a view can be expanded when there is a plus sign (+) beside that object. After an object is expanded, the plus sign is replaced by a minus sign (-). |
| + | Expands a tree structure to show more objects. To expand, click the plus sign (+) beside any object. |
| - | Collapses a branch of a tree structure to remove from view the objects contained in that branch. To collapse the branch of a tree structure, click the minus sign (-) beside the object at the head of the branch. |
| Entering commands | When instructed to "enter" or "issue" a command, type the command and then press Return. For example, the instruction "Enter the **ls** command" means type **ls** at a command prompt and then press Return. |
| [ ] | Enclose optional items in syntax descriptions. |
| { } | Enclose lists from which you must choose an item in syntax descriptions. |
| \| | Separates items in a list of choices enclosed in braces ({ }) in syntax descriptions. |
| ... | Ellipses in syntax descriptions indicate that you can repeat the preceding item one or more times. Ellipses in examples indicate that information was omitted from the example for the sake of brevity. |

*Table 1. Conventions used in this book (continued)*

| Convention | Meaning |
|---|---|
| IN | In function descriptions, indicates parameters whose values are used to pass data to the function. These parameters are not used to return modified data to the calling routine. (Do *not* include the IN declaration in your code.) |
| OUT | In function descriptions, indicates parameters whose values are used to return modified data to the calling routine. These parameters are not used to pass data to the function. (Do *not* include the OUT declaration in your code.) |
| INOUT | In function descriptions, indicates parameters whose values are passed to the function, modified by the function, and returned to the calling routine. These parameters serve as both IN and OUT parameters. (Do *not* include the INOUT declaration in your code.) |
| $CICS | Indicates the full pathname where the CICS product is installed; for example, C:\opt\cics on Windows NT or /opt/cics on Solaris. If the environment variable named CICS is set to the product pathname, you can use the examples exactly as shown; otherwise, you must replace all instances of $CICS with the CICS product pathname. |
| CICS on Open Systems | Refers collectively to the CICS products for all supported UNIX platforms. |
| TXSeries CICS | Refers collectively to the CICS for AIX, CICS for Solaris, and CICS for Windows NT products. |
| CICS | Refers generically to the CICS on Open Systems and CICS for Windows NT products. References to a specific version of a CICS on Open Systems product are used to highlight differences between CICS on Open Systems products. Other CICS products in the CICS Family are distinguished by their operating system (for example, CICS for OS/2 or IBM mainframe-based CICS for the ESA, MVS, and VSE platforms). |

## How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other WebSphere Application Server Enterprise Edition documentation, send your comments by e-mail to wasdoc@us.ibm.com. Be sure to include the name of the book, the document number of the book, the version of WebSphere Application Server Enterprise Edition, and, if applicable, the specific location of the information you are commenting on (for example, a page number or table number).

# Part 1. Understanding WebSphere Application Server

The IBM WebSphere Application Server offers customers a comprehensive set of middleware for designing, implementing, deploying, and managing a new generation of business applications. These applications can range from a simple Web site storefront to a complete revision of an organization's computing infrastructure.

WebSphere Application Server can help you to realize the promise of electronic commerce. It is a set of software products and architectures that enables customers to develop and manage high-performance Web sites and integrate those Web sites with new or existing non-Web business systems. It is focused on businesses that want to:

- Use the latest technologies to establish a powerful Web presence or upgrade their current Web presence
- Develop distributed, enterprise wide business systems and applications
- Integrate their Web presence with their non-Web systems and applications

WebSphere Application Server delivers the essential elements of an enterprise-class component server solution. In particular, it delivers an environment that:

- Allows for the natural separation of responsibilities across a design and development team
- Delegates object management to the system, freeing designers and programmers from concerns about object management
- Leverages operating system capabilities to deliver optimal performance, scale, integrity and security

WebSphere Application Server achieves this environment by providing:

- A framework and runtime implementation for object management built upon industry standards such as the Java 2 Platform Enterprise Edition (J2EE™) specification and the Common Object Request Broker Architecture (CORBA).
- A visual development tool that organizes program resources and generates the code necessary for integration with framework services.
- Application adaptors and connectors that support a variety of persistent data stores and allow objects to be rendered from existing data and transaction systems.
- A systems management tool to associate objects with servers and coordinate system updates.

The WebSphere product family's unified programming model allows developers to create systems that can be deployed on many different platforms and application servers. This paradigm enables code to be reused and protects computing infrastructure investments.

The WebSphere multi-tiered, distributed design separates presentation, business logic and data resources into different tiers. A variety of clients can access the enterprise business applications and data any time and from any location.

WebSphere offers the power of reuse. This includes reuse of newly developed code, existing procedural applications, and enterprise data. The portability of developed and deployed EJB applications allows for system scalability and interoperations with non-WebSphere environments. The interoperability between the WebSphere Application Servers allows systems to grow from simple business applications to complex enterprise wide systems running on high-end application servers. The variety of hardware and operating system platforms supported by WebSphere allows business to choose the platforms that are right for their needs.

WebSphere Application Server complies with the Java™ 2 Platform, Enterprise Edition (J2EE™) specification. J2EE reduces the cost and complexity of developing multi-tiered applications, enabling them to be easily deployed and expanded to meet the needs of an organization. It defines an architecture that includes a standardized model for developing multi-tiered applications and a platform for hosting such applications. J2EE supports Java clients, applets, servlets, JavaServer Pages (JSP pages), and enterprise beans. Its standard services include HTTP, HTTPS, the Java Transaction API (JTA), the Java Database Connectivity (JDBC) API, the Java Message Service (JMS) API, the Java Naming and Directory Interface (JNDI) API, and the J2EE Connector Architecture.

Part 1 of this book provides an overview of the features and capabilities of WebSphere Application Server. It includes the following topics:

"Part 2. Using WebSphere Application Server" on page 107 describes how to apply WebSphere Application Server's features and capabilities in a business application.

# Chapter 1. WebSphere Application Server Advanced Edition

WebSphere Application Server Advanced Edition offers customers a comprehensive set of middleware for designing, implementing, deploying, and managing a new generation of business applications. It is available in three different versions: WebSphere Application Server Advanced Single Server Edition, WebSphere Application Server Advanced Developer Edition, and WebSphere Application Server Advanced Edition. For more information on the different versions of Advanced Application Server, refer to "WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition" and "WebSphere Application Server Advanced Edition" on page 10.

## WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition

Both WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition are used to build active Web sites and Web applications such as servlets. They combine the portability of server-side business applications with the performance and manageability of Java technologies to offer a comprehensive platform for designing Java-based Web applications. They enable powerful interactions with enterprise databases and transaction systems and offer run-time, development, and system management environments. This section describes the following elements of these two versions of the Advanced Application Server:

- "Features"
- "Run-time and system management architecture" on page 6
- "Application development environment" on page 9

For more complete information, see the InfoCenter. It is available from the WebSphere Application Server library page at www.software.ibm.com/webservers/appserv/library.html.

### Features

Both the Advanced Single Server Edition and Advanced Developer Edition offer a single-machine Web application server. A typical installation of one of the versions supports a Web site. To create applications, developers implement extended Hypertext Markup Language (HTML) content. Developers build and test the Web site, then publish updates into the active site.

WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition provide an extended set of HTML tags, and a set of application development tools to support page creation (scripting) with these tags. Supported content and page styles include:

- HTML with embedded images, sounds, and video clips.
- HTML with embedded client side scripts (such as JavaScript).
- JavaServer Pages (JSP), which are active HTML pages with an extended set of markup tags. JSP tags allow programmers to develop data-driven page content and page flows. See "Chapter 2. Using JavaServer Pages" on page 17 for more information.
- Servlets written to the Sun Microsystem Servlet specification. A servlet engine is included with the Advanced Application Server. See "Chapter 3. Using servlets" on page 25 for more information.
- JavaBeans components written to the Sun Microsystem JavaBeans™ specification.
- Enterprise beans written to Sun Microsystem's Enterprise JavaBeans (EJB) specification. See "Chapter 4. Using enterprise beans" on page 33 for more information.
- Extensible Markup Language (XML) files, which are used to store administration information such as configuration data and state information.

For more information on the typical architecture of a Web application supported by WebSphere Application Server, see "Chapter 5. Developing Web applications" on page 65.

## Run-time and system management architecture

Figure 1 on page 7 shows the run-time architecture of both WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition.

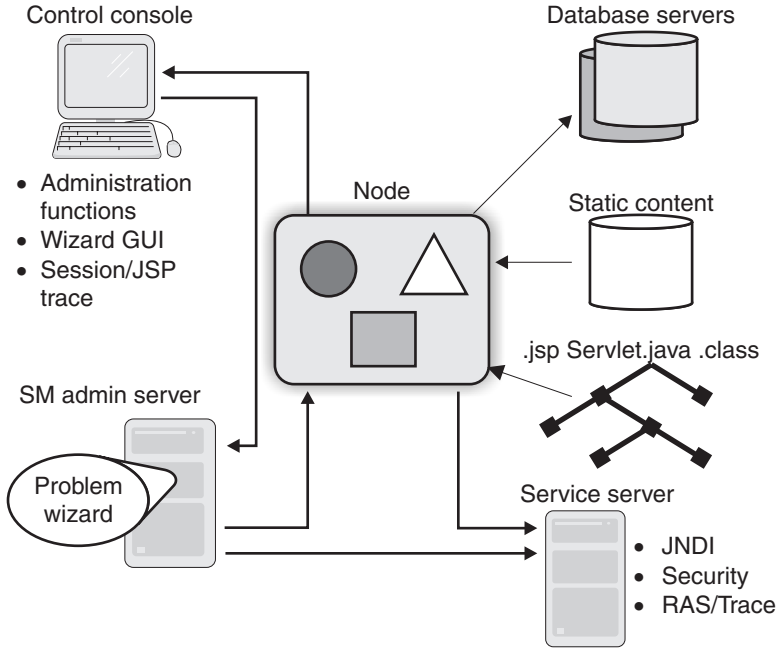


*Figure 1. WebSphere Advanced Application Server run-time architecture*

The components of the run time and system management are as follows:

**Note:** Most run-time and system management functions are internally implemented. Programmers and administrators see very few of these functions.

### Web application server

The Web application server is the run time environment for deployed servlets and compiled and deployed JSP pages. It also implements containers for servlets. The Web application server integrates with the Services server to provide security to the resources it manages. It is multithreaded and implements basic thread management and scheduling policies for managing incoming requests.

The Web application server provides an implementation of the Java Transaction Service (JTS) that is integrated with Java Database Control (JDBC™) and XA-compliant databases. It also provides a connection manager that improves interaction between Web applications and databases.

### Web server

WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition integrate with many different Hypertext Transfer Protocol Daemon (HTTPD) servers,

including the IBM, Netscape and Apache Web servers. They also provide two plug-in extensions that augment the Web server:

- **Services**—The Services plug-in enables the Web server to use the same security, naming, and trace services that are used by the application server run time.
- **NCF**—The NCF plug-in enables the Web server to route uniform resource locators (URLs) that match administrator-defined filters to the Web application server. These URLs represent either servlet instances or JSP pages (which are automatically compiled and executed as servlets). Explicit administration is not necessary for JSP pages. The NCF plug-in uses the Object Request Broker (ORB) to call the Web application server for executing servlet and JSP pages. The ORB supports intra-system optimizations for performance and propagates Web server context information such as user identification on calls to the Web application server. All versions of the Advanced Application Server run time and the ORB can optionally be configured so that their threads run within the Web server under threads that are controlled by the Web server.

### System Management server

The System Management (SM) server provides a single-site solution for integrating servlets, JSP applications, and their data with the overall management of the Web site. SM server functions include:

- **User management and group management**—Support management of Web site users and optional management of operating system user IDs and groups. User, group, and security management can also be done through the native Web server and operating system management tools.
- **Authorization management**—Enables administrators to control access to URLs, servlets, JSP pages, and enterprise bean homes. Authorization management is optional.
- **Definition and deployment**—These functions support the installation and deployment of enterprise beans and Java archive (JAR) files, servlet instances, and JSP pages. Placing files at certain URLs and directories triggers definition and deployment. Enterprise bean support must be explicitly enabled; by default, only named servlets and their properties are managed.
- **Operation**—This function allows the administrator to start and stop the Service server, stop servlets, and manage database connections and transactions.
- **Problem determination wizard (Apache Web server only)**—Web application servers that are integrated with the Apache Web server can use the problem determination wizard to read trace logs, merge trace and operation logs, detect problems, and recommend corrective actions.

### Services server

The Services server implements utility functions and applications needed by the application server's Web server plug-ins and the SM server. These include the security application that supports authentication, authorization, tracing, and auditing; the name server; and trace services. These applications and services can be configured to run in a separate server or within the Web application server itself.

This server's presence and operations are transparent to system administrators and application developers.

## Application development environment

WebSphere Studio is the application development environment for all versions of the Advanced Application Server. It is a multipurpose Web site development tool that can be used to create everything from personal Web pages to Web sites that serve as front ends for e-business applications. WebSphere Studio provides a tool set suite for developing HTML content and can be integrated with other content development tools.

The WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition application development environments support the following:

- **Static and active HTML**—Many URLs map to static content—HTML files with hypertext links, embedded pictures, sounds, and video. Some HTML files are actively generated by the client; other HTML files are actively generated by the server. Client-active HTML contains client-side scripts executed in the browser, such as JavaScript scripts or applets. Server-active HTML can be JSP pages or explicitly coded servlets.
- **Servlets**—Servlets can be developed in WebSphere Studio and run in all versions of the Advanced Application Server. They are automatically reloaded when their source code changes. A named servlet instance is explicitly configured with its initialization values by the SM server. For more detailed information on how WebSphere Application Server implement servlets, see "Chapter 3. Using servlets" on page 25.
- **JSP pages**—All access to relational databases occurs through tool-generated beans. JSP developers access the beans through bean markup tags in JSP page. JSP files are implicitly compiled, deployed, and installed when they are published from application development tools. For more detailed information on how WebSphere Application Server implements JSP pages, see "Chapter 2. Using JavaServer Pages" on page 17.
- **Database support**—All versions of the Advanced Application Server define an extended HTML tag set for performing the following database operations:
  - Defining a table's signature (for instance, column names and values)
  - URL data type for column values

- Create, Remove, Update, and Delete (CRUD) operations on a named table
- Basic query support
- Formatting tables from query results

## WebSphere Application Server Advanced Edition

WebSphere Application Server Advanced Edition builds on WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition. It introduces multi-machine server capabilities for applications built to the Enterprise JavaBeans (EJB) Specification (including supporting services such as workload management) and enables Web applications to be more tightly integrated with existing business systems.

For more complete information on the Advanced Application Server, see the Advanced Edition InfoCenter. It is available from the WebSphere Application Server library page at www.software.ibm.com/webservers/appserv/library.html.

### Advanced Application Server features

The full version of Advanced Application Server enables you to develop scalable and transactional Web applications. It can run on multiple machine systems (as opposed to the single-machine environment provided by WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition). It supports enterprise beans that implement business logic and access persistent data; distributed system management; and shared name, security, and transaction services.

See "WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition" on page 5 for a detailed description of the capabilities provided by WebSphere Application Server Advanced Single Server Edition and WebSphere Application Server Advanced Developer Edition.

### Run-time and system management architecture

The Advanced Application Server run-time environment supports clustered and distributed systems in addition to the single-machine server. Figure 2 on page 11 shows a high-level view of the Advanced Application Server system architecture.

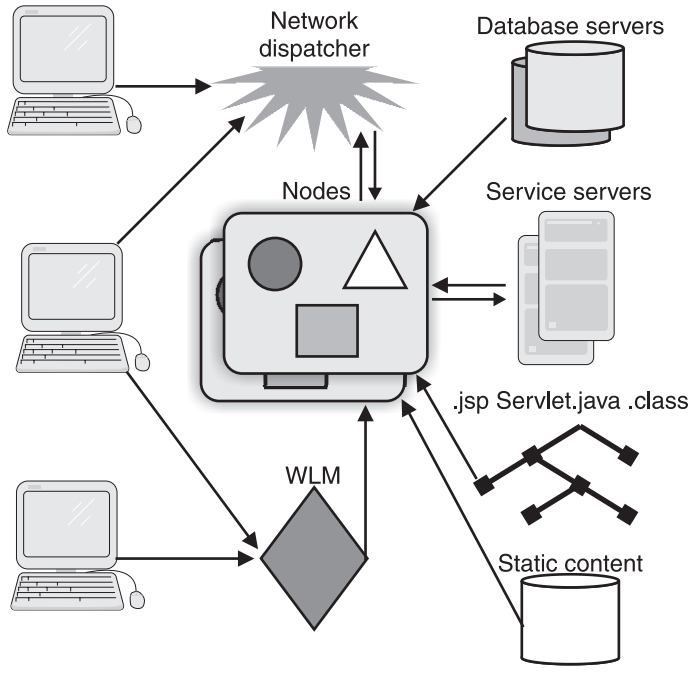


*Figure 2. A high-level view of the Advanced Application Server system architecture*

The elements of the Advanced Application Server system are as follows.

### Topology

The topology of an Advanced Application Server system contains three elements:

**Node:** A node represents a physical system on which the advanced application server run time is installed. Multiple instances of a Web application server can be defined and active on a node. Each instance is a single, multithreaded operating system process. Figure 3 on page 12 shows a typical node configuration.

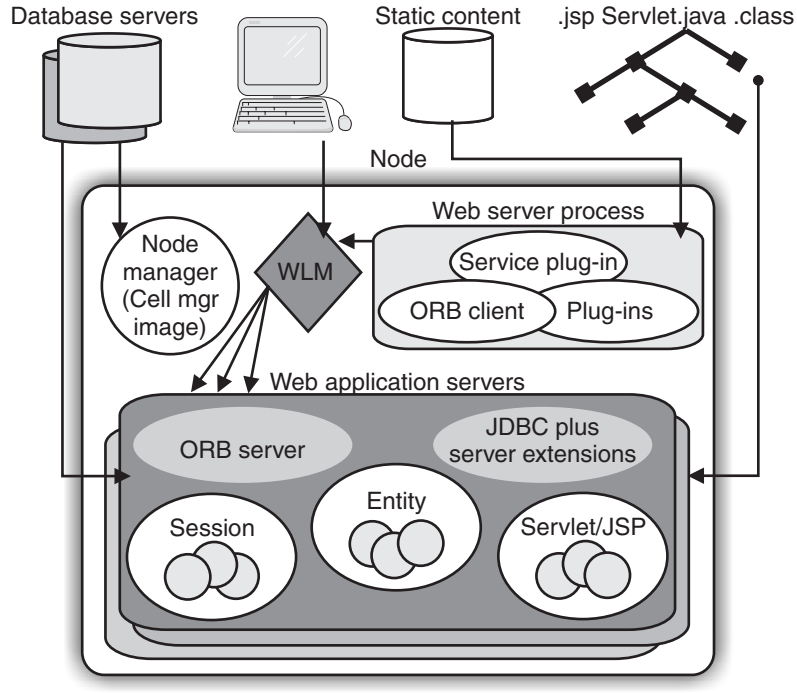


*Figure 3. A typical node in an Advanced Application Server system*

**Server groups:** A server group is a named collection of Web application server instances, which can optionally be configured onto different hosts. All instances in the group are connected to the same databases and same file systems. Web application server instances from different server groups can be configured in the same node.

Administrators define server groups for availability, performance, and manageability reasons. All Web application server instances in a server group are clones that run the same servlets, JSP pages and enterprise beans, and that have the same containers configured with the same policies. Client applications, application logic in a server group and other server groups see a server group as a single Web application server instance.

Each Web application server instance in the server group has a name. For example, the Underwriting ApplicationServerGroup can have an instance called UAS1 on the machine pinnacle.uas.com and an instance called UAS2 on the machine kaitlin.uas.com.

**Cell:** A cell is a named collection of nodes. A node can be in only one cell. A server group does not span cells.

In general, the nodes in a cell do not access the same file systems and databases. However, all nodes onto which servers from a server group are installed must have access to the following resources:

- The file systems holding the HTML, servlets, JSP pages, and Java classes
- The databases and database servers referenced by enterprise beans deployed into the server group

A cell manager notifies all of its active node managers when a server group membership changes

### Workload and availability management

Workload management optimizes the distribution of client requests. Incoming work requests are distributed to the application servers and other objects that can most effectively process the requests. Workload management also provides failover (redirection of client requests to other servers when a server is not available).

Clones (replicas of application objects) and server groups can be used to create multiple instances of entire application servers. The Advanced Application Server automatically manages workloads for clones. Administration servers can also participate in workload management for failover support.

Transactions, sessions, and workload balancing provide the basis for bypassing failures and improving availability. If a Web application server instance fails, the node manager detects the failure, rolls back the sessions and transactions that are active in that process, and restarts the process. The node manager also notifies the cell manager that a failure has occurred.

The Object Transaction Monitor (OTM) implements workload and availability management for server groups. OTM balances transactions and sessions over Web application server instances in a server group. Nontransactional and nonsession requests are randomly routed over the instances.

### System management

The System Management (SM) tools provide the following additional services:

- **Cell manager**—The cell manager resides in its own server group within the cell and uses enterprise bean transactions, relational database persistence and OTM to achieve scalability and availability. Cell manager instances (the server processes running the cell manager application) can be assigned to any node in the server group. However, they are typically consolidated into a small number of nodes.
- **Node manager**—The node manager is responsible for managing the local server instances and configuration information. Node managers perform

operations on behalf of a cell manager. Both the cell manager and the node manager are implemented as enterprise bean applications.

Node managers and cell managers also monitor system activity to check availability and can use clustered operating system process membership services if they are available.

The system management tools also provide support for defining, configuring, and operating Web application servers and server groups.

- **Single point of management**—This function manages remote nodes from the SM console.
- **Single logical image**—This function maps operations performed at the Server Group level (such as start and stop) onto operations on the individual server instances in the group. This eliminates the need to perform the same operation multiple times for each clone in the group.

## Object services and support services

Object and support services are shared among all nodes and server groups within an enterprise. They include the following:

**Name service**
Application server clusters within an enterprise share a global name space. Name servers implement the name space, and an enterprise may have multiple name servers.

**RAS service**
The Reliability, Availability, and Serviceability (RAS) service provides an enterprise wide view of the logs for the run-time, application RAS, and trace. Application and systems RAS records are written to a node-specific log, and the node manager, cell manager and RAS service collaborate to provide a centralized view of RAS and trace information on demand.

**Security service**
Security services provide access to user registry information and authorization rules for individual Web application servers. This enables them to perform local authentication and authorization.

**Transactions**
The Advanced Application Server supports distributed transactions and client-demarcated transactions.

**Persistence**
Persistence is implemented through calls to external databases that are compatible with Java Database Connectivity (JDBC). The databases may be shared among Web application server instances. The Advanced Application Server also supports databases that are shared with other applications.

These services are described in more detail in "Object Services" on page 49.

## Advanced Edition application development environment

The Advanced Application Server supports Web applications that can be developed by using WebSphere Studio (which is included with the Advanced Application Server), VisualAge for Java, and other Java and HTML development tools.

### Application model

The application model for the Advanced Application Server is object-oriented business logic backed by relational database systems. Applications can be integrated with Web driven thin or thick clients. The Advanced Application Server also provides support for limited integration with existing procedural applications running in application servers.

An application typically has four parts:

- HTML and JSP pages provide the user interface and form flow for the application.
  - The JSP pages access the state data through enterprise beans. JSP tags are used to access JavaBeans components that in turn access enterprise beans. (The application development tools generate JavaBeans components for accessing enterprise beans through JSP tags.)
  - To improve performance, use stateless JSP pages. The Advanced Application Server supports clustered servers to improve performance and availability, but caching JSP state information prevents these benefits from being realized. As an alternative, the Advanced Application Server provides tags for accessing session bean data and storing JSP page data within a cluster.

  For more information about JSP pages, see "Chapter 2. Using JavaServer Pages" on page 17.
- Enterprise beans implement business logic, transactional operations, and access to databases. They act as a bridge between Web applications and non-Web computer systems. For more information, see "Chapter 4. Using enterprise beans" on page 33.
- Servlets coordinate work between the other components of the application. They also can perform tasks such as dynamically generating Web page contents. For more information, see "Chapter 3. Using servlets" on page 25.
- Relational databases are used to implement persistence and query functions for enterprise beans. Although new databases are generally defined for new applications, the Advanced Application Server also allows the use of existing databases in new applications.

"Chapter 5. Developing Web applications" on page 65 describes the development of Advanced Application Server Web applications in more detail.

For an example of an application that is developed with the Advanced Application Server, see “Part 2. Using WebSphere Application Server” on page 107.

### WebSphere Studio

WebSphere Studio is included with the Advanced Application server. For more information on this product, see “Application development environment” on page 9.

### VisualAge for Java

VisualAge for Java is an integrated development environment that supports the complete cycle of Java program development. Although it is not a part of the Advanced Application Server, VisualAge for Java is tightly integrated with the WebSphere Application Server environment. This integration enables VisualAge developers to develop, deploy, and test their Java programs without leaving the VisualAge program. It also helps developers to manage the complexity of the enterprise environment and is capable of automating routine steps.

You can use the VisualAge for Java visual programming features to quickly develop Java applets and applications. In addition, SmartGuides lead you quickly through many tasks, including the creation of applets, servlets, applications, Java beans, and enterprise beans built to the Enterprise Java Beans (EJB) Specification. It also enables you to import existing code and export code as required from the underlying file system.

# Chapter 2. Using JavaServer Pages

JavaServer Pages (JSP) are server-side components that allow developers to embed scripting commands inside a HTML page and make use of server components. JSP pages dynamically generate HTML, Extensible Markup Language (XML), and other structured documents inside a server.

JSP pages use the WebSphere Application Server implementation of the JavaSoft servlet application programming interface (API). A Java class corresponds to and implements each JSP page. JSP pages make use of the JavaBeans component architecture and use Java as a scripting language. They use servlets as compiled page objects.

This section discusses the following JSP page-related topics:



This section is intended to provide background information necessary for understanding the WebSphere family example application described in "Part 2. Using WebSphere Application Server" on page 107. It does not focus on the practical development issues of creating JSP pages. For more detailed information on their development, see the following:

- The IBM VisualAge for Java product documentation and samples
- The WebSphere Studio documentation and samples

## How JSP pages work

JSP pages can be invoked directly or through a servlet. A JSP file is identified by a .jsp extension. When a JSP page is invoked, it is parsed into a temporary Java source file. This temporary source file is then compiled into a servlet. The output from the servlet is standard HTML, which the browser interprets and displays. This process is illustrated in Figure 4 on page 18.

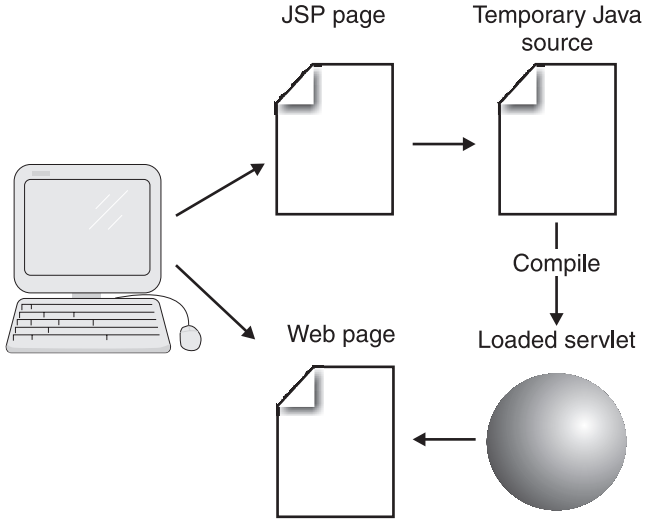


*Figure 4. JSP program flow*

A processor that can deliver requests from a client to a page and responses from a page to a client interprets the JSP page. All JSP page processors must support the HTTP protocol for request and responses; they can optionally support additional protocols.

All JSP page processors support scripting elements written in Java. A Java run time is needed to run scripts written in Java. If the tag extension mechanism is implemented, a Java run time is also needed to run the tag handlers.

The JSP compiler processes a JSP file and generates a corresponding Java servlet class. The class is then loaded into memory to handle all requests for that page. The class stays in memory until the server shuts down or the source page changes.

During each request to a JSP page, the server checks to see if the JSP page has changed. If there is no change, the server uses the loaded servlet; otherwise the server automatically compiles the JSP page and reloads the servlet.

The first invocation of a JSP page is delayed since the JSP page must be compiled into a servlet and then loaded into memory. (This also applies when a request is made to a JSP page that has changed.) However, subsequent invocations do not experience this delay.

## Invoking JSP pages

JSP pages can be invoked from a Web browser or by a servlet.

Figure 5 shows how a JSP page can be directly invoked from a browser. The JSP page is invoked either as a uniform resource locator (URL) or within a HTML page. After receiving the client request, the JSP page requests information from server components, which perform any necessary processing. The JSP page then inserts the results into the HTML page, which is then displayed by the browser.
Figure 6 shows how a JSP page can be invoked by a servlet. The browser

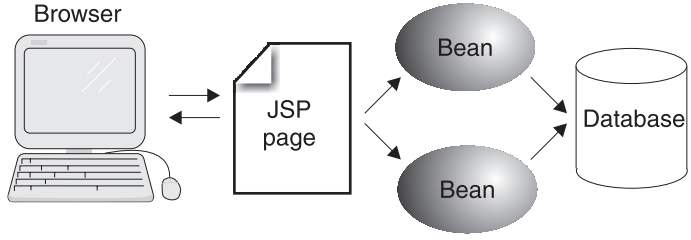


*Figure 5. Browser request to a JSP page*

request is sent to the servlet, which then invokes the JSP page. The servlet interacts with the server components (in this case, Java beans) to perform any necessary processing. The servlet can optionally create a bean to store the results. The JSP page then extracts the required information from the server components and merges the information with the HTML page. The browser then displays the resulting HTML.

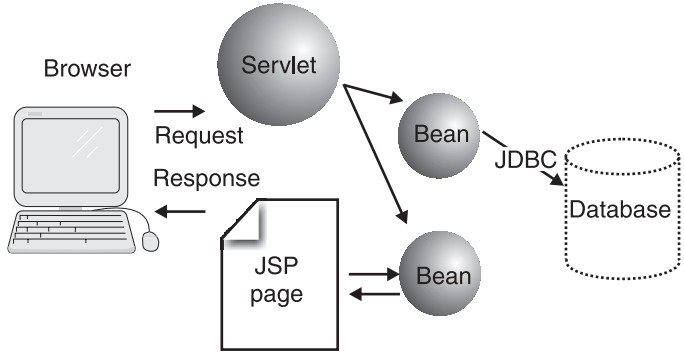


*Figure 6. Servlet request to JSP page*

This method of invoking JSP pages is used in the WebSphere sample application described in "Part 2. Using WebSphere Application Server" on page 107. The JSP pages are invoked by the servlet through a command bean, a Java bean that is used to isolate the servlets from the business logic. The application uses JPS pages to format and display processing results to the client.

## Using Java code in an HTML document

JSP pages support two ways to include Java code in a HTML document:

- A scripting approach in which the Java code is embedded directly inside the HTML document by using tags that are called *scriptlets*.
- A component-centric approach in which the Java code resides inside components such as Java beans. The programming logic resides in the components. Component-centric tags (<BEAN> tags) are used to request information from the components to be inserted into the HTML document.

Scriptlets are typically used to implement simple sequences of commands. Their use is recommended in cases where it does not make sense to use beans. However, using scriptlets extensively makes it difficult to maintain the HTML document. It also makes it more difficult to separate roles in a development team because the HTML and Java code are tightly coupled.

Including Java code by using components separates the business logic from its presentation. It facilitates a cleaner separation of roles in a development team, because the HTML author interacts with components only through a tag set and does not have direct responsibility for programming them. In addition, it enables JSP pages to reuse components.

## Using JSP tags

A *JSP tag* is an HTML or XML representation of a JSP page function that encapsulates the programmatic syntax of the function. JSP tags provide functions for:

- Accessing the properties, methods, request parameters, and session data of JavaBeans components.
- Conditional inclusion of HTML based on bean or data values, request parameters, and session data.
- Defining error pages and page inclusion.
- Running scripts written in the Java programming language or the JavaScript scripting language.

For more detailed information about the basic JSP tag set and tag extension, see the Sun JSP Specification.

JSP tags can be manipulated by a JSP authoring tool and a JSP compiler, both of which are included in the WebSphere Studio tool set. See the WebSphere Studio documentation for more information on how to use these tools.

## Using JSP pages in applications

JSP pages can be used in combination with servlets, HTTP, HTML, XML, applets, and enterprise beans to implement the following application models:

**Two-tiered model**

This application model allows a JSP page or servlet to directly access a resource (such as a database or legacy application) to service a client's request. Its advantages are that it is simple to program and allows the page author to generate dynamic content based upon the request and state of the resource or resources. However, the two-tiered model cannot handle a large number of simultaneous clients, since each client must establish or share a connection to the resources. The two-tiered model is often used as a replacement for Common Gateway Interface (CGI) programs.

**Multitiered model**

This application model comprises at least three tiers. The JSP page in the middle tier interacts with the back-end resources by using an enterprise bean. The EJB server and the enterprise bean provide managed access to resources, which addresses performance issues. The EJB server also supplies transaction and security services. This application model is used by the WebSphere sample application.

**Loosely coupled applications**

Applications that have peer or client/server dependencies are considered to be loosely coupled. They can communicate over an intranet, an extranet, or the Internet. Common examples of loosely coupled applications are supply-chain applications between vendor enterprises. Each application must be isolated from changes in the other applications that depend on it. To achieve this loose coupling, the applications communicate via HTTP, using either JSP-generated HTML or XML.

**Using XML with JSP pages**

JSP pages are an ideal way to process XML input and output. Static XML pages can be generated by writing the XML as a static template within a JSP page. Dynamic XML pages can be generated through JavaBeans components, enterprise beans, or custom tags that generate XML. Input XML can be received from POST or QUERY arguments, then manipulated by scripting or sent directly to JavaBeans, enterprise beans, or custom tags.

Two attributes of a JSP page aid in XML processing. XML fragments can be described directly in JSP pages either as templates for input into a component or as templates for output to be extended with XML fragments. In addition, the JSP tag extension mechanism enables the creation of tags and directives to manipulate XML.

**Redirecting requests**

Data to be sent to the client can vary significantly, depending on the properties of the client. These properties can be directly encoded in a request or stored in a client profile. In either case, the initial JSP page can determine details about the request and then, if necessary, redirect the request to a second JSP page.

This programming model is supported by the underlying servlet APIs. However, the HTTP protocol prevents a redirect of a request if the response stream is being sent back to the client. This makes it difficult to redirect requests in some common situations. To address this, the JSP specification provides buffering on the output stream. The JSP code can redirect the request at any point before flushing the output buffer. Buffering makes it easy to handle error pages, since that is also done by redirecting requests.

**Including requests**

The request reaches an initial JSP page and begins generating a result. The JSP page must dynamically include the contents of another page. These contents can be static or can be dynamically generated by another JSP page, a servlet, or a legacy application. This model is used most often for content that is independent of presentation — for instance, for generating XML that is later converted into another format.

## Advantages of using JSP pages

JSP pages have the following advantages over other Web application architectures:

**Separation of content presentation and generation**

The responsibility for generating content and data is delegated to server components. JSP pages are responsible only for extracting that content and merging it with a HTML document.

**Support for Model-View-Controller architecture**

JSP pages provide better support for a Model-View-Controller (MVC) architecture in Web applications. Prior to the development of JSP technology, servlets were responsible for both the control logic (controller) and dynamic content generation (view). Implementing this dual role made the servlet more difficult to maintain, because changes to the output format required the servlet to be recompiled. However, using JSP pages to generate content eliminates the need to modify and recompile the servlet whenever the output format changes. See "Implementing a Model-View-Controller architecture" on page 67 for more information on this application architecture.

**Separate development team roles**

Encapsulating the business logic in components, using servlets to handle the control logic, and dynamically generating HTML with JSP pages makes it easier to demarcate roles in a Web development team. The JSP file can be developed by an HTML author, and software developers can be responsible for coding the servlets and JavaBeans.

**Improved portability**

JSP pages use standardized, portable components such as the Java scripting language, the JavaBeans component architecture, and HTML. The standardized JSP application programming interface (API) makes JSP pages portable across application servers.

**Java programming benefits**

JSP pages reap the benefits of using Java, including a strong type system, object-oriented code, effective tools, and no memory management problems.

**Using existing skills**

JSP pages build on existing Java and HTML programming skill sets.

**JavaBeans introspection**

Using the underlying JavaBeans component architecture distinguishes JSP pages from other scripting languages. Combining simple tagging with the power of JavaBeans introspection makes it easy to extract data that is to be merged with generated HTML.

# Chapter 3. Using servlets

This section discusses servlets, Java applications that run inside Java-enabled Web servers and enable users to access Web server functionality. They are used to extend the capabilities of Web servers and application servers. Servlets use the WebSphere Application Server implementation of the servlet application programming interface (API). This section discusses the following servlet-related topics:



It provides background information necessary for understanding the WebSphere family example application described in "Part 2. Using WebSphere Application Server" on page 107. It does not focus on the practical development issues of creating servlets. For more detailed information on their development, see the following:

- The IBM VisualAge for Java product documentation and samples
- The WebSphere Studio documentation and samples
- *Writing Enterprise Beans in WebSphere*, which describes how to write servlets that use enterprise beans.

## Servlet programming model

Servlets support a request-and-response programming model. When a client sends a request to the application server, the server sends the request to the servlet. The servlet then constructs a response that the server sends back to the client. The client (usually a Web browser) never directly interacts with a servlet.

Servlets run within the same process as the Web server. The Web server's role is to initialize, invoke, and destroy each servlet instance. Each servlet runs as a separate thread within the Web server process, as shown in Figure 7 on page 26. There is only one instance of the servlet, with multiple threads created to handle multiple client requests. This uses server resources efficiently and enables servlets to interact with their environment.

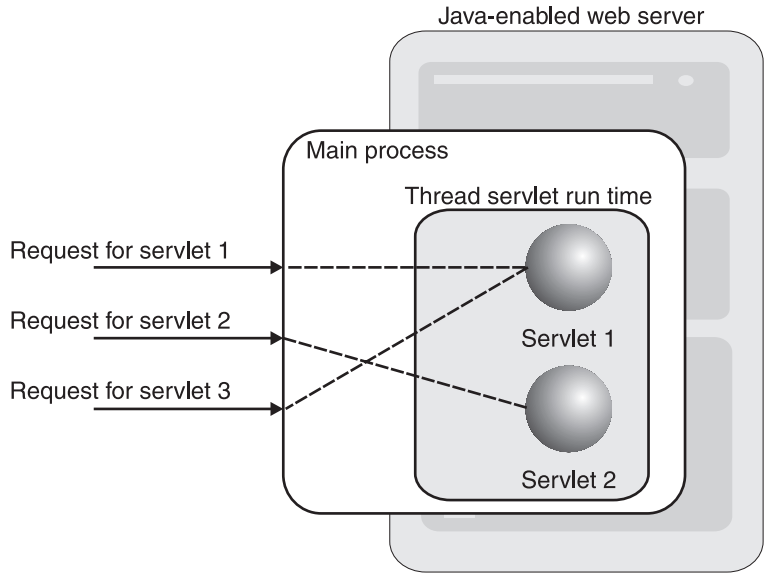


*Figure 7. Servlet execution model*

Servlets can be loaded dynamically when their services are first requested or loaded automatically when the Web server starts.

## Servlet API

Servlets are based on the Java Servlet Development Kit (JSDK) application programming interface (API), which defines a standard interface between a servlet and a Java-enabled server. This makes servlets platform independent and portable across Web servers supplied by different vendors. A Web server communicates with a servlet through the servlet interface. Client requests are made to the server, which then invokes the servlet to handle these requests through the JSDK interface.

A Java class becomes a servlet by implementing the servlet interface. This is done by extending the GenericServlet class (for protocol-independent servlets) or by extending the HttpServlet class (for HTTP-specific servlets). Like an applet, a servlet does not have a main method. This interface defines methods to perform the following tasks:

- Initialize and load the servlet.
- Construct responses for client requests.
- Destroy the servlet when it is no longer needed and release any resources that the servlet has used.

In writing a servlet, a developer implements these methods. Developers can extend them to customize the servlet's operation—for instance, to release database connections when a servlet is destroyed.

## Life cycle of servlets

The typical life cycle of a servlet is as follows:

1. The servlet can be loaded into memory by the invoking application server, or it can be loaded automatically when the application server starts up. There is only one instance of the servlet, with multiple threads created to handle client requests.
2. The application server initializes the servlet by calling the init method.
3. The HTTP server receives incoming client requests. It forwards the requests to the servlet run time, which passes it on to the servlet. Depending on the type of HTTP request, either the doGet or the doPost method is invoked.
4. The servlet processes the request and returns the results through an output stream to the HTTP server.
5. The HTTP server sends the results to the client.
6. The application server unloads the servlet by invoking its destroy method.

The life cycle of a servlet is shown in Figure 8.

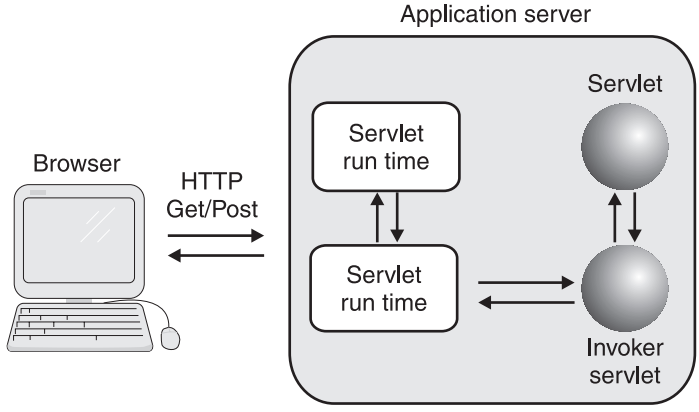


*Figure 8. Basic Flow of Servlet*

## Servlet run-time environment

The WebSphere Application Server servlet run-time environment provides performance and administration support for servlets. The components of the servlet run-time environment are described as follows:

### Servlet containers

Servlets run in special servlet-specific containers, which are primarily intended to improve servlet performance. Servlet containers do not have transaction, persistence, and security support because these services are not needed in the servlet environment. Although servlet containers follow a programming model similar to EJB containers, they cannot be used to deploy and run enterprise beans.

### Servlet queues

A servlet queue dispatches servlet requests from the HTTP server to the servlet process. All clones of a server group share the same queue and process requests from the queue as threads become available If all the clones are busy, the request stays on the queue and eventually backs up at the Web server thread.

### Servlet groups

A servlet group is an EJB server group that runs clones of a servlet. It can run one or more servlet containers and can listen on a servlet queue for incoming requests. Servlet groups are limited to servlet clones running on the same machine.

### Affinity

Affinity allows requests to be directed to specific servlet instances or groups. Two types of affinity are supported:

- **Application affinity**—Directs requests to specific servlet instances by mapping a uniform resource locator (URL) to an instance of a servlet. If an application's requests are returned to a specific client instance (that is, the application is stateful), the server group hosting that application must be configured as a single instance. No clones can be available. These types of servlets are nonscalable.
- **Session affinity**—Directs requests to servlet groups by providing a scalable shared session infrastructure. This improves the performance of clustered HTTP sessions.

## Managing servlet sessions

Clients connect to servlets by using sessions. A session is a continuous connection from a browser over a fixed period of time. It is not a permanent connection.

Servlets have built-in session management. The servlet API defines a number of classes and interfaces to manage sessions. Every client is associated with a HttpSession object. This object is used by a servlet to store or retrieve information about that client. The HttpSession object maintains information about a single session. Java objects and database connections can also be saved in a session object.

A servlet gets the current HttpSession object by calling the request object's getSession method. This method takes a Boolean parameter. If the parameter is true, a new session is created when a new session is detected. If the parameter is false, the method returns null if a new session is detected.

Sessions can be terminated in two ways:

- Automatically by the server after a fixed time period (generally, 30 minutes)
- Manually by the servlet

When a session expires, the HttpSession object and the data values it contains are removed from the system. To save session data beyond its life cycle, you must store it in a persistent resource such as a database. For instance, session data can be saved and analyzed for trends in site usage.

For more information on maintaining state information, see "Maintaining state in Web applications" on page 68. For more information on session management, see the JavaSoft Servlet Specification.

## Advantages of using servlets

Servlets offer a number of advantages as a technology for developing distributed applications. These advantages are described in the rest of this section.

**Portability**

Servlets are written in Java, which makes them portable across platforms. Because the servlet API defines a standard interface between a servlet and a Web server, servlets are also portable across Java-enabled application servers.

**Persistence**

Servlets remain in memory after they are loaded and can maintain system resources (such as database connections) between requests. They can also be loaded automatically when the Web server is started.

**Benefits of the Java programming language**

Servlets reap the benefits of using the Java programming language, including a strong type system, object-oriented code, and good memory management.

**Replacement for CGI**

Servlets are often used to replace CGI in Web applications. CGI supports a request-and-response programming model that is similar to the servlet programming model. However, CGI has the following problems:

- In the CGI model, each application component is an independent program. This limits the number of requests that a Web server can

handle, since a new process must be launched for each HTTP request that accesses a CGI program. A large number of CGI requests can exhaust a Web server's memory and processing resources.

- A CGI program runs in a separate process. After the CGI program begins running, it cannot interact with the Web server.
- Web server vendors support plug-in APIs that enable CGI applications to be linked directly to the Web server. Using these APIs improves CGI application performance but makes porting CGI applications between Web servers more difficult.

Using servlets in a Web application can solve these performance and portability problems. Servlets have better performance than CGI programs. The servlet programming model is very similar to the CGI programming model, which simplifies replacing CGI applications with servlets. Servlets run inside the Web server process and have no barriers to communication with the server. Each servlet represents a thread within the Web server, which uses server resources more efficiently. Servlets are also portable across platforms and application servers.

**Fewer client prerequisites**

Using servlets moves Java support to the middle tier of an application. This reduces the prerequisites for servlet clients. For instance, servlets can support HTTP clients that do not require a Java Virtual Machine (JVM).

**Support for thin clients**

Moving the back-end access, business logic, and connectivity from a client to a servlet results in a much thinner client that better supports the multitiered architecture of Web applications. For a description of this type of client, see *Getting Started with WebSphere Application Server*.

**Common security model**

Servlets share the same security model as JSP pages and enterprise beans. See "Security services" on page 52 for details.

**Web application security**

Security is a major concern for Web applications. Typically, firewalls are used to control outside access. Servlets use the HTTP protocol, which is supported by firewalls. This provides a single point of entry to the firewall and conceals back-end resources from the client. See "Implementing security in Web applications" on page 69 for details.

**Extend capabilities of applets**

Applets operate under a tight security model that prohibits them from making connections to any server other than the one from which they originated. They can use servlets to make connections to resources on

other servers. For instance, an applet can use a servlet to make a connection to a database that resides on a different server than the one from which the applet originated. This also uses resources more efficiently, since the servlet can be shared among many applets. See *Getting Started with WebSphere Application Server* for more information on applet-based clients.

# Chapter 4. Using enterprise beans

Enterprise beans are the standard component architecture for building distributed applications in the Java programming language. They are server-side components that must reside in a home environment and run in an execution environment. The execution environment must provide run-time services such as transaction support, persistence, and resource management.

This section provides an overview of the WebSphere Application Server Enterprise JavaBeans (EJB) environment. It includes the following topics:



It provides background information necessary for understanding the WebSphere family example application described in "Part 2. Using WebSphere Application Server" on page 107. It does not focus on the practical development issues of creating enterprise beans. For more detailed information on enterprise bean development, see the following:

- *Writing Enterprise Beans in WebSphere*
- The IBM VisualAge for Java product documentation and samples

## EJB environment

The EJB specification provides a detailed description of the services needed to support enterprise beans. It separates the enterprise bean's business logic from the intricacies of persistence, transactions, security, and other middleware-related services.

### Client view

Figure 9 on page 34 shows the client view of the EJB environment. EJB clients interact with enterprise beans through the interfaces defined by the EJB container, the client container, and the services provided by the EJB server. Clients interact with an enterprise bean through its home and remote interfaces. The EJB server environment and containers are discussed in the sections that follow.

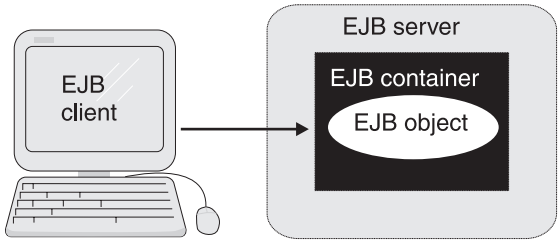


*Figure 9. A client view of the EJB environment*

## EJB server environment

The EJB server manages containers and provides the services required by the EJB Specification. Mandatory services are:

- Java Naming and Directory Interface (JNDI)
- A transaction service compatible with the Object Transaction Service (OTS)
- Security

The EJB server can optionally provide access to a data store through Java Database Connectivity (JDBC). See "Object Services" on page 49 for more detailed information on the services required by the EJB Specification.

## Containers

Enterprise beans are deployed into a *container*. As described in the EJB Specification, a container manages one or more enterprise beans. Deployment adds a concrete implementation of object services.

A container provides access to the required object services through two interfaces:

- The *home interface* exposes methods for creating, removing, and finding an enterprise bean in a container. A *deployment descriptor* is used by deployment tools to give a meaningful name to the home interface. The container registers this name in the name space, which is accessible by clients using JNDI.
- The *remote interface* (also referred to as the *EJBObject interface*) defines the business methods offered by an enterprise bean class.

A client interacts with the interfaces provided by the container. Figure 10 on page 35 shows how a client application interacts with an enterprise bean through the home and remote interfaces provided by a container.

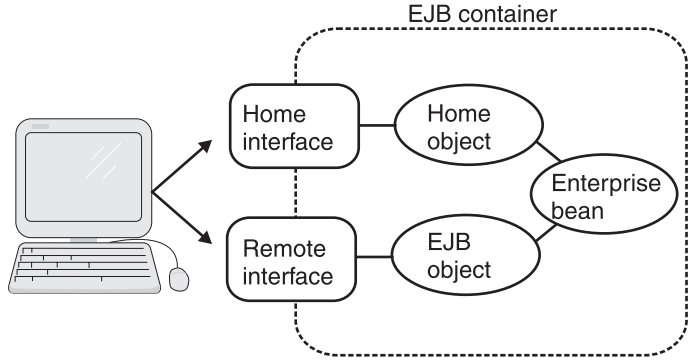


*Figure 10. Client interaction with an enterprise bean*

The client application uses the home interface to access the *home object*. The home object can create, remove and find instances of the enterprise bean class it uses. The client uses the remote interface to access the *EJB object*. The EJB object can invoke the business methods of an enterprise bean instance.

## Enterprise bean application development

This section provides a high-level overview of developing applications with enterprise beans. It discusses the following topics:

- "Enterprise bean types"
- "Using session and entity beans in an application" on page 36
- "Development team roles" on page 37
- "Application development process" on page 38

### Enterprise bean types

The EJB architecture provides two types of enterprise beans for building distributed applications: session beans and entity beans. Each type of bean performs different functions within an application.

- A *session bean* performs work on behalf of an EJB client. The *conversational* state of a session bean instance is not persistent and does not survive a server failure. The life cycle of a session bean is typically the same as the life cycle of the EJB client. For a more detailed description of this type of enterprise bean, see "Session beans" on page 38.
- An *entity bean* represents a permanent entity in an application. They are often used to represent and manipulate persistent data. Persistent data can be stored in a data source such as an object-oriented database or a relational database. It can also be produced by invoking an application or by executing a transaction. The persistent data represented by an entity bean can be managed either by the bean's container (container-managed

persistence) or by the bean itself (bean-managed persistence). For a more detailed description of this type of enterprise bean, see "Entity beans" on page 41.

One of the biggest differences between session beans and entity beans is that session beans do not have a primary key class that uniquely identifies the enterprise bean. Session beans do not require primary keys because they are not unique; they can be created, associated with a client, and removed as needed. In contrast, entity beans represent permanent data that can be uniquely identified.

## Using session and entity beans in an application

Entity beans represent persistent data and need to be shared among clients. They cannot maintain information related to a specific client. Session beans cannot access data directly, but can maintain information about their clients. Most applications therefore need to use both types of beans.

Figure 11 shows how session and entity beans can work together in an application. In this example, a client application logs onto an EJB server. It can display a list of employees and change employee data.

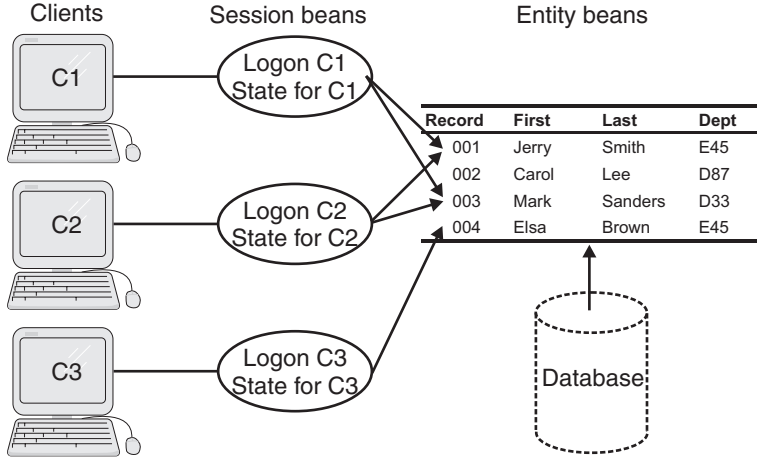


*Figure 11. Building an application with session beans and entity beans*

The EJB client needs to connect to the server and store connection information for later use. These tasks can be performed by using session beans.

To display the list of employees, the EJB client must retrieve employee data from a database. This task can be performed by using entity beans. Each entity bean represents a row in the database. A session bean that represents a login session can access employee data through the entity beans.

Changing employee information can be done through a transaction. Session beans can manage the transactional context of the entity beans that represent the data stored in the database.

## Development team roles

Separating the development of business logic from other tasks requires the members of the application development team to assume different roles. The EJB specification defines six different roles for the application and deployment process. Software developers are not limited to a single role; they can take on as many of these roles as a project requires. Each role is described as follows:

**Enterprise bean provider**
Providers understand the application's business logic and know how to implement it in the Java programming language. Providers also understand the interfaces and semantics of enterprise beans. They decide whether the enterprise bean is persistent or not. If the enterprise bean is persistent, they must identify which fields have a persistent state and decide whether bean-managed persistence or container-managed persistence is used. In addition, they specify how an enterprise bean behaves when it is involved within the scope of a transaction. Providers are responsible for packaging the enterprise beans and all associated files into Java Archive (JAR) files, which are then given to a deployer.

**EJB container provider**
Container providers create enterprise bean containers, which provide client interfaces, persistence, scalability, security, and transactional support for enterprise beans. They link the business logic to the underlying services.

**EJB server provider**
Server providers bring operating system and middleware services to the container. In most cases, the same vendor provides the server and container because there are as yet no specifications to define the interface between a container and a server.

**Deployer**
Deployers install the enterprise bean classes on the EJB server. They understand enterprise beans and the EJB server environment and can configure enterprise bean requirements by using EJB server tools. They also make enterprise beans accessible through JNDI.

**Application assembler**
Application assemblers write applications that use enterprise beans, such as applets, servlets, and native CORBA applications. They can also build new enterprise beans from existing enterprise beans.

**System administrator**

System administrators ensure that the system is working properly by using the monitoring and management tools furnished by the server and container providers.

### Application development process

Developing an application with enterprise beans is a five-step process:

1. The provider codes the application's business logic in a set of enterprise beans, then packages the beans and their deployment descriptor in a JAR file.
2. The container provider creates the container used by the enterprise beans.
3. The server provider sets up the EJB server to provide the essential services required by the application.
4. The beans are deployed on the EJB server. The deployer generates additional classes used internally by the container. Usually, this is done through tools that the container software provides.
5. The application assembler builds the EJB client.

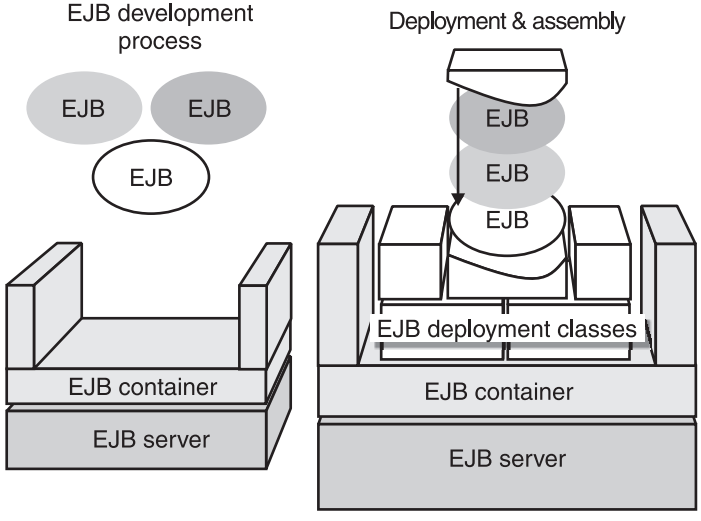


*Figure 12. Enterprise bean application development process*

## Session beans

Session beans are transient objects that exist for the duration of a single user session. They encapsulate data and methods associated with a user session, task, or ephemeral object.

A session bean executes a unit of work on behalf of an EJB client. Session beans are not shared among clients and can therefore maintain client-specific session information. Session bean support is mandatory for containers that comply with the EJB Specification.

## Stateless and stateful session beans

The data in a session bean instance is ephemeral; if it is lost, no real harm is done. The design of a session bean determines whether its data is shorter lived or longer lived:

- If a session bean needs to maintain specific data across methods, it is referred to as a *stateful* session bean. When a session bean maintains data across methods, it is said to have a conversational state. After a particular client begins using an instance of a stateful session bean, the client must continue to use that instance as long as the specific state of that instance is required.
- If a session bean does not need to maintain specific data across methods, it is referred to as a *stateless* session bean. For stateless session beans, a client can use any instance to invoke any of the session bean's methods.

A container can manage stateful session bean instances by removing an instance from memory and saving it to permanent storage. This is called *passivation.* When the session bean is invoked again, the container creates a new instance and initializes it with the data saved during passivation. This is called *activation.*

Containers do not use passivation with stateless session beans. Stateless session beans can be destroyed if memory resources are low. Because all instances of stateless session beans are the same, the container can use any available instance to satisfy a client request.

Session bean data is not permanently stored. The session bean class does not contain methods for storing and retrieving data from a persistent data source. However, a session bean can optionally access and update data in a transaction-safe mode.

## Session bean components

Every session bean must contain a bean class, a home interface, and a remote interface.

### Bean class

The bean class encapsulates the data associated with the enterprise bean and contains the business methods that access this data. It also contains the methods used by the container to manage the life cycle of an enterprise bean instance. Clients such as other enterprise beans and user applications never directly access objects of this class. Instead, they use the container-generated classes associated with the home and remote interfaces to manipulate the enterprise bean.

The bean class can contain business methods, but is not required to. These methods must be public. Business method names cannot conflict with other names used in the EJB architecture. (For instance, if you need to declare a method for performing a calculation, do not name it ejbCalculate. A better choice for the method name is calculate.)

Each business method parameter must be a permitted Java RMI type. The throws clause can include specific application exceptions.

Implementing the enterprise bean's remote interface in the bean class is not recommended.

#### Home interface

The home interface contains methods used by the client to create and remove instances of the enterprise bean. This interface is implemented by the container during enterprise bean deployment in a class known generically as the EJB home class.

A session bean's home interface defines the methods used by clients to create and remove instances of the enterprise bean and obtain metadata about an instance. The home interface is defined by the developer and implemented in the EJB home class created by the container during deployment. The container makes the home interface accessible to clients through the Java Naming and Directory Interface (JNDI).

Home interface methods must follow the Java RMI rules.

The home interface must implement one or more create methods. The return type for a create method is the enterprise bean remote interface type.

#### Remote interface

A session bean's remote interface provides access to the business methods available in the enterprise bean class. It also provides methods to remove an enterprise bean instance and to obtain the enterprise bean's home interface and handle. The remote interface is defined by the developer and implemented in the EJB object class created by the container during deployment.

After the client has used the home interface to gain access to an enterprise bean, it uses the remote interface to invoke the business methods defined in the bean class. This interface is implemented by the container during enterprise bean deployment in a class known generically as the EJB object class.

The remote interface contains the methods used by the EJB client developer. Each method must have a matching method in the bean class with exactly the

same parameters. Remote interface methods must follow the Java RMI rules. The return types must be valid Java RMI types, and the methods must include the java.rmi.RemoteException in the throws clause.

### Session bean life cycle

During its life cycle, a session bean instance goes through the following states:

1. *Creation state*. A session bean's life cycle begins when a client invokes a create method defined in the bean's home interface. The container sets the session context and creates a new session bean instance.
2. *Ready state*. After a session bean instance is created, clients can invoke the business methods defined in the bean's remote interface.
3. *Pooled state*. When a stateful session bean instance is no longer needed, the container passivates the instance. If a client invokes a method on a passivated session bean instance, the container activates the instance and returns it to the ready state.

   Stateless session bean instances are neither passivated nor activated. These instances exist in a ready state at all times until their removal.
4. *Removal state*. A session bean's life cycle ends when a client or the container invokes a remove method defined in the bean's home interface or remote interface.

For a more detailed description of the life cycle of a session bean, see *Writing Enterprise Beans in WebSphere*.

## Entity beans

An entity bean represents persistent data that can be stored in a database, produced by a transaction, or produced by an application.

### Bean-managed persistence (BMP)

In bean-managed persistence (BMP), the entity bean manages the storage and retrieval of persistent data. The entity bean developer must encode the bean's business logic and explicitly make database calls or other type of access to permanent storage. The developer must also save and restore the state of the enterprise bean when the container uses life cycle methods such as ejbFind, ejbLoad, and ejbStore to call the bean.

The advantage of using an entity bean with BMP is that the bean can be tailored to meet the data-handling requirements of an individual EJB client. For instance, it can be written to optimize access to a specific type of data storage. However, this customization also makes it more difficult to reuse and maintain the entity bean.

Entity beans with BMP are recommended for data stores that are not supported by EJB containers. For instance, storing persistent data in a file is

supported by only a few types of containers. (Most containers store persistent data in databases.) Using entity beans with BMP enables you to save data in a file if the container does not support this type of storage.

## Container-managed persistence (CMP)

In container-managed persistence (CMP), the container handles the interactions between the entity bean and the persistent data source. Database connections for entity beans with CMP are not explicitly coded by the entity bean developer. Instead, the developer simply specifies which fields are persistent; the container handles the details of making the database calls. The container maps entity bean fields to a database or an existing application. In addition, it can efficiently manage database access by using a shared pool of connections and caching data. VisualAge for Java also provides ways to associate fields in entity beans with CMP to database columns.

Every entity bean with CMP has the following components:

- Bean class
- Home interface
- Remote interface
- Primary key class
- Finder methods

These components are described as follows.

### Bean class

The bean class defines and implements the business methods used to access and manipulate the data associated with the entity bean. It also defines and implements the methods used to create instances of the entity bean, and implements the methods used by the container to inform the instances of the entity bean of significant events in the instance's life cycle (callback methods). The bean class is called *name*Bean, where *name* is the name assigned to the entity bean.

### Home interface

The home interface defines the methods that are used by EJB clients to perform the following tasks:

- Create new instances of the bean.
- Find instances of the bean.
- Remove instances of the bean.
- Obtain information about an instance.

The home interface is implemented by the EJB home class generated by the container's deployment tool. The container registers the home interface in the name space accessible to EJB clients that use the Java Native Directory

Interface (JNDI) application programming interface (API). The home interface is called *name*Home, where *name* is the name assigned to the enterprise bean.

**Note:** All home interfaces must be associated with an enterprise bean implementation. Abstract home interfaces, or home interfaces that are not associated with a bean, are not supported.

### Remote interface

The remote interface provides access to the business methods available in the bean class. It is called *name*, where *name* is the name assigned to the enterprise bean.

The container implements methods for returning the home interface, the handle, and the primary key of an enterprise bean instance; comparing interface instances; and removing enterprise bean instances.

### Primary key class

Every entity bean has a unique identity within a container. The bean's identity is defined by using a combination of the object's home interface name and its *primary key*. The client uses the primary key to create or find an instance of an entity bean. The primary key is assigned when the instance of the entity bean is created. Two enterprise bean instances that have the same identity are considered to be identical.

Simple primary keys that consist of a single field of a primitive Java data type (such as integer, long, or string) can be assigned at deployment. Composite primary keys that are composed of multiple fields or more complex Java data types must be encapsulated in a *primary key class*. This class must be public and serializable. Its instance variables must be public and the variable names must match a subset of the variable names defined in the bean class.

By convention, the primary key class is called *name*Key, where *name* is the name assigned to the enterprise bean.

### Finder methods

The finder methods retrieve instances from the database by using search criteria other than the primary key value. (For example, these methods can be used to perform a search by a customer's account number.) You must define a unique query string for each finder method other than the findByPrimaryKey method.

You can create a finder helper interface for the EJB server (Advanced Application Server) environment, if required. The container uses it to generate the necessary code for querying the database by using finder methods.

### Entity bean life cycle

After an entity bean is deployed into a container, clients can create and use instances of that bean as required. During its life cycle, an entity bean instance goes through the following states:

1. *Creation state* — An entity bean instance's life cycle begins when the container creates that instance and sets its session context.
2. *Pooled state* — After an entity bean instance is created, it is placed in a pool of available instances of the specified entity bean class. Every instance of that bean class in the pool is identical. While an instance is in this pooled state, the container can use it to invoke any of the bean's finder methods.
3. *Ready state* — When a client requests a specific entity bean instance, the container picks an instance from the pool and associates it with the EJB object initialized by the client. Two events cause an entity bean instance to be moved to the ready state:
   - When a client invokes the create method in the Bean's home interface to create a new and unique entity of the entity bean class (and a new record in the data source).
   - When a client invokes a finder method to manipulate an existing instance of the entity bean class, which is associated with an existing record in the data source.

   When an enterprise bean instance is in the ready state, the container can synchronize the data in the instance with the corresponding data in the data source. In addition, the client can invoke the bean instance's business methods.

   Entity bean instances in the ready state are moved to the pooled state when they are no longer required.
4. *Removal state* - An entity bean instance's life cycle ends when the container removes its context while it is in the pooled state.

**Note:** Removing an entity bean instance does not remove data that is stored in the data source.

For a more detailed description of the life cycle of an entity bean, see *Writing Enterprise Beans in WebSphere*.

## Extensions to the EJB specification

WebSphere Application Server has extended the EJB specification to supply additional functionality for application designers. The extensions to the EJB specification include access beans, inheritance, and association.

## Access beans

An access bean is a JavaBeans component that serves as a wrapper around one or more enterprise beans. Access beans help to bridge the gap between developers who concentrate on presentation and developers who concentrate on business logic. They simplify the use of enterprise beans in a client application by hiding the enterprise bean's home and remote interfaces from EJB clients.

Access beans enable the development of EJB clients that use enterprise beans in the same way as they use local JavaBeans components. The EJB client interacts only with the access bean wrapper and does not directly use the enterprise bean. This eliminates the need to explicitly make remote calls to a naming service and an enterprise bean's home and remote interfaces. Access beans can also improve performance by maintaining a local copy of an enterprise bean's attributes. They are written in pure Java code and do not affect the portability of applications.

Access beans are primarily intended to support servlet and JavaServer Pages (JSP) programs. However, they can be used by any application that accesses a server-side enterprise bean.

VisualAge for Java supports three types of access beans:

- **Wrapped bean**—Acts as a wrapper for a session bean instance.
- **CopyHelper for an entity bean**—Acts as a wrapper for an entity bean instance and maintains a local copy of the attributes of that instance. An EJB client can access the copy of the attributes instead of retrieving them directly from the remote entity bean.
- **Rowset for multiple entity bean instances**—Acts as a wrapper for one or more entity bean instances and stores local copies of their attributes. It enables an EJB client to use multiple entity beans without having to initialize and contact them individually.

Access beans are available only through the Advanced Application Server run time and can be created only in VisualAge for Java. When Visual Age for Java creates a client JAR file, it exports the access beans associated with the selected enterprise beans. Each access bean class is identified as a Java bean in the manifest file.

For more detailed instructions on developing access beans, consult the VisualAge for Java documentation.

## Inheritance

The WebSphere Application Server enterprise bean architecture recognizes inheritance in enterprise beans to support polymorphism and enable enterprise beans to be reused more easily. Inheritance allows an enterprise

bean to be used as the basis for creating new beans that inherit its interfaces. Inheritance can be implemented only in VisualAge for Java.

This section provides background information on how inheritance fits into the WebSphere enterprise bean model. It is not intended to be a guide for using inheritance. For more detailed information, see the Visual Age for Java documentation.

### How inheritance affects an enterprise bean

Enterprise beans do not directly inherit the interfaces of their parents. Instead, the parent enterprise bean interfaces serve as templates for creating the child interfaces as shown in Table 2. In this example, the parent enterprise bean is named Parent and the child enterprise bean is named Child. Note that the parent's primary key class is the only interface that retains its name in the child.

*Table 2. Parent and child interfaces*

| Interface | Name of Parent bean interface | Name of Child bean interface |
|---|---|---|
| Remote interface | Parent | Child |
| Home interface | ParentHome | ChildHome |
| Bean class | ParentBean | ChildBean |
| Primary key class | ParentKey | ParentKey |

**Note:** Enterprise beans that do not use inheritance or are at the root of an inheritance hierarchy do not have any additional implementation requirements.

Inheritance affects the following components of an enterprise bean:

**Remote interface**

The remote interface of the child enterprise bean extends the remote interface of the parent bean. This ensures that an instance of the child can be used where an instance of the parent is expected.

**Home interface**

The home interface of the child enterprise bean must not extend the home interface of the parent bean. The following methods are implemented differently in parent and child enterprise beans:

- A create method on the parent's home interface creates only an instance of the parent enterprise bean.
- A create method on the child's home interface creates only an instance of the child enterprise bean (which is treated like the parent).

- A remove method on the parent's home interface removes only parent enterprise bean instances.
- A remove method on the child's home interface removes both the child and parent enterprise bean instances.
- A custom finder method on the parent's home interface returns an enumeration that contains parent instances.
- A custom finder method on the child's home interface returns an enumeration that contains both parent and child instances.

**Bean class**

Bean implementations that are the root of an inheritance hierarchy define a discriminator field that determines the type of instance of the child entity bean. A remote interface method for determining the type of instance must also be defined for the discriminator.

The child bean's implementation class must extend from a proxy class that contains all of the methods defined in the remote interface of the parent bean. An invocation of any of these methods is redirected to an instance of the parent bean.

**Primary Key Class**

Entity beans must define a Java key class for their primary keys. All entity beans in an inheritance hierarchy must use the same key class for their primary keys.

### Inheritance in entity beans

Inheritance provides the following additional functions for entity beans with CMP:

- Identity relationships can be specified between entity beans.
- Primary key searches return the target class and its subclasses.
- Relationship finders return results that contain instances of the target class and its subclasses.

For instance, suppose you have created an entity bean, Account, that represents a bank account. The Account bean can serve as the parent of a group of related entity beans, such as SavingsAccount, CheckingAccount and CorporateAccount enterprise beans. The children of the Account bean are identified as Account beans. A search for all Account beans returns the Account bean and all of the entity beans that are derived from it.

Entity beans with BMP that use inheritance follow the same programming model as entity beans with CMP, with one exception. BMP finder methods can return only enumerations that contain the type of bean managed by the home on which the finder method is defined.

### Inheritance in session beans

Session beans that use inheritance follow the same programming model as entity beans that use inheritance. In addition, child session beans must manage their state in the same way as their parents. All session beans that inherit from a stateless session bean must be stateless; all session beans that inherit from a stateful session bean must be stateful.

### Inheritance support in VisualAge for Java

The EJB tools in VisualAge for Java can be used to define inheritance between enterprise beans. They enable a hierarchy of enterprise beans to be mapped to database tables. They can also generate persistor and finder code to extract the instances of a child bean according to the selected database mapping schema. (This means that the results of a find operation on a parent home may return instances of its child beans.)

The relationships between parent and child enterprise bean classes are defined in the metadata for enterprise beans that are identified as child classes. This is done by creating a generalization object that relates the parent and child class bean models.

VisualAge for Java recognizes inheritance relationships between created and imported enterprise beans. The deployment descriptors in a JAR file contain parent class information. Entity beans from other sources derive this information from the interfaces and classes specified by the deployment descriptors.

For inheritance hierarchies that are mapped to persistent storage, the root class defines the create table string for the whole hierarchy. This defines either a single table into which all subclasses map, or the root and leaf tables into which the entire hierarchy maps. Since any class could be asked to create the tables, the child classes returns the create table string of their parent class.

A child class must have a minimum persistence map provided by the map browser. This is required to generate the queries for the persistor for the new child class. At a minimum, the map must specify which inheritance strategy is to be used. It can also optionally supply a value for the discriminator field.

## Association

The WebSphere Application Server enterprise bean architecture provides association as an extension of the EJB specification. Association defines relationships between entity beans with CMP in order to improve database searches. It is implemented only in VisualAge for Java.

VisualAge for Java provides mapping and deployment support for association. Its implementation of association can be deployed only to the WebSphere Application Server Advanced Edition EJB server. WebSphere Application Server Enterprise Edition users can modify enterprise beans that have

association relationships, then deploy them to an enterprise server by using the VisualAge Component Development Toolkit's Object Builder.

This section describes how association fits into the WebSphere enterprise bean model. For more detailed information, see the Visual Age for Java documentation.

Association applies only to entity beans with CMP. The EJB tools in VisualAge for Java allow users to define associations between beans according to the Uniform Modeling Language (UML) definition of associations. These tools can be used to create, edit, and delete associations and maintain consistency between the participating entity beans. Both single-valued forward relationships (one-to-one relationships) and many-valued forward relationships (one-to-many relationships) can be defined.

For each association end, only navigability and multiplicity are collected. The map browser enables associations to be mapped to key fields in other entity beans. The deployment tool generates finders for backward association mapping.

Key fields contain data to support single relationships between entity beans. Initially, relationships can only refer to entity beans within the same group (which is defined in VisualAge for Java). Subclasses inherit the key fields and accessors for single-valued forward relationships between entity beans. However, the finders for many-valued forward relationships are generated only on the home of the target class, not its subclasses.

When a single-valued association role is defined for a class, an appropriate public key field is added to the bean class. A validation of these definitions occurs when deployment code is generated. If the appropriate constructs are not in the bean to support the defined relationships, code generation stops.

## Object Services

WebSphere Application Server supports the following object services for the EJB environment:

- "Naming and directory services" on page 50
- "Security services" on page 52
- "Persistence services" on page 55
- "Transaction services" on page 59

Both Java and Common Object Resource Broker (CORBA) object services are supported. The Java object services apply to the Advanced Application Server implementation of Java standard interfaces such as the Java Naming and Directory Interface (JNDI), the Java Transaction Service (JTS), and Java

Database Connectivity (JDBC). The CORBA object services apply to the Enterprise Application Server implementation. Interoperability issues are described in "Part 2. Using WebSphere Application Server" on page 107. The object services apply to both the Advanced Application Server and the Enterprise Application Server.

## Naming and directory services

A *naming service* enables enterprise beans to be found by their names. Many naming services are extended with a *directory service*. A directory service also enables objects to have attributes. In addition to looking up an enterprise bean, it enables you to find an enterprise bean's attributes or search for enterprise beans by their attributes.

WebSphere Application Server supports three naming and directory services:

- Java Naming and Directory Interface (JNDI) furnishes directory and naming functionality for Java applications. It provides a common way to access directories that is independent of a specific directory service implementation. The JNDI Service Provider Interface (SPI) enables different directories to be accessible from applications that use JNDI.
- The Common Object Request Broker Architecture (CORBA) CosNaming Naming Service enables clients of Object Request Broker (ORB) based systems to locate remote objects. The Advanced Application Server supports a persistent Java implementation of the CosNaming Naming Service that can be used in situations where basic naming support is needed.

### Naming service components

The WebSphere Application Server naming service has three components:

- Location service daemon
- Persistent name server
- EJB server

Figure 13 on page 51 provides a system-level view of the naming service components.

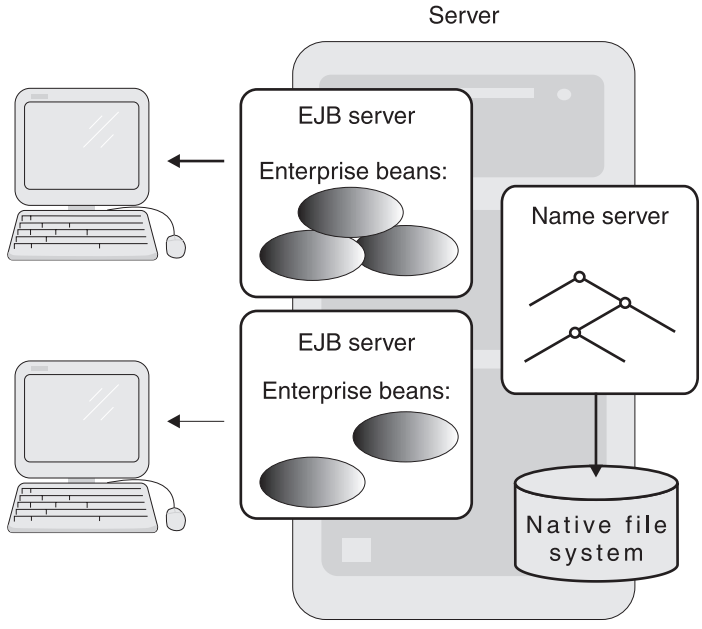


*Figure 13. WebSphere Application Server object services system view*

**Location service daemon:** The location service daemon keeps track of the name spaces in use across the network. The tasks it performs depend on the specific naming service used by the EJB client.

- An EJB client that uses JNDI requests an initial session context. It then connects implicitly to the location server daemon, which waits for object requests.
- An EJB client that uses the CosNaming naming service issues an object request. The request is forwarded to the location server daemon, which retrieves the host on which the object implementation is running. A new interoperable object request (IOR) pointing to the host running the enterprise bean is returned to the client ORB.

**Persistent name server:** An EJB server registers enterprise beans with a persistent name server. Each persistent name server maintains an independent name space, regardless of how many are running in the network.

A persistent name server uses the native file system to maintain the EJB name space in persistent storage. When the system is started, the persistent name server contacts the location server daemon and loads a copy of the name space in memory. The cached naming information is used to perform operations that do not affect the name space data in permanent storage, such

as listing objects. Operations that permanently affect name space data, such as creating new object names, are performed transactionally on the copies in memory and on disk.

**EJB server:** Enterprise beans are deployed in an EJB server. When an EJB server is started, it automatically registers all EJB home objects in the name space. To perform this operation, an EJB server is given the location server daemon's hostname and port number at startup. It then retrieves the initial context of the name space root and binds EJB home objects with their JNDI names into the name space.

## Security services

WebSphere Application Server provides a unified security model. A single policy can govern the security of Web pages, servlets, and enterprise beans. The following security topics are discussed in this section:

- "Security server"
- "Security collaborator"
- "Security plugin" on page 53
- "Authentication services" on page 53
- "Authorization services" on page 53
- "Delegation policy" on page 54
- "Using security services" on page 54

The security application comprises the security server, the security collaborator, and the security plug-in.

### Security server

The security server gives centralized control over security policies and security services. It provides the following for Web servers and EJB servers:

- Authentication, authorization, and delegation policies
- Authentication and authorization services, including token services when the lightweight third-party authentication (LTPA) model is used

The security server is coupled with the System Management Facility (SMF). All instances of the security server obtain their security policies through SMF.

### Security collaborator

The security collaborator works with the security server to perform the following services for every remote method invocation of an enterprise bean:

- Check authorization
- Log security trace information
- Enforce the delegation policy

It supports user registries based on the Lightweight Directory Access Protocol (LDAP) and on operating systems such as Windows NT and UNIX. It also supports the single user registry of the Advanced Application Server.

### Security plugin

The security plug-in resides on the Web server and protects access to resources like HTML pages, servlets and JSP pages. It consults the security server for authentication and authorization services.

### Authentication services

The underlying assumption of an authentication scheme is that the client and the server do not trust each other. In WebSphere Application Server, authentication is based on validating credentials (such as a user ID and password), certificates, or tokens. Credentials are verified against the user registry. A certificate validation list is used when authentication is based on a client certificate presented by the user over a mutual Secure Sockets Layer (SSL) connection.

WebSphere Application Server supports a three-party authentication scheme. The client principal and the server principal are authenticated to a mutually trusted third party—for example, the authentication token server. This allows the user registry to be centrally administered.

A principal has a number of attributes that can be used to control its access to system resources. These attributes can be administered through an individual user ID or through groups. Where possible, administer access control on groups. Administering access control to individual users can be cumbersome if many attributes need to be changed at once.

The policy for performing authentication between a user and a Web server or EJB server can be specified by the following mechanisms:

- A *challenge mechanism* specifies how a server receives authentication data from a user. It can be based on a user ID and password or certificates. Using a challenge mechanism is optional.
- An *authentication mechanism* validates the authentication data against a user registry. Authentication mechanisms can include network user registry authentication, LTPA, and operating system-based authentication.

A set of *constraints* can be specified as a part of the authentication policy. This can include using a secure channel (for instance, where an SSL connection is required) or limiting access to a group of trusted clients.

### Authorization services

WebSphere Application Server uses a capability-based model for security. Individual resources are collected into applications, and methods are collected into method groups. Each user has a set of (application-method group) pairs

that identify the methods in an application that the user is permitted to execute. Each (application-method group) pair is called a permission. WebSphere Application Server administrators manage permissions. When a user attempts to perform an operation, the security run time determines the permissions that grant access.

The authorization policy of an enterprise bean can be managed in one of the following ways:

- The authorization policies of an EJB application apply to the Home objects contained within the application and the enterprise beans contained within the Home objects. This prevents authorization policies to be applied to a Home object instance independently of the enterprise bean instances, except when the Home object methods are assigned to a different method group than the enterprise bean methods.
- The authorization policies of the enterprise beans do not apply to the Home object instance. This allows authorization policies to be defined independently.

### Delegation policy

In general, a method is executed under the principal of the process that issued the operation. However, methods sometimes need to run under a different principal (for instance, to use resources that the client does not have permission to access). A method can be executed with the identity of the client, the identity of the system server or server group, or an identity that is specified from the enterprise user registry. This identity does not have to map to an operating system identity if the enterprise user registry is not an operating system registry.

The default delegation policy for the methods of an enterprise bean comes from its deployment descriptor. It can specify delegation for a method or for a bean. Administrators can override the delegation policy by using the WebSphere Administrative Console.

### Using security services

When a client executes a method on an enterprise bean or its Home object, the EJB server must determine whether the principal (client) is permitted to do so.

Figure 14 on page 55 shows an example of the principal Teller attempting to access a getBalance method on the Account enterprise bean.

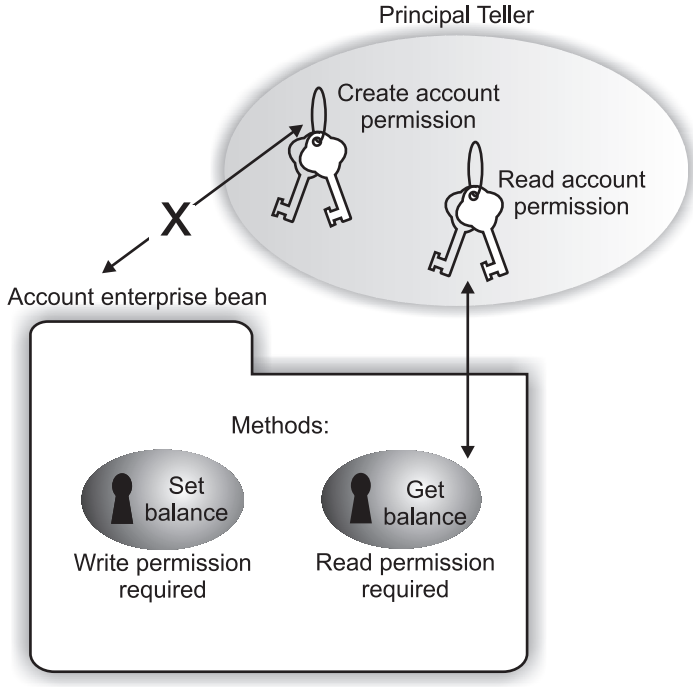


*Figure 14. Permission-based protection of an EJB method*

The following steps are taken to determine if access is permitted:

1. The EJB server uses the delegation policy to determine the identity to be checked—in this case, the identity of the client.
2. The EJB server identifies the principal of the client. If the principal cannot be determined, the request is rejected. In this case, Teller is identified as the principal.
3. The set of permissions corresponding to the method is resolved. In this example, the permission required to use the getBalance method is the AccountRead permission.
4. If Teller has the permission AccountRead, the method is invoked.

## Persistence services

This section describes the underlying components and interfaces of the WebSphere Application Server persistence service. They include:


- "Connector beans" on page 56
- "Adaptor beans" on page 56
- "Persistors" on page 57
- "Common Connection Framework (CCF)" on page 57
- "Implementing persistence" on page 59

### Connector beans

*Connector beans* are Java beans that simplify access to applications, components and databases. They can be used to implement persistence for servlets and JSP pages. There are two types of connector beans:

- **Data connector beans**—Support access to relational databases.
- **Procedural connector beans**—Encapsulates a single call-return interaction with a legacy procedural system.

WebSphere Application Server, VisualAge for Java and WebSphere Studio support and generate the following implementations of connector beans:

- JDBC
- SQLJ
- Customer Information Control System (CICS)
- ECI, EPI and EXCI
- MQSeries
- Access to enterprise beans
- Internet Management Specification (IMS) transactions

Servlets use connector beans to access relational data sources through JDBC and SQLJ. They also use connector beans to call interfaces to external applications such as CICS transactions. Servlets may also implicitly use persistence services by calling enterprise beans. For more information on servlets, see "Chapter 3. Using servlets" on page 25.

JSP pages access connector beans by using the UseBean tag. For more information on servlets and JSP pages, see "Chapter 2. Using JavaServer Pages" on page 17.

### Adaptor beans

*Adaptor beans* are JavaBeans that support relational database management schemes. They can be used to implement persistence for servlets and JSP pages. Adaptor beans directly use persistence interfaces such as Java Database Connectivity (JDBC) and the IBM Common Connector Framework (CCF).

WebSphere Studio and VisualAge for Java provide tools that generate adaptor beans. Their implementation of adaptor beans simplifies the use of JDBC, SQL, RowSets, and other database query languages. It also interacts with the WebSphere database connection manager.

Adaptor beans can be accessed directly from servlets or by using a UseBean tag in JSP pages.

### Persistors

The WebSphere Application Server enterprise bean programming model recommends direct use of JDBC and other persistence APIs within a helper class. This helper class is called a *persistor*. The persistor class is generated by EJB development tools such as VisualAge for Java.

Persistors are implemented by enterprise beans that directly maintain persistent data. Entity beans with BMP implement functions that create, retrieve and modify persistent data by calling Java persistence services (such as JDBC) within a persistor. Entity beans with CMP do not need to implement persistors, since they rely on the helper classes produced by their container and deployment tools to maintain persistent state data. Although session beans are not persistent, they can maintain persistent state data by using entity beans or by calling other Java persistence services within a persistor.

For more information on enterprise beans, see Enterprise JavaBeans.

### Common Connection Framework (CCF)

CCF provides a framework for developing *connectors*. A connector is a software package or library that implements the client side of a client-server protocol. Examples of connectors include JECI for CICS, JBAPI for SAP, and JDBC.

All of these packages make connections to a server system. Each connection defines a server instance to which requests are sent (for example, a CICS region or a database table). In addition, the connector and server associate the connection with user information, such as security delegation and transaction participation.

Figure 15 on page 58 shows the CCF architecture divided into two interface groups: the CCF client interfaces and the CCF infrastructure interfaces.

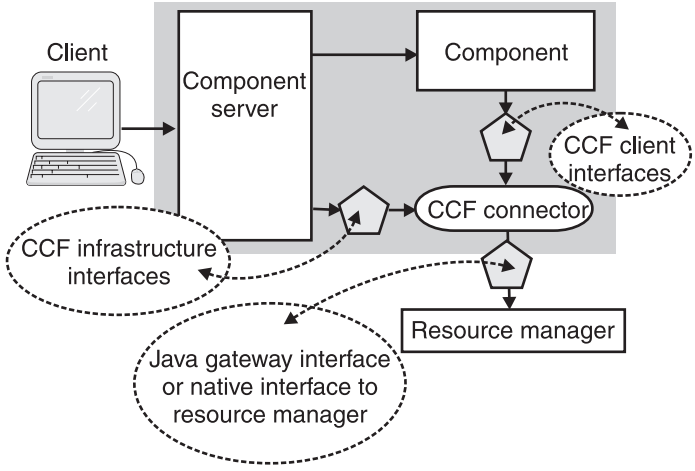


*Figure 15. CCF architecture*

A connector uses the CCF client interfaces to interact with a resource manager.

The CCF infrastructure interfaces contain two interfaces:

- **Quality of service (QOS)**—This interface is implemented by a component server. QOS adapts the CCF connector quality of service view to the actual implementation of the service in the component server. QOS can be used for tasks such as retrieving security information and enlisting with the current transaction.
- **State management (SM)**—This interface is implemented by a CCF connector. SM is used by the component server's QOS implementation to control CCF connector states. The states controlled by SM include the physical connection and the transactional state of the connected resource manager (with respect to the current transaction).

Internally, a CCF connector uses proprietary connectors that can be accessed through a Java gateway interface or a native interface.

The Enterprise Access Builder (EAB) tool in VisualAge for Java simplifies the use of CCF connectors by defining two Java beans:

- **Command beans** (or commands) encapsulate single interactions with an application system, eliminating the need to manually script calls to the CCF connector. CCF's client interface interacts with EAB commands, allowing you to use any CCF-based connector with an EAB command.
- **Navigator beans** (or navigators) implement sequences of interactions with an application system. They can be combined with command beans and other navigator beans to create a complex series of interactions.

For more detailed information on EAB, command beans, and navigator beans, see the VisualAge for Java documentation.

### Implementing persistence

The WebSphere Application Server programming model supports the use of general Java classes. A developer can use any class library for implementing persistence within a servlet, JavaBean, session bean or entity bean with BMP. The following guidelines for implementing persistence simplify reusing and moving components.

- Use access beans that encapsulate CCF. Do not use CCF directly. See "Access beans" on page 45 for more information on using this type of bean.
- JDBC calls can be encapsulated in a predictor for session and entity beans with BMP. They can be encapsulated in JavaBeans components for servlets and JSP pages.
- Session beans and entity beans with BMP can access legacy systems through a predictor that delegates legacy system connections to an access bean.

## Transaction services

Transaction support is an essential component of the EJB architecture. WebSphere Application Server provides transaction processing services for enterprise beans. This section discusses the following transaction-related topics:



### Managing transactions for enterprise beans

The EJB architecture provides two ways to manage transaction processing for enterprise beans:

- In *bean-managed transactions*, an enterprise bean controls transactions.
- In *container-managed transactions*, a container controls transactions for its enterprise beans.

Session beans can use either container-managed transactions or bean-managed transactions. Entity beans must use container-managed transactions. Transaction processing is handled by the EJB server. Server providers can use transaction services such as two-phase commit, transaction context propagation, and distributed two-phase commit. Nested transactions are not supported.

EJB clients can also manage transactions; for more detailed information on managing transactions in the EJB environment, see *Writing Enterprise Beans in WebSphere*.

**Bean-managed transactions:** In some situations, a session bean must participate directly in a transaction. To indicate that a session bean is an active participant in a transaction, set the transaction attribute in its deployment descriptor to TX_BEAN_MANAGED. The session bean developer must explicitly demarcate transactions by using the javax.transaction.UserTransaction interface. (This interface can also be used for client-managed transactions.) You must also set the transaction isolation level in a session bean's deployment descriptor.

**Container-managed transactions:** The EJB API does not require enterprise bean and EJB application developers to write special code to use transactions. Instead, a container can manage transactions, freeing the enterprise bean and EJB application developers to concentrate on the business logic of their applications.

When an EJB client invokes a method on an enterprise bean, the container intercepts the method invocation to manage its transactions. The way in which the container controls transaction demarcation is specified by the transaction attribute of either the method or the enterprise bean.

To enable container-managed transactions, you must set the transaction attribute of the enterprise bean's deployment descriptor to any value other than TX_BEAN_MANAGED. The value of the transaction attribute depends on the requirements of your application. You must also set the transaction isolation level; see "The transaction isolation level attribute" on page 61 for details.

### The transaction attribute

The transaction attribute defines the transactional manner in which the container invokes enterprise bean methods. This attribute can be set for the bean as a whole and for individual methods in a bean. Values for this attribute are as follows:

**TX_BEAN_MANAGED**
: Notifies the container that the bean can call methods to explicitly manage transaction boundaries. (See "Bean-managed transactions" for more information.)

**TX_MANDATORY**
: The container always invokes the bean method within the transaction context associated with the client.

**TX_NOT_SUPPORTED**
: The container invokes bean methods without a transaction context.

**TX_REQUIRES_NEW**
The container always invokes the bean method within a new transaction context, regardless of whether the method is invoked within an existing transaction context.

**TX_REQUIRED**
The container invokes the bean method within a transaction context. If a method is invoked outside a transaction context, the container create a new transaction context and invokes the bean method from within that context.

**TX_SUPPORTS**
If the client invokes the bean method within a transaction, the container invokes the bean method within a transaction context. If the client invokes the bean method without a transaction context, the container invokes the bean method without a transaction context.

### The transaction isolation level attribute

The transaction isolation level determines how isolated one transaction is from another. This isolation is for read purposes only. The transaction isolation level can be set for the bean as a whole. The first method call within a transaction uses the bean's isolation level. Different isolation levels on subsequent calls are ignored.

The container uses the transaction isolation level attribute as follows:

- **Session beans and entity beans with bean-managed persistence (BMP)**—For each database connection used by the bean, the container sets the transaction isolation level at the start of each transaction.
- **Entity beans with container-managed persistence (CMP)**—The container generates database accesses that achieve the specified isolation level.

### Locking

Write locks cannot be defined for enterprise beans. As a work around, either define methods as read only or acquire write locks on database rows for non-read-only methods.

### The OTS and EJB transaction models

The EJB transaction model is similar to that of the Object Transaction Service (OTS). The key components of OTS can be mapped almost directly to the EJB transaction service. Understanding how these transaction models work helps to understand how transactions work in the EJB environment.

The EJB transaction model makes use of the Javba Transaction API (JTA) and the Java Transaction Service (JTS). JTA specifies the interfaces between a transaction manager and the applications, resource managers, and application

server that are involved in a transaction. JTS is a Java programming language binding of the OTS. It provides a standard IIOP protocol for propagating transactions between servers.

The OTS transaction model is shown in Figure 16. The dashed box represents a transaction. It contains all of the objects that are participating in the transaction. Commits and rollbacks are applied to all the resource objects in this group.

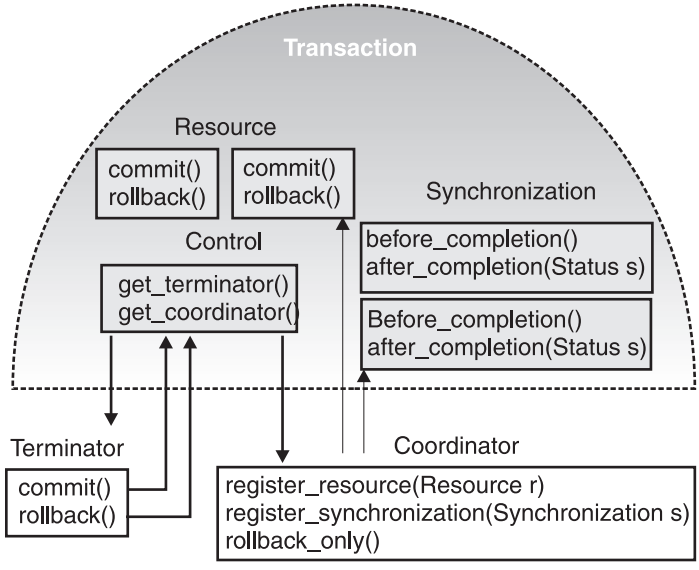


*Figure 16. OTS transaction model*

**control object**
Represents the transaction. An enterprise bean does not directly contact the control object. Instead, the control object is used by the container to manage the transaction on behalf of the bean.

**terminator object**
Used by the container to commit or roll back a transaction in cases where the container is required to terminate a transaction when a method returns.

**resource objects**
Implement the two-phase commit protocol. For example, if a resource object represents a connection to a database, committing the transaction causes the database to be updated. Rolling back the transaction reverts all database changes made through this connection since the transaction started. After the commit or rollback is completed, the corresponding rows in the database are unlocked. The

locking level is specified in the deployment descriptor. The resource objects allow each object to vote as to whether the entire transaction should be committed or rolled back.

**synchronization object**
Is notified as to whether a completed transaction was committed or rolled back. Unlike the resource objects, it does not participate in the two-phase commit protocol and plays a passive role in the transaction. (A session bean can play this role by implementing a special interface.)

**coordinator object**
Registers the resource and synchronization objects that participate in the transaction. An enterprise bean does not access this object directly. Transaction-aware objects that are intended for use with an enterprise bean register themselves by transparently obtaining a reference to the current transaction's coordinator.

OTS distinguishes *transactional* objects from *recoverable* objects. This distinction is relevant to enterprise beans.

- Transactional objects are associated with a transaction. They do not have commit and rollback methods and cannot be directly manipulated by the transaction. They serve as managers of the recoverable objects (or resources) that are associated with the object's current transaction.
- Recoverable objects have commit and rollback methods, allowing the transaction to directly manipulate their state or behavior.

An example of a transactional object is an enterprise bean that uses container-managed transactions. The container maintains a transaction on behalf of the bean. Recoverable objects that are allocated by the enterprise bean are transparently placed in the current transaction with the help of the container. Because the enterprise bean does not have a commit or rollback method, the transaction cannot manipulate the bean directly. (Creating an enterprise bean that is a recoverable resource requires additional work; enterprise beans rarely have an internal state that directly affects the outcome of a transaction.)

A bean can vote to roll back a transaction before the container attempts a commit or rollback. A bean can still be notified of the outcome of a transaction through the session synchronization interface.

### Implementing session synchronization

A session bean can optionally implement an interface that provides the bean with notifications of transaction synchronization in the form of container callbacks. Session beans use these notifications to manage database data that is cached within transactions.

The *afterBegin* notification signals a session instance that a new transaction has begun. At this point, the instance is already in the transaction and can do any database work it requires within the scope of the transaction.

The *beforeCompletion* notification is issued when the client of a session instance has completed work on its current transaction but the instance has not committed its resources. This is when the instance must write any database updates it has cached. The instance can cause the transaction to roll back by invoking the *setRollbackOnly* method on its session context.

The *afterCompletion* notification signals that the current transaction has completed. A completion status of *true* indicates the transaction committed; a status of *false* indicates a rollback occurred.

In the Advanced Application Server, entity beans can implement the session synchronization interface and can be notified when a transaction begins or completes. This type of session synchronization is used in the example in "Part 2. Using WebSphere Application Server" on page 107 to find out how many transactions are processed during testing. Each time an entity bean calls the afterBegin method, the relevant information is displayed on the console.

**Note:** Implementing session synchronization in an entity bean is not supported by all container vendors.

# Chapter 5. Developing Web applications

JSP pages, enterprise beans, and servlets can be used to implement Web applications in all editions of WebSphere Application Server. This section discusses issues related to developing Web applications with these components. It includes the following topics:

- "Web application programming model"
- "Using JSP pages, servlets, and enterprise beans in Web applications" on page 67

For more detailed information on the components of a Web application, see:

- "Chapter 2. Using JavaServer Pages" on page 17
- "Chapter 3. Using servlets" on page 25
- "Chapter 4. Using enterprise beans" on page 33

## Web application programming model

The Web application programming model is based on a multitiered architecture. Enterprise applications are partitioned into multiple components that can run on different computers, creating a set of physical tiers. Application components are assigned to logical tiers based on the functions that they perform. The physical and logical tiers do not have to correspond directly — for instance, application components that are assigned to different logical tiers can run on the same physical tier. This model is designed to support thin clients with high-function Web application and enterprise servers. Multitiered architectures and client topologies are described in more detail in *Getting Started with WebSphere Application Server*.

Figure 17 on page 66 shows an example of a multitiered Web application. The Web server provides the content that is displayed in the browser. To generate the content, the server communicates with a resource manager that manages a database and transaction processing monitor.

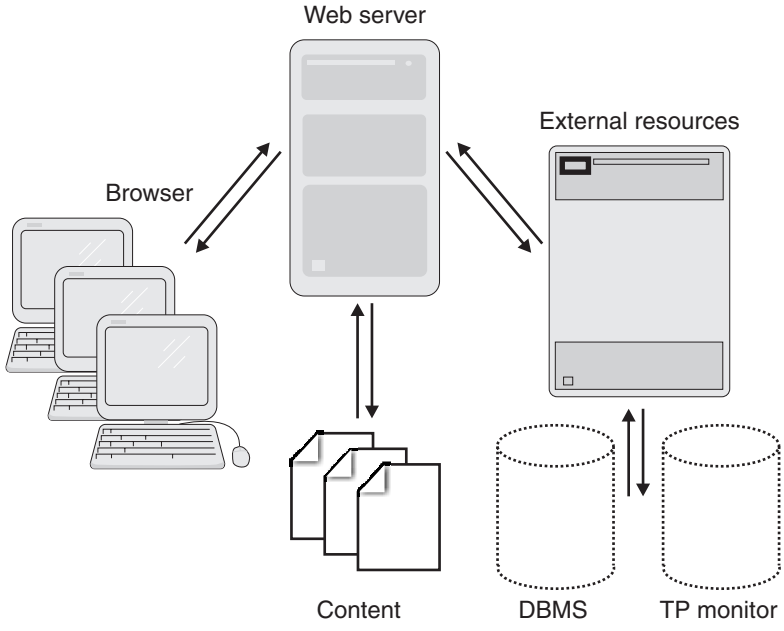


*Figure 17. Components of a Web application*

### First tier

The first tier contains the client — in this case, a Web browser. The client communicates with the Web server by using industry standard protocols such as Hypertext Transfer Protocol (HTTP) and Internet Inter-ORB Protocol (IIOP).

### Second tier

The second tier (or middle tier) resides between the client and the enterprise server resources and data. Application servers (such as WebSphere Application Server) are located in the middle tier. In this example, the middle tier includes a Web server that coordinates, collects, and assembles Web pages composed from static and dynamic content and delivers them to clients. It also includes a resource manager that controls access to the third tier.

This tier is the heart of the Web programming model. Although many Web applications still use a Common Gateway Interface (CGI) based programming model, a Java based programming model is becoming more popular. In the WebSphere Application Server sample application, the middle tier logic is implemented in Java by using servlets and JSP pages. Enterprise beans implement the application's business logic. Java beans act as the interface between servlets and the business logic.

### Third tier

The third tier contains resources that are used by the entire organization, such as databases and external transaction processing monitors (for example, CICS). Third-tier resources are managed by the second tier.

## Using JSP pages, servlets, and enterprise beans in Web applications

JSP pages and servlets are complementary technologies. This section discusses how they can be used together in Web applications. It includes the following:

- "Implementing a Model-View-Controller architecture"
- "Maintaining state in Web applications" on page 68
- "Implementing security in Web applications" on page 69

### Implementing a Model-View-Controller architecture

A Model-View-Controller (MVC) architecture divides an application into the following parts:

- **Model**—How the application works internally (for example, its business logic)
- **View**—How the client sees the state of the model (for example, a client user interface)
- **Controller**—How the client changes the application state or provides input to the application

To apply an MVC architecture to a Web application, the various WebSphere component architectures can be used as follows:

- JavaBeans components and enterprise beans can be used as the model, since they implement the business logic.
- JSP pages can be used as the view, since they display the dynamic content.
- Servlets can be used as the controller. They coordinate with other entities to handle tasks such as generating dynamic content. Servlets also handle Hyper Text Transfer Protocol (HTTP) connectivity. JavaBeans components can be used as the interface between the controller and the model.

Although servlets can also function as the view component of the MVC architecture, using them for this role is not recommended. If the servlet generates dynamic content, changes to the output format require it to be modified and recompiled. This makes the application more difficult to maintain and blurs the roles between developers and HTML authors. In addition, some servlet run times require the Web server to be stopped and restarted to load the latest version of the servlet. (In contrast, the WebSphere Application Server servlet run time always checks for servlet changes and automatically loads the latest version.)

Using JSP pages to view dynamic content solves these problems. JSP pages are independent from servlets. Changes to output formats are made in a JSP file and do not require the servlet to be recompiled. Using JSP pages to display content also separates the roles of HTML authors (who maintain the JSP pages) and developers (who maintain the servlet and the components that implement the business logic).

The sample application (described in "Part 2. Using WebSphere Application Server" on page 107) uses a MVC architecture.

- The model is implemented as enterprise beans. These business components perform processing tasks and represent permanent entities such as accounts, customers, and transaction records.
- The view is implemented by the Web site and the JSP pages. They control how the application displays data to the client.
- The controller is implemented by servlets. They receive client requests from the Web site (view) and pass them on to the business components (model) for processing. JavaBeans components act as an interface between the servlets in the controller and the components in the model.

## Maintaining state in Web applications

HTTP is a stateless protocol. It sets up a new connection for each client request and does not maintain information between requests. This means that a server cannot recognize whether a series of requests have come from the same client.

However, maintaining information across client requests is a core requirement for many Web applications. Several different approaches can be used to add state information to Web applications that use HTTP. These approaches include:

- Web server authentication
- Hidden form fields
- Cookies
- Servlet session management

All of these approaches are based on the concept of a *session* — a continuous connection from a browser over a fixed period of time.

### Web server authentication

Most Web servers have built in user authentication that permits access to resources only by clients who have logged in by using a user ID and password. The user ID can be used to track a client session. When a client logs in, the browser stores the user ID and sends it with every request.

This approach has the following advantages:

- Easy to implement.
- Automatically used by most Web servers.
- Handles client requests from different machines.

However, Web server authentication has two disadvantages:

- Users must login during every visit to your Web site. Although users expect and appreciate a login procedure when requesting sensitive information, they see it as intrusive and restrictive for public information.
- It does not support multiple sessions from the same client.

#### Hidden form fields

As their name implies, hidden form fields are fields in an HTML form that are not displayed in the client browser. They are sent to the server whenever the HTML form is submitted.

This approach has a number of advantages, including:

- It is supported by all browsers.
- A special server setup is not required.
- A user does not have to be logged in.

The disadvantage of hidden form fields is that they work only with dynamically generated forms. This approach is not possible for static Web pages. Hidden form fields also do not work with e-mailed pages, bookmarked pages, and browser shutdowns.

#### Cookies

A cookie is a piece of data passed between a Web server and a browser. The Web server sends a cookie to the browser, which stores it locally. When the browser accesses a page on the server, it sends the cookie back to the server, which uses it to identify the client. This makes it easy to track sessions.

The disadvantage of using cookies is that clients can configure browsers to reject them, which makes tracking sessions through cookies unreliable.

#### Servlet session management

Servlets have built-in session management. The servlet API defines a number of classes and interfaces to manage sessions. Servlets also use persistent cookies to track sessions. Session management is described in more detail in "Managing servlet sessions" on page 28.

### Implementing security in Web applications

Servlets , JSP pages, and enterprise beans use the same security model. See "Security services" on page 52 for details.

Servlets and JSP pages that communicate with other applications over the Internet have additional security needs. The Internet is a two-way communication channel that makes it possible for organizations to make information and services available to millions of users. It also makes it possible for users to break into these organizations. As companies provide services to their customers by using the Internet, they must implement security measures.

Securing the Web server involves securing the machine on which the Web server runs and securing the Web server itself. In order to secure the server, a user management system must be set up to allow different levels of access; anonymous access must be prohibited.

Securing the client's computer involves controlling what software the user is permitted to run. The key piece of software is the browser that is used to access information on the World Wide Web. Although browsers sometimes have security problems, clients that run the most up-to-date versions of the browser can take advantage of fixes for these problems.

Encryption is used to secure information in transit between the Web server and its intended recipient. Servlets and JSP pages can make use of the following encryption mechanisms:

**Secure Sockets Layer (SSL)**
SSL automatically encrypts information that is sent over the Internet and decrypts it before it is used. SSL sits between the raw TCP/IP data stream and the application. The standard TCP/IP protocol sends an anonymous stream of data between two computers. SSL adds features including authentication and data integrity.

SSL hides the complexity of encryption from both the user and the server developer. It allows for authentication of both the client and the server through digital certificates and digitally signed challenges. It also permits connections that are not encrypted but are authenticated and protected against tampering.

The SSL protocol is designed to protect against both man-in-the-middle and replay attacks. In a man-in-the-middle attack, the attacker intercepts all of the communications between two parties, making each think that it is communicating with the other. In a replay attack, an attacker captures the communications between two parties and replays the messages.

**Digital certificates**
Digital certificates consist of a private key and a public key. The private key is used to assign a signature to a block of data. The public key is used to verify the signature.

Figure 18 shows an example of how digital certificates work. Two entities, A and B, are communicating over the Internet. Entity A sends Entity B a message. Entity B signs the message with its private key and sends it back to Entity A. Entity A compares the message with its copy of Entity B's public key to make sure it is authentic. If a third party, Entity C, intercepts the message and alters it, the message does not match Entity B's public key, causing Entity A to reject it.

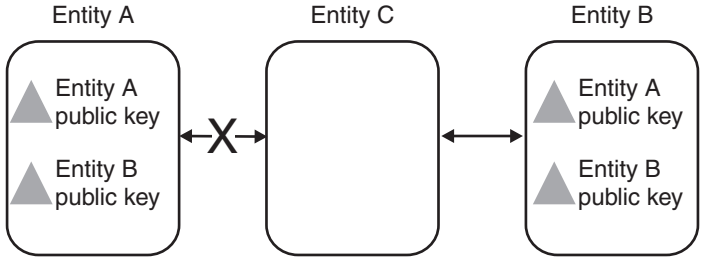


*Figure 18. Authentication using digital certificates*

### Client certificates

The purpose of a client certificate is to verify the identity of an individual. A client certificate binds a individual's name to a particular key. Client certificates can eliminate the need to remember user IDs and passwords. They eliminate anonymity and can be used to gather information about visitors to Web sites. Client certificates are sold by certificate authorities.

### Server certificates

A server certificate is issued by a Web server that implements SSL. When a browser connects to the Web server by using the SSL protocol, the server sends the browser its public key in a certificate. The certificate is used to authenticate the identity of the server and to distribute the server's public key, which is used to encrypt the initial information that is sent to the server by the client.

Server certificates are sold by certificate authorities.

# Chapter 6. WebSphere Application Server Enterprise Edition

WebSphere Application Server Enterprise Edition (also known as the Enterprise Application Server) extends the WebSphere Application Server Advanced Edition to handle enterprise-level distributed transaction processing and distributed object applications. It combines the TXSeries transactional application environment with the functionality offered by the Enterprise Edition services. Enterprise Edition also includes development tools for creating enterprise beans.

This section describes the following elements of the Enterprise Application Server:



## Enterprise Application Server features

The Enterprise Application Server has the full capabilities of the Advanced Application Server. In addition, it includes the following products:

### TXSeries

TXSeries offers cross-enterprise integration and provides high levels of application scalability, availability, integrity, longevity, and security. It consists of two popular middleware packages that are used to create distributed transactional applications:

#### Customer Information Control System (CICS)

CICS is IBM's general-purpose online transaction processing software. It is an application server that runs on a range of operating systems from the desktop to the largest mainframe. TXSeries CICS runs on AIX, Solaris, and Windows NT, but other versions of CICS run on OS/390®, OS/400, OS/2®, and VMS. CICS handles security, data integrity, and resource scheduling. It integrates basic business software services required by online transaction processing applications. CICS supports enterprise beans produced by IBM VisualAge for Java. It provides a natural evolution to the WebSphere Application Server programming model for customers that want to use their existing CICS skill sets.

### Encina

Encina is a family of software products used to develop and manage open distributed systems. It consists of the following products:

**Encina Monitor**
A transaction processing monitor that provides the means to develop, run, and administer transaction processing applications.

**Recoverable Queueing Service (RQS)**
RQS allows applications to queue transactional work for later processing.

**Structured File Server (SFS)**
SFS is a record-oriented file system that provides transactional integrity, log-based recovery, and broad scalability.

**Peer-to-Peer Communications (PPC) Services**
PPC services enables Encina transaction processing systems to interoperate with systems, typically mainframes, that have System Network Architecture (SNA) LU (Logical Unit) 6.2 communications interfaces.

**Encina++**
Encina++ is an object-oriented application programming interface (API) for Encina. It supports both CORBA and the Distributed Computing Environment (DCE). Encina++ servers can be written in the C++ programming language; clients can be written in the Java or C++ programming languages.

**Encina Toolkit**
The Encina Toolkit is a collection of modules, libraries, and programs that provide the functions required for large-scale distributed client/server system development. It includes Transactional-C, a transactional extension to the C programming language.

**DCE-Encina Lightweight Client (DE-Light)**
DE-Light extends the power of DCE and Encina to systems that are not running as DCE clients. It supports clients written in the Java and C programming languages, and provides a gateway server for accessing Encina applications.

The Enterprise Application Server provides application development tools for using Encina applications and enterprise beans with WebSphere applications and CICS applications. Encina supports customers who want to develop high performance transaction processing applications in the C, Java, or C++ programming languages.

## Enterprise Edition services

The Enterprise Edition services enable the Advanced Edition application server to function in an enterprise environment. The Enterprise Edition

services include a WorkArea service; tools for developing Business Rule Beans; an Internationalization service; a Java Messaging Service (JMS) Listener service; and support for Common Object Request Broker Architecture (CORBA) interoperability with WebSphere Application Server.

## Run time and system management architecture

The Enterprise Application Server run-time environment extends the functionality of the Advanced Application Server run-time environment. It is designed to be flexible and serve the run time and development environments of crucial business applications. It supports and implements industry standards such as CORBA and the EJB specification, which makes interoperability under secure and transactional environment possible.

Figure 19 shows how this interoperability is enabled.

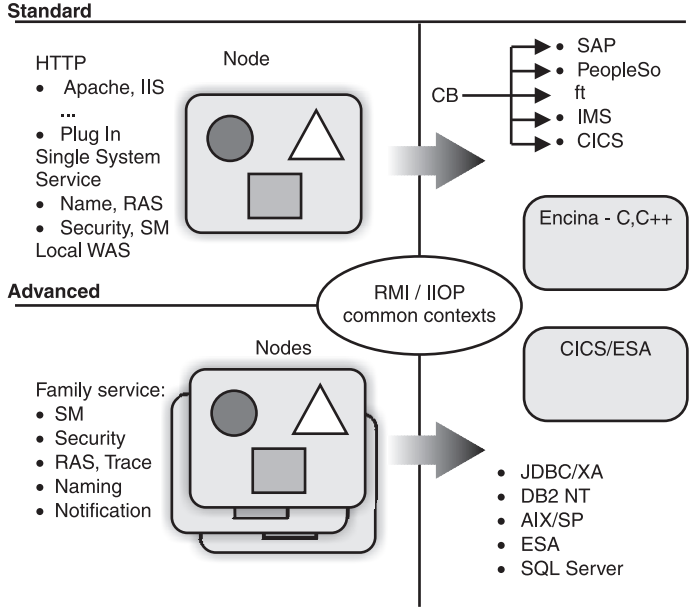


*Figure 19. Enterprise Application Server Interoperability*

- **Common protocols**—The Remote Method Invocation (RMI) protocol and the Internet Inter-ORB Protocol (IIOP) supply the common security, transactional, and naming context for WebSphere family interoperability.
- **Resource manager access**—Access to resource managers is achieved through the Common Connector Framework (CCF).
- **Servers**—The Enterprise Application Server is composed of one or more Advanced application servers providing Internet front-end support for

TXSeries CICS, Encina, or Encina++ systems. The Advanced application servers provide the run-time for servlets, JavaServer Pages (JSP), and enterprise beans that directly support Web applications. The EJB servers implement shared entities and business processes and map session beans to alternative run time implementations.

- **System Management**—The Enterprise Application Server System Management (SM) solution builds on the functions of the Advanced Application Server System Management. It provides integration with Tivoli® Suites system management.

  The Enterprise Application Server system management model manages and integrates all of the components that make up a business application. This differs from the Advanced Application Server, which can manage individual components but does not manage the entire application.

## Enterprise Edition application development environment and tools

The Enterprise Application Server contains a tool set for building applications that span all aspects of a customer-oriented and supplier-aware business. Whether you want to build a powerful Web presence, create distributed, transactional applications that can tie together non-Web business computing resources, integrate your Web and non-Web systems, or accomplish all of these goals, the Enterprise Application Server can help you.

The Enterprise Application Server contains all application development tools provided by the Advanced Application Server. In addition, it provides the following application development tools:

### WebSphere Studio

WebSphere Studio is included with the Enterprise Application server. For more information on this product, see "Application development environment" on page 9.

### IBM VisualAge for Java Enterprise Edition

VisualAge for Java is IBM's platform for designing, implementing, and testing enterprise beans. It provides support for:

- Session and entity enterprise beans
- Model import from and export to Rational Rose
- Unit test and debugging
- Enterprise bean inheritance and associations
- Relational database mapping
- Bottom-up production of enterprise beans from existing databases
- Support for by-value dependent objects
- Programming models for bean-managed persistence and session beans that make database calls

- Automatic generation of object adaptor beans
- Automatic generation of JSP pages
- Export of enterprise bean Java archive (JAR) files
- Export of predeployed JAR files for the Standard Application Server

VisualAge for Java Enterprise Edition is included with the Enterprise Application Server.

### IBM Enterprise Access Builder (EAB)

EAB provides support for integrating EJB applications with non-EJB applications. EAB imports legacy interface definitions to produce command adaptor beans, which are JavaBeans components that handle a single call and return interaction with an external application. Using a legacy application often requires more than a single call and response interaction. The EAB programming model can also combine multiple command adaptor beans into a special, non-portable enterprise bean that can be used to connect EJB applications with other types of applications.

### IBM TeamConnection®

TeamConnection supports team development in a distributed environment. It allows different levels of source code access, distributed compilation, and version control.

### VisualAge Component Development Toolkit

The VisualAge Component Development Toolkit consists of Object Builder, a distributed trace facility, and a debugger. These tools provide the following functionality:

- Cache and object-oriented SQL (OOSQL)
- Integration of EAB parts with Application Adaptors
- Distributed trace and debug
- Rational Rose integration
- A C++ implementation of the EJB interface

### VisualAge for C++ Professional Edition

VisualAge for C++ provides a complete C++ development environment. The development environment is especially valuable for high-performance and highly computational applications. Its Open Class Library provides advanced class libraries and frameworks to build robust applications on AIX and Windows NT systems.

### IBM DB2®

IBM DB2 is a distributed relational database that can be used as a resource manager in conjunction with TXSeries. DB2 can be used by the EJB administration servers contained in the Advanced Application Server. It can

also be used to store persistent data associated with container-managed persistence (CMP) entity beans in both the Advanced and Enterprise Application Server.

## MQSeries

MQSeries® is IBM's premier messaging and queuing service. It provides an open architecture for integrating enterprise business processes. MQSeries applications exchange information across different platforms by sending and receiving data as messages. The underlying MQSeries software takes care of network interfaces, assures delivery of messages, and deals with communications protocols so that programmers can use their skills to handle key business requirements, instead of wrestling with underlying network complexities.

Although MQSeries is not formally part of the WebSphere platform, a version of MQSeries that is licensed specifically for use with WebSphere Application Server is included with the Enterprise Application Server. MQSeries can be used to improve the scalability, performance, and portability of multitiered WebSphere applications. It integrates with Java applications running under WebSphere Application Server to provide access to back end resources and legacy systems on a wide variety of platforms.

# Chapter 7. Using TXSeries

IBM TXSeries is an advanced transaction processing solution that coordinates and integrates servers, managing high-performance applications and data sources across the network. It enables customers to create a distributed, client/server environment with all the reliability, availability, and data integrity required for online transaction processing.

This section discusses the following members of the TXSeries product family:

- Encina, a family of software products used to develop and manage open distributed systems and perform transaction processing. For more information, see "TXSeries Encina".
- Customer Information Control System (CICS), which is IBM's general-purpose online transaction processing software. For more information, see "TXSeries CICS" on page 94.

This section provides background information for the WebSphere family example application described in "Part 2. Using WebSphere Application Server" on page 107. It does not focus on the practical issues of using TXSeries to develop enterprise transaction processing applications.

- For general information about TXSeries, see the *Concepts and Facilities* guide.
- For more information about Encina, see the Encina product documentation.
- For more information about CICS, see the CICS product documentation.

## TXSeries Encina

Encina is a family of software products used to develop and manage open distributed systems. Using the underlying technology of the Open Group Distributed Computing Environment (DCE), Encina provides the infrastructure to handle the complexities of large distributed systems and to maintain data integrity across them. Furthermore, Encina simplifies many aspects of programming distributed systems, allowing application developers to concentrate on the business logic of the program and to ignore many of the underlying details.

This section provides an overview of the following Encina family members:

- "Encina Monitor" on page 80
- "The Recoverable Queueing Service (RQS)" on page 84
- "The Structured File Server (SFS)" on page 85
- "Peer-to-Peer Communications (PPC) Services" on page 86

For more information on Encina, see the Encina product documentation.

## Encina Monitor

The Encina Monitor, or just the Monitor, is a transaction processing (TP) monitor that provides the means to develop, run, and administer transaction processing applications. The Encina Monitor, in conjunction with resource managers, provides an environment to maintain large quantities of data in a consistent state, controlling which users and clients access specific data through defined servers in specific ways. The Monitor provides an open, modular system that is scalable and that interoperates with existing computing resources such as IBM mainframes.

### Encina Monitor functionality

The Monitor provides three functional areas for a TP system:

- **Runtime environment.** The Monitor runtime environment coordinates TP applications and resource managers, and performs runtime administration tasks, such as load balancing and collecting diagnostics. In addition, this environment provides for other interactions with the execution environment, such as scheduling calls for later execution and retrieving information about users, transactions, data-dependent routing, and client/server bindings.
- **System administration facility.** The Monitor system administration interface is used to construct, initiate, control, and terminate a Monitor system. Administering the Monitor is done through Monitor administrative and configuration interfaces.
- **Application development environment.** Monitor applications are developed using the Monitor application programming interface (API) in conjunction with other Encina interfaces, such as Transactional-C (Tran-C). The Monitor saves the programmer effort by performing some tasks, such as interaction with DCE RPC and Security, on the application's behalf. Servers can be developed using Tran-C or other transactional interfaces such as the one provided by the Monitor API. The Monitor API also provides functionality to manage client and server application programs in a distributed transaction processing environment.

### The Monitor runtime environment

The Monitor provides an environment in which client/server applications can be run. This runtime environment is the Monitor *cell*, or single administrative unit. A cell consists of a collection of *nodes*; each node is a machine on which applications and Monitor software run. The nodes that make up a Monitor

cell are a subset of the nodes in a DCE cell; the term cell in this document refers to a Monitor cell unless stated otherwise. A DCE cell can contain more than one Monitor cell. Some services, such as security and the name service, are provided by the DCE cell, and so all nodes must be able to access these DCE services.

The physical architecture of a Monitor cell, shown in Figure 20 is a cluster of nodes connected together by a local area network, with potentially one or more wide area network connections to similar clusters.

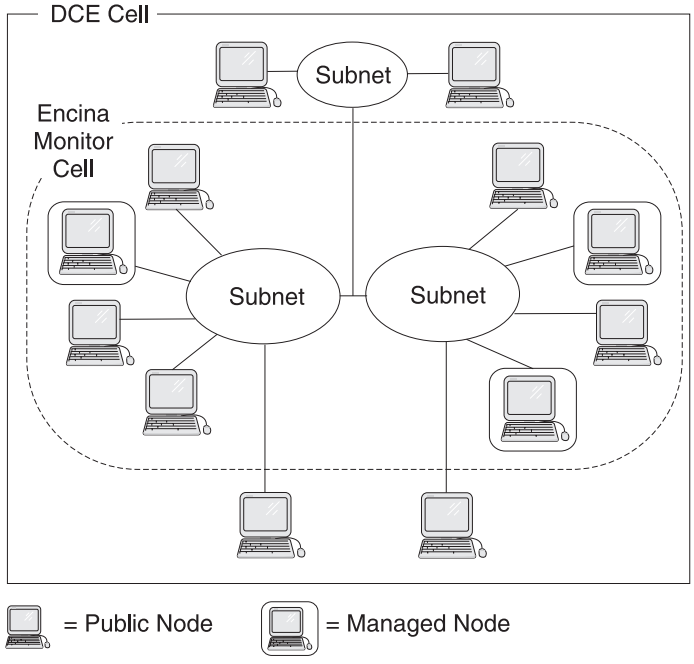


*Figure 20. Physical Architecture of a Monitor Cell*

A Monitor cell contains the following components:

**Cell manager**
The part of the Monitor that monitors and controls the node managers and data repository within the cell. The cell manager communicates with node managers, clients, and DCE services. The most important task of the cell manager is the management of the data needed to configure and administer the system.

**Node manager**
The part of Encina that controls all application servers on a single managed node on behalf of the cell manager. The node manager starts and monitors the application servers running on a single node. The

cell manager and its node managers monitor the system constantly, detecting, reporting, and restarting application servers that have failed.

**Application server**
Application servers are the components that process client requests, usually by interacting with one or more resource managers. They run on managed nodes and exports services required by the application's clients.

An application server consists of one or more processes called processing agents (PAs) that receive and handle client requests for services. The Monitor automatically parcels out client requests among the various processing agents

**Application client**
The part of the application through which a user interacts with the Monitor system, making requests for services exported by active application servers. A client can incorporate an interactive user interface or use other methods for generating the requests. Clients are the only components that do not need to run on managed nodes.

**Resource manager**
A component that manages a shared resource, such as application data. Application servers communicate with resource managers. Note that Monitor resource managers include the Structured File Server (SFS) and Recoverable Queueing Service (RQS).

The Monitor manages several kinds of data for its cell, including configuration data for the cell and Monitor components and diagnostic data for each configured diagnostic class.

For more information on the Monitor runtime environment and administering Encina applications, see *Encina Administration Guide Volume 1: Basic Administration*.

### Monitor interactions with DCE

The Monitor provides a transaction processing environment built on top of DCE and the Encina Toolkit. Encina extensions to the underlying DCE components provide the necessary application execution environment and associated administration facilities. The Encina Toolkit provides the semantics necessary for transactional integrity.

DCE is a modular collection of interfaces that provides the basic building blocks for constructing distributed systems. It supplies the following:

- Remote procedure calls (RPCs), which provide the programming paradigm for communication between clients and servers. The Monitor's Transactional RPC (TRPC) mechanism is built on top of DCE RPC.

- Cell Directory Service (CDS), which provides a consistent way of identifying resources in a DCE cell.
- DCE Security Service, which provides the necessary authentication and authorization controls for secure operation of Monitor applications

### Monitor application development environment

The Monitor development environment consists of the application development tools, languages, and libraries that enable programmers to develop application programs for the Monitor. The development environment consists of three major components:

- The Transactional-C (Tran-C) programming language is a set of extensions to the C programming language that simplify the development of transactional applications. The Tran-C runtime system automatically invokes the necessary functions to support the Tran-C functions used in a program and monitors the scope and state of transactions and their associated low-level data structures and constructs.
- Encina Transactional RPC (TRPC) is used to define and execute client/server interactions Interfaces are described using the Transactional Interface Definition Language (TIDL), which is an extension to the DCE IDL. The TRPC runtime, which is built on top of the DCE RPC, serves as the communication mechanism between an Encina Monitor client and an application server.
- The Encina Monitor API is a collection of functions that support the development of distributed transaction processing applications, such as functions allowing clients and application servers to register with the Monitor system.

Additional programming and diagnostic tools are available only on the Windows NT and Windows 95/98 systems platforms.

- The Encina Server Wizard can be used to create Encina and Encina++ servers. It generates much of the standard initialization code for the server, organizes the code into a project, and associates the appropriate Encina and system libraries required to build a server.
- The Encina COM Wizard is used to create an Encina COM component (in the form of a DLL file) from an Encina TIDL interface. The DLL file can then be incorporated into a client written in any language to enable that client's access to any Encina server that exports the interface defined in the DLL file.
- The WinTrace tool aids developers in debugging distributed client/server applications. This Encina-specific tool is used to format and view application output and Encina trace files and to translate error codes and trace identifiers. It can also be used to start Encina Trace Listener servers for use in viewing output while a process is running. For information on using this tool, consult its online help.

For more information on writing programs that run in the Encina Monitor environment, see the following documents:

- *Writing Encina Applications*
- *Writing Encina Applications on Windows Systems*
- *Encina Transactional Programming Guide*
- *Encina Monitor Programming Guide*

## The Recoverable Queueing Service (RQS)

The Recoverable Queueing Service (RQS) allows applications to queue transactional work for later processing. Applications can store data related to a task in a queue. This data can be subsequently processed by another program. Applications can then commit their transactions with the assurance that the queued work will be completed transactionally at a later time.

*Queues* are linear data structures that can be used to pass information from one application to another. Applications *enqueue* (add) elements to the tail of a queue and *dequeue* (remove) elements from the head of a queue in a first-in first-out (FIFO) manner (although RQS also provides other ways of accessing elements).

Each queue is maintained by a single RQS server. All interactions with that queue are handled by the server. An RQS server can contain multiple queues—for example, one or more queues for storing data associated with each of the billing, shipping, and inventory maintenance tasks of a retail business.

Applications use queues to store data in the form of elements. An *element* is record-oriented data specific to an application. The fields of an element store related pieces of the information. For example, a billing element might have fields for storing the customer name, customer account number, and current account balance.

Typically, business transactions are structured so that each step in the work of a transaction must complete successfully before the entire transaction can finish. An RQS server permits a client to take a subtask of a transaction, represent the subtask with data, and enqueue the data.

An application can requeue an element to another queue for subsequent processing by another application. *Requeueing* is the process of moving an element from one queue to another.

Applications that select from several different queues when processing dequeue requests can use *queue sets* (collections of queues) to simplify the

selection process. A queue can belong to more than one queue set. A queue that belongs to a queue set can be accessed as part of that queue set or can be accessed individually.

*Locking* guarantees the consistency of elements and queues in RQS. RQS supports locking for the duration of an operation, or for the duration of a transaction. The locking models provided by RQS support high concurrency among multiple applications that are accessing the same elements or queues.

For more information on writing RQS programs, see the *Encina RQS Programming Guide*. For more information on administering an RQS server, see the *Encina Administration Guide Volume 2: Server Administration*.

## The Structured File Server (SFS)

The Encina Structured File Server (SFS) is a record-oriented file system. SFS uses *structured files*, which are composed of records. A *record* is a grouping of related information with a predefined size and a predefined number of *fields*, which hold specific parts of the record's information. These fields can be of various predefined data types. The field layout of a record is defined when the file is created. For example, each record can contain information about an employee, with fields for the name, employee number, and salary.

SFS provides both data processing and administrative functions. The data processing functions provide the standard operations used to access and modify data: read, insert, update, delete, lock, unlock, and so on. The administrative functions enable programs to create, query, and modify SFS files and volumes, copy files, delete files, and so forth.

All data in SFS files is managed by the SFS server. Programs that require access to this data must submit their requests to that server, which retrieves the requested data or performs the specified operation.

SFS provides a number of benefits, including:

- **Transaction protection**. SFS provides transactional access to data stored in a file. Files managed by SFS are thus fully recoverable from server problems, network outages, and media failures.
- **Support for distributed computing and open systems**. SFS provides a consistent mechanism for requesting access to structured data across multiple platforms. The client/server model used by SFS allows applications to be easily and transparently distributed on the network.
- **Ease of porting existing applications**. SFS enables you to simplify porting existing structured file or database applications by providing a logical (rather than physical) data model.

- **ISAM compatibility**. The Encina Transactional Indexed Sequential Access Method (T-ISAM) library provides an X/Open ISAM-compliant method of accessing data stored in SFS.
- **COBOL record interface**. The SFS External File Handler (EXTFH) supports the use of Micro Focus COBOL with SFS. Existing COBOL applications, using standard COBOL I/O statements, can be made to access SFS files; the native COBOL I/O calls are transparently mapped to SFS calls.
- **Compatibility with database systems**. The Encina Transaction Manager-XA (TM-XA) Service enables SFS applications to interact with database applications that support the X/Open XA interface.
- **Compatibility with RQS** Many SFS are compatible with RQS. For example, the field types used by SFS have corresponding types in RQS. Thus, applications can easily use both SFS and RQS.

For more information on writing SFS programs, see the *Encina SFS Programming Guide*. For more information on administering an SFS server, see the *Encina Administration Guide Volume 2: Server Administration*.

### Peer-to-Peer Communications (PPC) Services

PPC Services enables Encina transaction processing systems to interoperate with systems, typically mainframes, that have System Network Architecture (SNA) LU (Logical Unit) 6.2 communications interfaces. It enables integration and migration between mainframe systems using SNA and Encina transaction processing systems.

PPC Services enable bidirectional communications, so that both applications and data can be shared between mainframes and Encina, with either side initiating communications. Applications that run on an SNA network can allocate a conversation through the PPC gateway to an application in the Encina Monitor.

PPC Services supports several interfaces that provide thread-safe routines for execution in DCE environments.

- Distributed Program Link (DPL). DPL enables Encina PPC Executive applications to operate in a client/server relationship with CICS applications and other Encina DPL applications. DPL applications communicate with other applications in a way that is similar to an Encina transactional remote procedure call (TRPC).
- Both the X/Open Common Programming Interface Communications (CPI-C) and the IBM System Application Architecture (SAA) CPI-C. The CPI-C interface provides a more generalized way to implement peer-to-peer communications between a PPC Executive application and a remote application.
- The SAA Common Programming Interface Resource Recovery (CPI-RR) interface for transaction demarcation.

- The X/Open TX interface for transaction demarcation.

These interfaces are not mutually exclusive. The interfaces that you use depend on your environment and the requirements of your application.

The PPC Services model for communicating through a gateway server is illustrated in Figure 21.

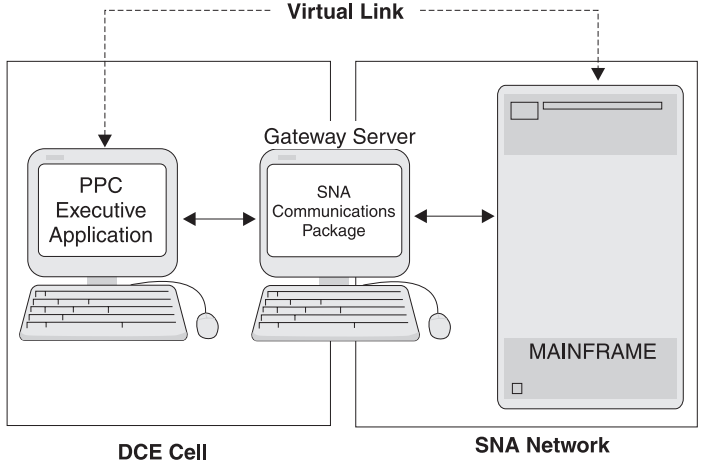


*Figure 21. PPC communications model*

The gateway server runs on a machine that is part of a DCE cell and an SNA network. The Gateway/SNA establishes a virtual link between an SNA LU 6.2 application on a mainframe and a PPC Executive application on a DCE node. PPC Executive applications are fully integrated into the Encina/DCE environment. That is, an application can communicate with a mainframe using SNA while using Encina and DCE to communicate with other applications in the Encina/DCE environment.

The two products that make up a PPC application are:

- **The PPC Executive**—A library that supports peer-to-peer communications and two-phase commit transactional semantics. Programmers use the functionality provided by the PPC Executive to create programs (*PPC Executive applications*) that communicate with other applications that support the SNA LU 6.2 protocol.
- **The PPC Gateway Server**—A communications manager that provides communications between Encina applications in a DCE cell and LU 6.2 applications in an SNA network. Allocating conversations through a gateway is bidirectional, enabling transactional access to data on either a mainframe or a machine in the Encina system from an application on the

other network. The gateway server isolates SNA requirements (such as logging and recovery) from PPC Executive applications.

For more information on writing PPC programs, see the *Encina PPC Services Programming Guide*. For more information on administering a PPC Gateway server, see the *Encina Administration Guide Volume 2: Server Administration*.

## Encina++

Encina++ is an object-oriented application programming interface (API) for Encina. It is composed of classes that access many Encina components.

### Encina++ interfaces

Encina++ supports the development of object-oriented applications that are based on DCE, CORBA, and a mixture of both DCE and CORBA. The latter type of application is often referred to as a mixed application. Of the components that make up Encina++, some can be used only with Encina++/DCE or only with Encina++/CORBA, and some can be used by either or both.

**Common Encina++ interfaces:** The following Encina++ interfaces can be used in any type of Encina++ application:

**Encina++**
The Encina++ interface defines C++ classes and member functions that enable the creation and management of client/server applications and provide support for the underlying environment.

**Transactional-C++ (Tran-C++)**
The Tran-C++ interface defines C++ constructs and macros as well as classes and member functions for distributed transaction processing. This interface provides an object-oriented alternative to the Encina Transactional-C interface.

**Object Transaction Service (OTS)**
The Object Management Group (OMG) OTS interface also defines C++ classes and member functions for distributed transactional processing. This interface implements the OMG *Object Transaction Service* specification as documented in *OMG document 94.8.4*.

For more information on writing programs using the common Encina++ interfaces, see the *Encina Object-Oriented Programming Guide*.

**Encina++/DCE interfaces:** The Encina++ programming interfaces that are supported only for DCE (including mixed applications) provide object-oriented access to two types of Encina servers offering specialized services:

**Recoverable Queueing Service C++ interface (RQS++)**
The RQS++ interface defines C++ classes and functions for enqueueing and dequeueing data transactionally at an RQS server.

**Structured File Server C++ interface (SFS++)**
The SFS++ interface defines C++ classes and functions for manipulating data stored in record-oriented files at an SFS server while maintaining transactional integrity.

The Encina Data Definition Language (DDL) compiler is used to generate the classes used by RQS++ and SFS++ applications. For more information on these interfaces, see the *Encina RQS++ and SFS++ Programming Guide*.

**Encina++/CORBA interfaces:** Encina++ provides two programming interfaces that are supported only for CORBA applications and mixed DCE-CORBA applications:

**Object Concurrency Control Service (OCCS)**
The OCCS interface defines C++ classes and functions that enable multiple clients to coordinate access to shared resources. This interface implements the OMG *Concurrency Control Service Proposal* as documented in *OMG TC Document 94.5.8*.

**Java OTS Client interface**
The Java OTS Client interface defines Java classes and functions that enable Java client applications to begin and control distributed transactions. This interface implements the OMG *Object Transaction Service* specification as documented in *OMG document 94.8.4*.

For more information on writing programs using the Encina++/CORBA interfaces, see the *Encina Object-Oriented Programming Guide*.

### Encina++ programming model

The Encina++ classes support a client/object programming model in which clients access objects instead of servers. Servers export one or more interfaces (classes) and one or more instances of each class (objects). The client application can access objects exported by servers without the application developer knowing how the objects available in the system map to servers.

Clients can bind to objects exported by servers. They can bind to individual objects when the objects are known, or they can bind to a class when the objects are not known or when all objects of a specific class provide the same capabilities. Typically, the application developer specifies a name for an object. Although each object created has a universal unique identifier (UUID), naming an object allows clients to bind to the object by name instead of by UUID.

In Encina++, an interface definition language (IDL) is used to specify the interfaces to objects in the form of remote procedures. The remote procedures are used for communications between the client and server applications. The interface compiler generates files that include client stub and server stub classes for each interface. These stub classes give the client and server a slightly different view of the same interface.

### Client and server support

Encina++ offers the following features for object-oriented, distributed transaction-processing applications:

- Initialization of clients and servers
- Transparent and explicit binding
- Object registration and binding
- Integration of XA-compliant databases
- Transactional and nontransactional threads
- Integrated exception handling

Encina++ enables you to develop several different types of client and server applications in C++ as well as Java clients that access Encina++ servers.

The Encina++ interfaces are designed to support functionality exported by the Encina Monitor and can be used to create Monitor application servers and clients in C++. Monitor application servers and clients can use DCE or CORBA or both. See the *Encina Monitor Programming Guide* and the *Concepts and Planning* for more information on the Encina Monitor.

The Encina++ interfaces also support the development of C++ client and server applications that do not run under the control of the Encina Monitor. These applications can use either DCE, CORBA or both.

Encina++ provides a set of Java classes for use with the OrbixWeb object request broker from IONA Technologies, Ltd. OrbixWeb allows you to build distributed applications in the Java language. The Encina++ Java interfaces allow you to write Java client applications that begin and control transactions by using the Java-language implementation of the OTS. Encina++ Java clients can bind to objects exported by Encina++/CORBA servers.

## The Encina Toolkit

The Encina Toolkit is a collection of modules, libraries, and programs that provide the functions required for large-scale distributed client/server system development. The modules of the Toolkit include log and recovery services, transaction services, and Transactional Remote Procedure Call (TRPC, an extension to the DCE RPC technology). These modules transparently ensure distributed transactional integrity. The Toolkit also provides Transactional-C (Tran-C), a transactional extension to the C programming language.

Lower-level modules of the Encina Toolkit include the following:

- The Distributed Transaction Service (TRAN), which coordinates transactions.
- The Lock Service (LOCK), which prevents conflicting access to data.
- The Log Service (LOG) and Recovery Service (REC), which guarantee that changes made to data on behalf of a transaction are either performed in their entirety or appear never to have occurred.
- The Volume Service (VOL), which enables applications to address storage in terms of logical units called *volumes*. Volumes can consist of single or multiple physical disk partitions, can include entire disks, and can span multiple physical disks. The Volume Service maintains the storage used by the Log Service, which in turn stores the data required by the Recovery Service to restart a recoverable application.
- The Transaction Manager-XA Service (TM-XA), which implements the transaction manager side of the X/Open XA interface. TM-XA coordinates distributed transactions with relational database managers.
- The Transarc/Encina DCE Utilities Library (TRDCE), which provides utilities for constructing client and server programs.

Application programs do not generally use the lower-level modules directly, although they can be accessed if needed. For example, Tran-C (not TRAN) is used for creating transactions. Tran-C itself calls TRAN; the application does not have to call TRAN directly.

For more information on using the Encina Toolkit, see the *Encina Toolkit Programming Guide*.

## DCE-Encina Lightweight Client (DE-Light)

The DCE Encina Lightweight Client™ (DE-Light) is a set of application programming interfaces (APIs) and a gateway server that extends the power of DCE and TXSeries Encina to systems that are not running as DCE clients.

You can use DE-Light to build clients that require less overall effort to create, involve fewer administrative resources, and generate less network traffic than standard DCE or Encina clients. Yet these DE-Light clients can still take advantage of the benefits of load balancing, scalability, and server replication that were formerly available only to full DCE and Encina clients. In addition, DE-Light enables you to access DCE and Encina from systems that do not support DCE, but that do support Java.

DE-Light is composed of the following components:

- Java API—Used to develop Java clients for standalone Java applications. DE-Light Java clients communicate with gateways via TCP/IP and Hypertext Transfer Protocol (HTTP).

- C API—Used to develop clients for Microsoft Windows environments. DE-Light C clients use TCP/IP to communicate with gateways at known endpoints.
- Gateway server—Enables communications between DE-Light clients and DCE or Encina.

DE-Light Java clients can consist of either of the following:

- A Java applet embedded into a Hypertext Markup Language (HTML) document residing on a Web server. With this type of client (shown in Figure 22), a user contacts a Web server through a Java-enabled Web browser, and the browser automatically downloads the DE-Light Java client applet along with the HTML document. The DE-Light Java client applet then contacts the appropriate DE-Light gateway. The gateway is a separate server that resides on a DCE client machine. The Web client can now communicate with DCE or Encina servers via the DE-Light gateway.
- A standalone Java application.

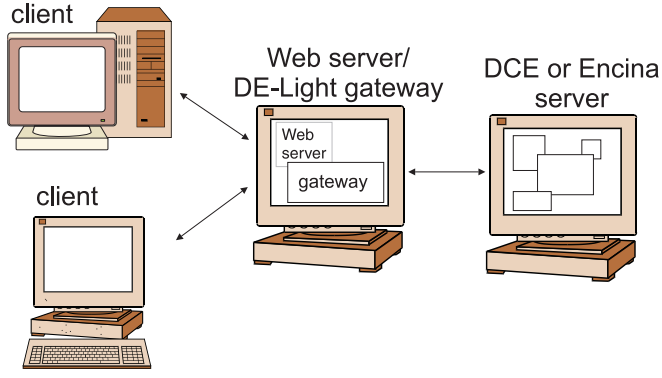


*Figure 22. DE-Light Java client*

For more information on how to write DE-light programs, see the *Encina DE-Light Programming Guide*. For more information on how to administer a DE-light gateway, see *Encina Administration Guide Volume 2: Server Administration*.

## WebSphere Advanced to Encina Interoperability

Advanced Application Server applications written in the Java programming language can use the WebSphere Advanced to Encina Interoperability functionality to communicate with Encina and Encina++ servers. This functionality is used in the sample application to enable the Java-based sample application to communicate with an Encina++ server.

Java applications communicate with an Encina or Encina++ application by using a bridge server. The bridge server exports CORBA interfaces whose methods correspond to the TIDL interfaces of an Encina or Encina++ application. Java applications contact the bridge server by using IIOP; the bridge server makes TRPCs to the Encina Monitor Application Server. Calls from the Java client to Encina are in one direction only.

**Note:** The CORBA interfaces are encapsulated; they cannot be called directly. CORBA clients are *not* supported.

Figure 23 illustrates a distributed system that uses a bridge server.
The **wstidl** tool takes an Encina TIDL file as input and generates the required

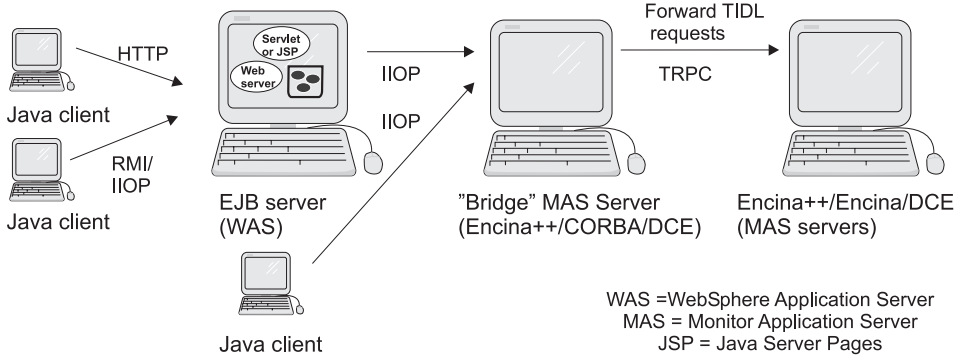


*Figure 23. Interoperability between Java applications and Encina/Encina++ servers*

CORBA IDL interfaces, Java classes, server main function, and other files used to achieve connectivity. The generated bean or enterprise bean can be used as part of an Advanced Application Server application.

The connectivity layer on the client side is embedded in the bean class. On the bridge server, the connectivity layer consists of CORBA interfaces and additional specialized files that map TIDL parameters, TIDL data types, and Encina exceptions to analogous elements in CORBA.

Using a bridge server provides the following additional advantages:

- The bridge server uses the IBM C++ ORB. Thus, you do not have a dependency on third-party ORBs.
- The bridge server propagates transactions originating at the client to the server, preserving the appropriate behavior as defined in the original TIDL file.

For more information on how to create a bridge server, see the *Encina Object-Oriented Programming Guide*.

## TXSeries CICS

CICS is IBM's general-purpose online transaction processing software. It an application server that runs on a range of operating systems from the desktop to the largest mainframe. TXSeries CICS, which is part of WebSphere Enterprise Edition, runs on AIX, Solaris, HP-UX, Windows 2000, and Windows NT, but other versions of CICS run on OS/390, OS/400, OS/2, VMS, and other platforms. CICS handles security, data integrity, and resource scheduling. It integrates basic business software services required by online transaction processing applications.

This section provides a high-level overview of CICS. For more information, see the TXSeries *Concepts and Planning* guide and the CICS documentation set.

### Basic CICS concepts

This section describes the components of a CICS system. For more information on basic CICS system concepts, see the *Concepts and Planning*, the *CICS Administration Guide for Open Systems*, the *CICS Administration Guide for Windows Systems*, and the *CICS Administration Guide for Windows Systems*.

#### Regions

An instance of a CICS system is called a CICS *region*. On UNIX or Windows systems, this region consists of a number of processes. A CICS region provides the transaction processing services that are convenient to manage as a single administrative unit. These services typically support the business logic of user applications, running as transactions requested by CICS client applications and 3270 terminal users.

A CICS region frees application programs from having to negotiate with the operating system to acquire and release resources. Each task being processed uses processor cycles, processor memory for the programs doing the work, and memory for the task and user data. It needs small areas of memory for scratchpad-type calculations and, sometimes it needs data-communications channels, data files, and databases. For all these, the region gets the resources needed from the operating system. It then allocates the resources to the tasks that need them. It reacquires the resources when the tasks complete their processing or specifically release a resource.

A CICS region controls the simultaneous access to resource managers, even across multiple machines and platforms, and preserves the integrity of data updates. For example, it synchronizes updates, logs changes, and recovers in-doubt updates. It can also use security services to ensure that only authorized users can access and update data.

Figure 24 on page 95 shows the components that make up a CICS region.

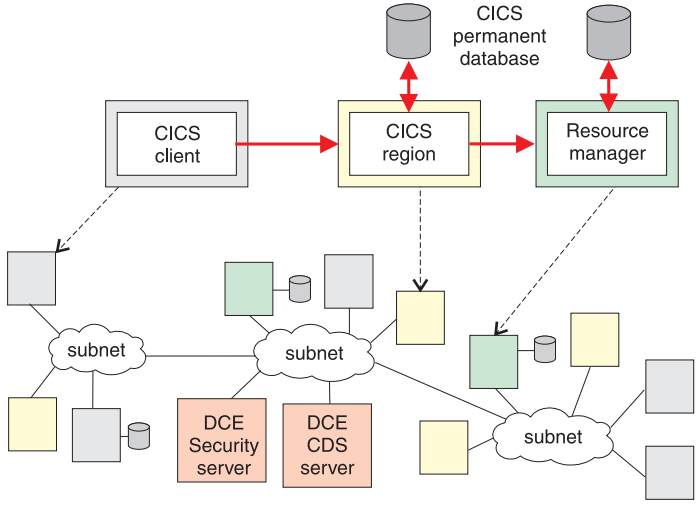


*Figure 24. A CICS region*

It is common to run multiple CICS regions on the same system. These regions can be independent of one another—for example, one for accounting, one for inventory management, and so on—or they can be closely tied together. CICS provides a number of facilities for inter-region (also called intersystem) communication. (See "CICS intersystem communication" on page 101 for more information.)

### Tables

CICS is a table-driven system. That is, the operation of a CICS region is defined and controlled by a number of tables. These tables tell CICS which files to use, which terminals to use, which transactions are available (and which programs these transactions are associated with) and so on. The contents of these tables are controlled by resource definitions. CICS administration involves a large number of resource definitions. Every transaction, program, file, queue, terminal, remote system, and so on must be defined.

### Transactions

Each user's interaction with a CICS region involves one or more transactions. In CICS terms, a *transaction* is a basic operation that is offered to the user. For example, a banking application could include a query transaction, a debit transaction, a funds-transfer transaction, and so on. Note that this use of the term is a bit different than the term as used in other contexts (such as how the term is used by Encina). CICS calls a group of actions that must be performed as an atomic unit of work and which must be durable and recoverable a *logical unit of work* (*LUW*).

The transactions that make up an application are written by CICS application developers. An administrator specifies which transactions are to be offered to users in the region's Transaction Definitions. The users typically select transactions by specifying the four-character name for the transaction. CICS then schedules the transaction to run. The transaction is then run in a process called an *application server*. If the transaction is implemented by several programs, those programs can run in the same or separate processes. The CICS region monitors the progress of the program, serving its requests for data, communications, and other resources. When the transaction completes, the CICS region commits any data changes, terminates the program, and frees resources for use by other transactions.

You can distribute transactions across multiple CICS regions to spread the workload across those regions. You can predefine whether a transaction request is to be served locally (on the CICS region that received the request), routed to a specified remote region to be run there, or routed dynamically to any region that can run the transaction.

A number of predefined transactions are supplied with CICS. These transactions allow users to sign on and off. They also provide utility, management, and debugging facilities.

### Task management services

*CICS Task management* services control the creation, processing, and termination of tasks. Whenever possible, the CICS region provides the best response times to the most important work. Usually, several tasks are competing for processing time, so the CICS region determines the priority. At any time there are likely to be many tasks to be performed concurrently, all requiring use of the same resources. The region schedules and dispatches tasks according to their relative priority and the availability of application servers and other system resources. This controls the rate and order in which tasks are processed, thereby minimizing the chances of conflict or system overload. The CICS region calculates the priority several times during the lifetime of a task. For example, a task can be processed until it needs input from a file or a user. The CICS region suspends the task, awaiting its input, and starts a new waiting task or resumes work for another suspended task.

### Program management services

*CICS Program management* services are used to associate a task with the application program that is to do its work. Although many tasks can need to use the same application program, program control loads only one copy of the code into memory. Each task threads its way through this code independently, so many users can run tasks concurrently using the same physical copy of an application program.

If a task involves many interactions with the user (for example, for data to be input), it is usually implemented as a number of programs that run in sequence and end before the next program starts. This technique is called *pseudoconversational programming*. When each program ends, it displays a screen for the user to input data. The CICS region remembers which program to run next to process the input, but releases the memory used by the task and the program that ended. If that program was not being used by other tasks, the region also reuses the memory used to hold the program. So, while the CICS region is waiting for a user to input data, the system resources are freed to be used by other tasks. However, the user is unaware of the program ending and can continue to communicate with the system, almost as if in a conversation.

### Time management services

Time management services enable programs to start and control a range of time-dependent operations—for example, starting a transaction (task) at a certain time of day and signaling when a specified time period has elapsed. These services also enable date- and time-stamped events to be logged to disk for accounting purposes or to ensure data integrity, and they enable a degree of automation for the CICS region.

### Security services

A CICS region provides security against unauthorized logon, and protects individual resources (programs, files, and so on) from use by all but certain users. The security management services provide the data needed for the checks, which are performed by CICS internal security, an external security manager, Distributed Computing Environment (DCE) security services, or by some combination of these.

CICS regions can use DCE security services for centralized authentication and authorization of users who want to use transaction processing services. CICS provides its own internal security services that are a more basic alternative to DCE security. It also provides interfaces to special-purpose external security packages to manage all aspects of system security.

### Recovery management services

A CICS region ensures that the business system and its data are always in a consistent state. In the event of an application or system failure, the region can automatically restart itself (if needed) and recover any uncompleted work that was in progress at the time of failure, including any changes to data. If it cannot commit data changes for a task, a region dynamically *backs out* the changes to a point when the system was last in a consistent state.

### User interface

*CICS Terminal management* provides a standard way for applications to communicate with any type of terminal. The CICS region queries the users' devices and determines the optimum characteristics to use for application

output. The region can use models to influence its choice of characteristics and can use terminal definitions to apply specific characteristics to devices.

The typical user interface to a CICS application is through an IBM 3270 terminal—either an actual 3270 terminal or through a terminal emulator package. Other user interfaces that do not present a 3270 interface to the user often still use a 3270 data stream to communicate with the CICS region, allowing the existing CICS applications to be used by new clients.

## CICS application programming interface

CICS provides a rich application programming interface (API) that enables developers to create transaction application programs.

### CICS region application programming

The CICS API is made up of a number of commands for performing operations in CICS regions. These commands are embedded in an application program written in a high-level programming language (such as COBOL, C, C++, PL/I, or Java). The developer simply precedes the CICS command by the phrase EXEC CICS. For example:

```
EXEC CICS READ FILE('ORDER') INTO(RECORD)
```

The program source file is then processed by a precompiler before it is processed by the compiler for the programming language (the COBOL compiler, the C compiler, and so on).

CICS API commands are available to perform the following types of functions:

- File control—reading, writing, and updating files
- Storage control—allocating and freeing memory
- Program control—passing control between CICS programs
- Temporary storage control—reading from and writing to temporary storage queues
- Transient data control—reading from and writing to transient data queues
- Interval control—using timers
- Journal control—writing journals for audit trails, change records, and so forth
- Basic mapping support and terminal support—sending and receiving data from 3270 terminals
- Advanced program-to-program communications—communicating using SNA LU 6.2 communications
- Task control—controlling the CICS internal dispatcher
- Syncpoint and abend support—handling logical units of work

Commands are also provided for security, authentication, batch data exchange, monitoring and diagnostics, and a number of other areas.

CICS provides a Java API that encapsulates these commands. Software developers can use the classes in this API as an alternative to embedding EXEC CICS commands in their programs.

For more information on programming CICS applications, see the following documents:

- *CICS Application Programming Guide*
- *CICS IIOP and Java Programming Guide*

### CICS client application programming

CICS provides two APIs that enable non-CICS applications on a client machine to use the facilities of connected CICS regions. These APIs are common to all IBM CICS Clients.

- The external call interface (ECI) enables a non-CICS application running on the client machine to call a CICS server program running on a CICS region. The client program can call the server program synchronously or asynchronously as a subroutine.
- The external presentation interface (EPI) enables client applications to start and converse with legacy 3270 CICS applications running on CICS regions. The CICS application sends and receives 3270 data streams to and from the client application as though it is conversing with a 3270 terminal. The client application captures these data streams and, typically, displays them with a presentation product such as a graphical user interface. The legacy application itself does not need to be altered.

ECI and EPI application programs that run on CICS clients can be written in COBOL, C, or C++. Programs that do not make operating-system-specific calls are portable between the CICS client products. Client application programs can use both the ECI and the EPI. To use the ECI and EPI, programs must be linked to the appropriate ECI and EPI libraries.

### CICS Internet application programming

Developing Internet applications that use CICS is an extension to normal programming for the Internet, and uses the standard interfaces of Web servers. The most commonly used interface is the Common Gateway Interface (CGI), which is a way of invoking programs from a Web server and returning any output from the programs to the Web server. By using programs invoked by a Web server, you can run CICS transactions from a Web browser, dynamically create Hypertext Markup Language (HTML) pages in response to user input, and issue Structured Query Language (SQL) queries to relational database managers.

An Internet application using the CICS Internet gateway generally contains the following:

- An HTML form, which presents a user with a Web page for entering data and sends user input to the CICS Internet gateway whose name is encoded in the form. This form can be developed by using standard Web development tools and processes.
- The CICS Internet gateway, which is a CGI script called to convert between HTML and 3270 data streams. You can also develop your own CGI scripts, using interpretive languages like REXX or compiled languages like C.
- A program to be invoked on a CICS region. The program can be developed by using the standard CICS application development facilities.

Besides forms, you can use environment variables and command-line arguments to pass data to the CICS Internet gateway.

### Support for relational databases

CICS supports the use of a number of relational databases. These databases can be used to store the information used by CICS applications, which can include embedded Structured Query Language (SQL) statements. CICS supports the XOpen X/A specification, which means that databases that also support this specification can fully participate in CICS LUWs using a full two-phase commit process if needed.

CICS provides support for the following relational database management systems:

- IBM Universal Database (UDB) (DB2)
- Oracle
- Informix
- Sybase
- Microsoft SQL Server

### Queue services

Queues are sequential storage facilities that are global resources within either a single CICS region or a system of interconnected CICS regions. That is, queues, like files and databases, are not associated with a particular task. Any task can read, write or delete queues, and the pointers associated with a queue are shared across all tasks.

Two types of queues are provided by CICS:

- CICS transient data queue services provide a generalized queueing facility. Data can be queued (stored) for subsequent internal or external processing. Applications can:
  - Write data to a transient data queue.
  - Read data from a transient data queue.

  - Delete an intrapartition transient data queue.
- CICS temporary storage queue services provide the application programmer with the ability to store data in temporary storage queues. These queues can be located either in main storage, or in auxiliary storage on a direct-access storage device. Data stored in a temporary storage queue is known as temporary data. Temporary storage queues are efficient, but they must be created and processed entirely within CICS. However, they can be defined dynamically by an application and do not have to be defined to CICS before the application uses them. Applications can:
  - Write data to a temporary storage queue.
  - Update data in a temporary storage queue.
  - Read data from a temporary storage queue.
  - Delete a temporary storage queue.

Although these names imply impermanence, CICS queues are permanent storage. Except for temporary storage queues kept in main storage, CICS queues persist across executions of CICS, unless explicitly discarded in a cold start. Persistent queues are stored by the CICS file manager—either the Encina Structured File Server (SFS) or DB2.

#### User exits

A *user exit* is a place in a CICS module at which CICS can transfer control to a program that you have written (a user exit program). CICS resumes control when your exit program has finished its work. You can use user exits to extend and customize the function of your CICS system according to your own requirements. User exits provide a powerful way for you to control the operation of your CICS system.

## CICS intersystem communication

In a multiple system environment, CICS regions can communicate with other regions to provide users of the local region with services on remote systems and offer services in the local region to users on remote systems. Both data and applications can be shared.

#### CICS intercommunication facilities

The CICS intercommunication facilities simplify the operation of distributed systems. In general, this support extends the standard CICS facilities (such as reading and writing to files and queues) so that applications or users can use resources situated on remote systems without needing to know where the resources are located. The CICS intercommunication facilities are:

**Distributed program link (DPL)**
: DPL extends the use of the EXEC CICS LINK command to allow a CICS application program to link to a program that resides on a different CICS system. In doing so, the first program (the one initiating the DPL request) passes control to the second program.

Conceptually, DPL has many similarities to the remote procedure calls (RPCs) and remote method invocation (RMI) used by other WebSphere components.

**Function shipping**
Function shipping allows an application program to access files, transient data queues, and temporary storage queues belonging to another CICS system.

**Transaction routing**
Transaction routing allows you to execute a transaction on a remote system. The transaction is able to display information on your terminal as if it were running on your local system.

**Asynchronous processing**
Asynchronous processing extends the EXEC CICS START command to allow an application to initiate a transaction to run on another CICS system. As with standard EXEC CICS START calls, the transaction requested in the START command runs independently of the application issuing the START command.

**Distributed transaction processing (DTP)**
DTP uses additional EXEC CICS commands that allow two applications running on different systems to pass information between themselves. These EXEC CICS commands map to the Logical Unit (LU) 6.2 mapped conversation verbs defined in IBM System Network Architecture (SNA). DTP can be used to communicate with non-CICS applications that use the advanced program-to-program communications (APPC) protocol.

TXSeries intercommunication is based on the SNA LU 6.2 protocol, often referred to as APPC. CICS regions and Encina Peer to Peer Communications (PPC) applications can communicate across SNA with any system that supports APPC; for example, IBM mainframe-based CICS and APPC workstations.

CICS and the Encina Monitor support all three synchronization levels defined by SNA, across both SNA and TCP/IP networks. TXSeries can use CICS local SNA support and the Encina PPC Gateway Server. Both methods support all the CICS intercommunication facilities to other CICS regions, and DTP is supported to non-CICS region (such as Encina). Also, CICS can use local SNA support to communicate with IBM CICS Clients.

For more information on CICS intercommunication facilities, see the *CICS Intercommunication Guide*. See "Peer-to-Peer Communications (PPC) Services" on page 86 for more information about the PPC Gateway server.

### Communicating with users

Users communicate with the CICS region through clients. Clients are typically products dedicated to communicating with servers and providing interfaces to users and their application programs. Clients run on a range of platforms, for example, laptop computers and Open Systems workstations.

Users can communicate with CICS through the following:

- IBM CICS Clients, which also enable you to access CICS regions from the Internet and Lotus Notes.
- Telnet clients with 3270 emulation capability.
- Local 3270 terminals attached directly to CICS regions.

Other devices for communicating with users can be connected to CICS Clients. For example, an automatic teller machine (ATM) can be connected to a client to provide its user interface to CICS. Similarly, a printer attached to a client can be used for output from CICS.

Figure 25 illustrates the various communications methods used by CICS clients.

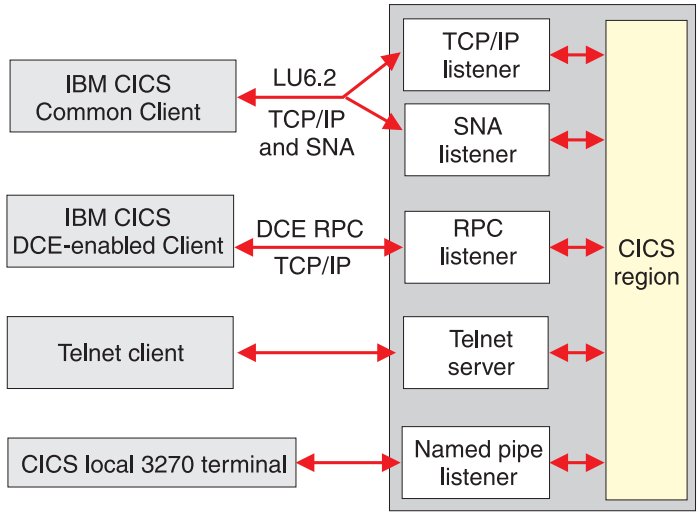


*Figure 25. Communication between CICS clients and a CICS region*

A CICS client can communicate with multiple CICS regions. A client initialization file determines the parameters for client operation and identifies the associated regions and protocols used for communication.

### Communications gateways

CICS supports the following communications gateways:

**CICS Transaction Gateway**

The CICS Transaction Gateway enables Java applications and Web browsers to access TXSeries. The gateway resides on the same machine as the Web server, and can be used by a Java-enabled Web browser or a network computer (NC). The CICS Transaction Gateway provides an interface between a Web server and a CICS client and, through the client, the rest of the CICS transaction processing environment. It also provides an API that enables conversations between an application or applet in Java and a transactional application running in a CICS region. The CICS Transaction Gateway runs on the Windows NT, AIX, Solaris and OS/2 operating systems.

**CICS link for Lotus Notes**

CICS link for Lotus Notes is a gateway that enables Lotus Notes applications to access data managed within the TXSeries environment. The gateway resides on the same machine as the Lotus Notes server and can be used by any Lotus Notes application that can connect to that server. The gateway provides an interface between a Lotus Notes server and a CICS client and, through the client, to the rest of the CICS environment. To use CICS link for Lotus Notes, you need a Lotus Notes Server and client. You can get these through the Lotus Domino Server or you can use your existing Lotus Notes software (of the required version). The CICS link for Lotus Notes runs on the Windows NT and OS/2 operating systems.

## CICS administration

Systems administration for CICS consists of configuring the CICS environment so that CICS regions can be started, monitoring running regions, shutting regions down, and recovering from problems. Administering CICS involves procedures that affect other components such as the Structured File Server (SFS), DB2, and DCE.

The administrative tool used to configure and manage CICS depends on the operating system you are using. For example, you can use the Administration Utility for CICS on Windows NT or the System Management Interface Tool (SMIT) for CICS on AIX. The tools simplify and automate the administrative procedures. You can also use other tools, such as CICS commands and transactions.

The CICS administration tools are designed to manage the CICS environment on one machine. To use them, you log into the machine as a systems administrator, then invoke the tool that you want to use. To manage the CICS environment on several machines, you can use standard techniques to log into each machine remotely and use the tools on those machines. For example, you can use one machine as a single point of control, with sessions set up to run tools on other machines. You can control access to the administration tools by controlling access to this machine.

For more information on administering CICS applications, see the following documents:

- *CICS Administration Guide for Open Systems*
- *CICS Administration Guide for Windows Systems*
- *CICS Problem Determination Guide*

## CICS workload management

CICS Workload Management (WLM) is a stand-alone utility that optimizes the distribution of processing tasks in a CICS environment with two or more regions that can process work requests. WLM is available only on the AIX and Solaris platforms and must be installed separately.

WLM distributes the incoming work requests over the systems that can process the work. The combined machine resources are now available to each application. This provides several benefits:

- Application availability is improved because applications can be accessed from multiple locations.
- Available processing power is optimized because a distribution algorithm is used to route tasks.
- Scalability is improved because additional regions can be created and easily added to the WLM configuration.
- Servers can be shut down and maintained transparently because requests are automatically routed to healthy servers.
- The cost of hardware upgrades is reduced because work requests are distributed over all available servers in the WLM configuration.

For more information, see *Using CICS Workload Management*.

# Part 2. Using WebSphere Application Server

This section describes an example application that illustrates some of the capabilities of WebSphere Application Server. The sample application is an e-business Web site that is created by using members of the WebSphere Application Server product family.

The following topics are discussed:



The sample application can be accessed from the WebSphere Application Server samples page at www.ibm.com/software/webservers/samples.

# Chapter 8. Overview of the sample application

This section introduces the sample application and discusses its goals.

## Sample application scenario: Online banking

The scenario for the sample application centers around a bank's plan to create a corporate Web site. The goals for the site are to establish a World Wide Web presence and enable customers to do online banking.

The bank plans to make the following information available to its customers:

- Account balances
- Transaction histories of accounts
- Company information, such as the bank's mission statement, a list of branches and their hours, and the bank's current promotions
- Site information, such as a site map and other navigation aids

The bank also plans to allow customers to perform the following tasks through the Web site:

- View account balances
- View transaction histories of accounts
- Transfer funds between accounts
- View other types of information on the site by navigating through the site's pages.

The site must meet various performance requirements — for example, the number of users that can be logged onto the site concurrently and the total number of users permitted. It must also meet performance requirements for transactions, such as the maximum duration of a transaction and the timeout period for a typical transaction.

To understand how customers use their site, the bank has defined a set of use cases. They illustrate site flow and exceptions, and can identify areas that might need redesign. An example of a use case follows.

1. The customer logs onto the Web site by entering a valid user ID and password.
2. The customer gets a list of accounts associated with that user ID.
3. The customer views the transaction history for one of the accounts.

Finally, the site must interact smoothly with the information systems currently in place at the bank. For example, it must be able to access account information from an existing customer database and communicate with legacy transaction-monitoring systems. This requirement affects the site's architecture. If customer data can be easily and reliably accessed (for example, in a DB2 database), the site can be designed to directly manipulate data. However, if data must be accessed through existing mechanisms like CICS transactions, the site must be designed to use these legacy transactions.

## Goals of the sample application

The sample application illustrates how WebSphere Application Server can be used to implement an electronic commerce system.

- It shows how the individual parts of WebSphere Application Server work together. The different implementations of the sample application use most of the WebSphere tools and features. The application was created by using WebSphere Studio, VisualAge for Java, and the VisualAge Component Development Toolkit. It makes use of a Web server, Web site, servlets, JavaBeans components, and JavaServer Pages (JSP). The business logic is implemented with distributed objects such as enterprise beans. The various parts of the sample application run on a Web server, the WebSphere Advanced Application Server, and Enterprise Application Server TXSeries.
- It shows how a business system can be implemented by using the Advanced Application Server and the Enterprise Application Server. The different implementations of the sample application show how enterprise beans can be used with both the Advanced Application Server and TXSeries Encina++.
- It shows the development process and design trade-offs involved in creating an application that uses WebSphere Application Server. The sample illustrates the following software development concepts:
  - Customer-driven design and implementation. The sample application's architecture is derived directly from how bank customers intend to use it.
  - Incremental application design. The application is created as a set of related components, each of which can be designed and implemented independently. The application can be built on a component-by-component basis from its individual parts.
  - Reusability. The application is designed so that its components can be reused between different implementations. For example, the same Web site, servlets, and JSP pages are used for all versions of the business logic.
  - The model-view-controller (MVC) architecture, which serves as a paradigm for designing Web applications that isolate business processing from the Web site front end.

- It illustrates access beans, which are a WebSphere Application Server extension to the Enterprise JavaBeans (EJB) specification.
- It provides example code that can be used for designing business systems. Although the sample application is relatively simple, its basic design and code can be modified for use in more complex applications.

# Chapter 9. Sample application design

This chapter describes the high-level architecture of the sample application. It discusses the following topics:



Each of these sections applies to all implementations of the sample application, except where noted. For more information on the different ways that the sample application is implemented, see "Chapter 10. Implementing the sample application" on page 119.

## Application design

The sample application consists of the following components:

- A Web site that acts as the customer interface
- Servlets that coordinate the business processing
- WebCommands, which are JavaBeans components that encapsulate the interface between the servlets and the business logic
- Objects that implement the business logic and perform the actual business processing
- JavaServer Pages (JSP) that display the results of the business processing
- A database that handles persistent storage of customer, account, and transaction information

Figure 26 on page 114 shows how the sample application is designed.

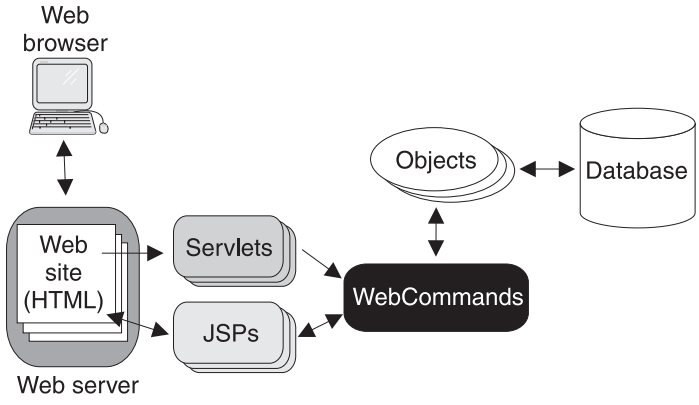


*Figure 26. The sample application*

## Client/server relationship

The sample application is an example of client/server design. The client is a customer's Web browser. The customer loads the application Web site into the browser. From the browser, the customer can interact with the sample application by requesting that banking tasks such as transfers be performed and information about accounts be displayed.

The server is the sample application. It processes client requests and returns the information to be displayed in the client browser. The Web server is the intermediary between the client browser and the application server on which the business logic runs. It hosts the Web site, handles HTTP requests, and so forth. The application server handles all information processing and hosts the servlets, JSP pages, and business logic.

## Model-view-controller architecture

The sample application implements a Model-View-Controller (MVC) architecture. The MVC architecture of the sample application is implemented by using various WebSphere Application Server features:

**Model**

Defines the application's business logic and other internal functions. The model performs the banking tasks defined in "Sample application scenario: Online banking" on page 109. It also makes permanent updates to customer data by accessing a database. The model is implemented as objects that encapsulate processing tasks and represent permanent entities such as accounts, customers, and transaction records.

**View** Defines the client user interface and displays the results of operations to the client. The view is implemented by the Web site and the JSP pages. The Web site provides a user interface for the sample application that enables the client to interact with the application. The JSP pages are used to dynamically generate HTML containing the results of client operations (transfers, information requests, and so forth). They work with the Web site to display the results to the client.

**Controller**
Defines how the client provides input to the application by interacting with the model and view. The controller is implemented by the servlets and WebCommands (which are JavaBeans components). The servlets receive client requests from the Web site (view) and pass them on to the business logic (model) for processing. They also receive the results of the business processing and pass them back to the JSP pages (view) for display. The WebCommands act as the interface between the servlets and the business logic, simplifying their interactions and allowing them to be developed independently.

## Object model

As described earlier, the business logic is encapsulated into objects, each serving a different function in the sample application. Figure 27 shows the objects that make up the business logic and how they interact.

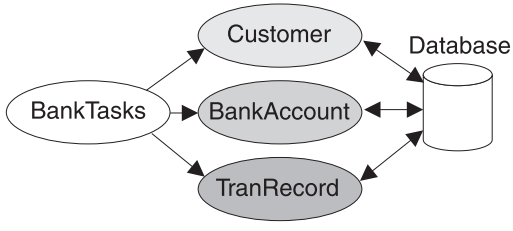


*Figure 27. Object model*

The objects perform the following functions in the sample application:

**BankTasks**
Looks up customer, account, and transaction information and makes transfers between accounts. The BankTasks object acts as the front end to the other objects. It is the only object that contains methods that can be accessed (although indirectly) by the customer. Its methods manipulate the Customer, BankAccount, and TranRecord objects to perform the tasks requested by the customer. For instance, to perform a transfer, the BankTasks method looks up the Account objects specified in the request. It then withdraws the amount of the transfer from the account that is the source of the transfer, and deposits the

amount into the account that receives the transfer. It does not perform the transaction if the amount is less than zero or is more than the current balance of the source account.

**Customer**
Represents a bank customer and is associated with information about a customer in the database. The Customer object contains a method for retrieving information about a customer (such as the customer ID) from the database. Since a customer can have more than one account, a Customer object can be associated with multiple BankAccount objects.

**BankAccount**
Represents a bank account. (The sample application does not differentiate between different types of accounts; in a real application, there might be CheckingAccount, SavingsAccount, and objects representing other types of accounts.) The BankAccount object is associated with information about an account (such as the account number and balance) in a database. It contains methods for retrieving account information and depositing and withdrawing money in the account, which update the account balance in the database. These methods are not accessible to the customer; instead, they are used by the BankTasks object while it is performing transfers. Because an account is owned by a single customer, a BankAccount object can be associated with only one Customer object. However, because an account may have many records of transactions associated with it, a BankAccount object may be associated with multiple TranRecord objects.

**TranRecord**
This object records transactions for a bank account and is associated with a transaction in the database. It contains methods for retrieving information on completed transactions from the database and writing information about new transactions to the database. Because an account can have many transactions, multiple TranRecord objects can be associated with an Account object.

These objects are implemented in the sample application as enterprise beans. See "Chapter 10. Implementing the sample application" on page 119 for details.

## Data model

The sample application is backed by a DB2 database for persistent storage. Customers, accounts, and transaction records are associated with specific rows in the database. The business logic retrieves this information from the database in response to queries. It can also update account balances and transaction records in response to requests for balance transfers.

# Chapter 10. Implementing the sample application

The sample application features a Web site whose business logic is encapsulated by a set of commands implemented in JavaBeans components. These commands make calls to implementation-specific code that actually performs the business functions (accessing account data, performing transfers, and so forth). In all implementations of the sample application, the Web site is served by the IBM HTTP Server. The Advanced Application Server supports the servlets and JavaServer Pages (JSP) that serve the Web site.

The business logic of the sample application is implemented in two different ways using the Advanced and Enterprise Application Servers:

- By enterprise beans deployed in the Advanced Application Server, using entity beans with container-managed persistence (CMP) and a DB2 database as persistent storage. This implementation is described in "Advanced Edition implementation with enterprise beans".
- By enterprise beans deployed in the Advanced Application Server, using entity beans with bean-managed persistence (BMP) and the TXSeries Encina++ bridge server to communicate with an Encina++ Monitor application server that uses a DB2 database as persistent storage. This implementation is described in "Enterprise Edition implementation with enterprise beans and TXSeries Encina++" on page 121.

"Sample application platforms" on page 123 discusses how the sample application is implemented on various operating systems and hardware platforms.

Each implementation has its benefits and drawbacks. The implementation you select for your organization depends on your business requirements, your legacy systems, and which edition of WebSphere Application Server you decide to use.

## Advanced Edition implementation with enterprise beans

Figure 28 on page 120 shows how the sample application's business logic is implemented by using enterprise beans deployed in the Advanced Application Server. Only the business logic design is shown. (The overall design of the sample application is shown in Figure 26 on page 114.)

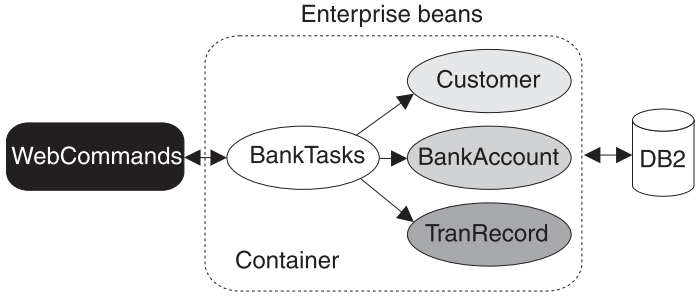


*Figure 28. Enterprise bean implementation in the Advanced Application Server*

The WebCommand implementation calls BankTasks, a session bean that contains the business logic necessary to manipulate the entity beans Customer, BankAccount, and TranRecord. For example, the BankTasks session bean has a method called transfer that transfers money between two accounts represented by BankAccount beans. The transfer method transactionally handles debiting and crediting money between two instances of a BankAccount bean. It also instantiates a TranRecord bean for each account to document the transfer.

Persistence is handled by the container in which the enterprise beans are deployed. The container (in response to requests by the entity beans) retrieves account information from the DB2 database and writes the updated information back to the database. The enterprise bean developer does not need to worry about the persistence model that is used when the entity beans with CMP are deployed. If the persistence model changes later, the entity beans can be deployed to a new container without changing the application's business logic.

The container also manages transactions for the enterprise beans. Transactions associated with enterprise beans that are deployed with a Transaction characteristic of TX_REQUIRED are automatically scoped by the container. The client programmer and enterprise bean programmer do not need to include any transaction-related coding in the enterprise beans. Instead, the container automatically manages the context of the transaction, enabling the appropriate resources to participate in it. (In this case, the resource is the DB2 database.) The container also manages the life cycles of the objects involved in the transaction. For more information about transaction characteristics, see "Transaction services" on page 59.

The entity beans Customer, TranRecord, and BankAccount are all wrapped with access beans to reduce the number of remote requests for multiple attributes from the objects.

Access beans are an extension of the EJB programming model. They can be created by using the Access Bean Builder in VisualAge for Java. All types of enterprise beans can use access beans.

An access bean is designed to allow an enterprise bean to be used like a local JavaBeans component. It serves as a wrapper that hides an enterprise bean's home and remote interfaces. This simplifies the enterprise bean's use. An access bean also maintains a local copy of the bean's attributes, reducing the number of remote requests and improving performance.

## Enterprise Edition implementation with enterprise beans and TXSeries Encina++

Figure 29 shows how the sample application's business logic is implemented by using enterprise beans and TXSeries Encina++. Only the business logic design is shown. (The overall design of the sample application is shown in Figure 26 on page 114.)
The sample application uses the WebSphere Advanced to Encina

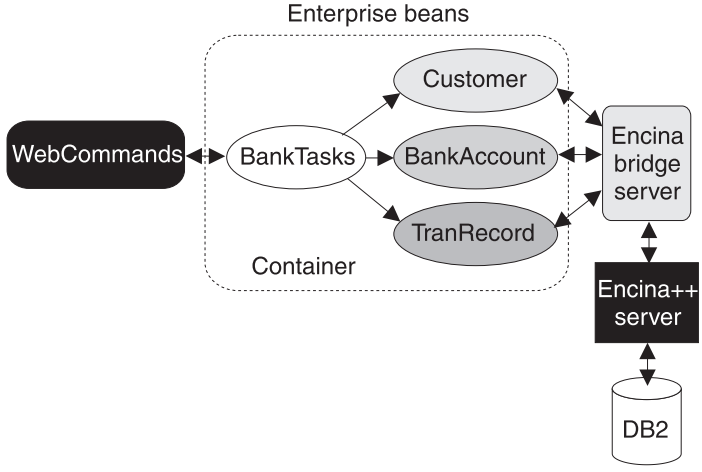


*Figure 29. Enterprise bean implementation with Encina++*

Interoperability functionality to communicate with an Encina++ server that manages persistence. A bridge server enables Java applications developed in WebSphere Application Server to communicate with Encina and Encina++ servers. The bridge server exports interfaces whose methods correspond to the transactional interface definition language (TIDL) interfaces in an Encina++ application. Through these interfaces, the enterprise beans can access an Encina++ application. The **wstidl** command generates the files required to connect Java applications to Encina and Encina++ applications.

The enterprise bean implementation is similar to the Advanced Application Server implementation described in "Advanced Edition implementation with

enterprise beans" on page 119. However, Customer, BankAccount and TranRecord are entity beans with BMP (not entity beans with CMP). The entity beans refer all requests for accessing or updating persistent data to JavaBeans components that are generated by using the **wstidl** command.

The JavaBeans components reside on the bridge server, whose files are also generated by the **wstidl** command. The JavaBeans components pass on the requests through the Encina bridge server to the Encina++ application. Depending on the operation, the Encina++ application server then queries or transactionally updates the DB2 database.

The interfaces that are shared between the entity beans and the Encina++ server are the methods for accessing and updating persistent data. These shared interfaces are generated from a series of TIDL files. The **wstidl** command then generates JavaBeans components and a set of files that are used to create the Encina bridge server.

Using WebSphere Advanced to Encina Interoperability offers several advantages:

- New Java-based applications can communicate with existing Encina and Encina++ servers.
- Transactions can begin in a Java application and continue through Encina to access back-end resources.
- It uses standard IIOP and EJB interfaces.
- The only manual step in creating the client and server files is writing TIDL files to represent the interfaces that are shared between the Encina or Encina++ application and the Java application. The **wstidl** command automatically generates the files needed to implement these interfaces and create the bridge server.

However, there are currently several drawbacks to using the Encina Java connectivity functionality:

- A TIDL file can contain only one interface. (This is actually a limitation of the Distributed Computing Environment (DCE), which is required for Encina.) However, multiple TIDL files can be created for an application.
- IIOP connections to the bridge server do not have workload management.
- There is no security between the bridge server and WebSphere Application Server.

## Sample application platforms

The sample application is available on the following platforms:

- AIX
- Solaris
- Windows NT

It is supported for the same versions of these operating systems supported by the WebSphere Application Server software.

# Chapter 11. Technical details of the sample application

This chapter describes the technical details of each component of the sample application. It includes the following topics:



## Web site

The Web site is created by using WebSphere Studio and runs on the IBM HTTP Server. Its purpose is to interact with the bank's customers, who access it through a browser client. It enables customers to access information and perform banking tasks.

The Web site serves as a front end for displaying information produced by servlets and JavaServer Pages (JSP). The site includes both static HTML files and HTML that is dynamically generated by servlets and JSP pages. Servlets are associated with Web site commands that perform business processing or dynamically build an HTML page. The servlet calls a JavaBeans component in a command style format. The JavaBeans component is then passed to a JSP page for generating formatted HTML data.

### Web site design

The overall design of the Web site is based on the model-view-controller (MVC) paradigm described in "Model-view-controller architecture" on page 114. WebSphere Studio's JavaBeans wizard generates a site that follows this design. The typical site usage pattern is as follows:

1. A Web site visitor goes to (requests) the input page.
2. The visitor enters data and submits the form, which calls the servlet.
3. The servlet uses the JSP file and the JavaBeans component generated by the servlet to dynamically create the output page.
4. The servlet returns the output page to the browser.

WebSphere Studio's JavaBeans wizard creates the following files:

- An HTML file (*.html) for the input page. This page is used to gather user input and invoke the servlet.

- A servlet (*.class and *.java files) to call the JavaBeans component and execute the methods specified by the user's action.
- A servlet configuration file (*.servlet), which is processed when the servlet calls the JavaBeans component.
- A JSP page (*.jsp) that contains the HTML and JSP tags, including formatting for the variable fields. This file is used by the servlet and the JavaBeans component to dynamically create the output HTML file that is returned to the requesting browser.

**Note:** The input HTML file, the class files, the servlet file and the .jsp file are all publishable. The Java files are not.

In the sample application, the client also selects which implementation of the business logic is to be used. This information is used by the servlet to communicate with the proper implementation of the WebCommands. See "Servlets" for a more detailed discussion.

### Client validation and back-end processing

Client-side validation with JavaScript functions is useful for preliminary validation of customer input. It reduces the need for remote requests to identify input errors. The sample application Web site uses JavaScript functions to verify that input fields for the customer ID and the amount of a transfer contain non-blank numeric fields with positive values. An example of this type of client-side validation can be found in the login page, which takes a customer ID as input.

All of the back-end processing for the Web site is encapsulated in a set of JavaBean classes called WebCommands. These JavaBeans components are set up in a command-style format where each component (or WebCommand) performs one set of business processing operations. The command object has a group of attributes that define the command's parameters, a method to perform the command, and another set of attributes that are queried for the result of the command. For more information on how WebCommands are implemented, see "WebCommands" on page 131.

## Servlets

Servlets perform several different functions in the sample application. They collect information that is entered by the client through the Web site, then send the collected information to the WebCommands (and ultimately the business logic) for processing. They receive the results and forward the information to the JSP page for display.

The sample application servlets are created by using the WebSphere Studio JavaBean wizard. The JavaBean wizard generates the basic servlet code, which is then modified to add things like headers and format colors.

The same servlets are used with all implementations of the sample application. They are designed to perform the following tasks:

1. Gather input parameters from the Web site.
2. Load a WebCommand bean and set its input parameters.
3. Process the command bean.
4. Pass the command bean to the JSP page to display the results to the client. Another JSP page receives and handles any servlet errors.

The version of the WebCommands that is loaded by a servlet depends on which business logic implementation is being used for processing. The client selects the business logic implementation on the Web site. This information is used to select the correct version of the WebCommands for the servlet.

A utility loads a specific implementation of a command class based on a setting in the client's Httpsession. When a servlet needs to call a command, it first invokes the utility class to load the correct implementation of the command. It then invokes the command as indicated by the client request. All exceptions that are specific to a particular implementation of the command are mapped into user exceptions. This mapping encapsulates the implementation of the commands under the command itself.

The servlet uses a utility to request a generic command bean. A command bean is an abstract WebCommand whose subclasses implement specialized methods for accessing different versions of the business logic. The utility uses the information in the Httpsession and the name of the generic command bean to instantiate a specific subclass of the command bean and pass it back to the servlet. The servlet then invokes the methods on it (the interfaces on all of the subclasses are the same as those on the generic command bean).

For example, Figure 30 on page 128 shows a code excerpt from the utility class. It illustrates how the utility class uses a value from the Httpsession to call the appropriate version of a WebCommand. (The utility class was added in WebSphere Studio to the basic servlet generated by the Servlet wizard.)

```
private static String[] subClass = {"EJB","EJB","JBO","ENC"};
...
public  Commands.CMD loadCommand(HttpSession session, String className)
            throws Commands.LoginRequired {
    int sample = 0;
    sample = Integer.valueOf((String)session.getValue("sample")).intValue();
    Commands.CMD obj =
         (Commands.CMD)java.beans.Beans.instantiate(getClass().getClassLoader(),
         "Commands." + subClass[sample] + "." + className + "_CMD");
    obj.setSession(session);
    return obj;
}
```

*Figure 30. Code example: Servlet utility class*

For example, if the customer loads the enterprise bean implementation that runs on the Advanced Application Server, a version of the command object that is specifically designed to work with that implementation is loaded (as shown in Figure 31.

```
public void performTask(HttpServletRequest request, HttpServletResponse response)
{
...
HttpSession session = request.getSession(true);
// Instantiate the beans and store them so they can be accessed by the called page
    Commands.ShowAccounts_CMD showAccounts_CMD = null;
    showAccounts_CMD = (Commands.ShowAccounts_CMD)
        Util.instance().loadCommand(session, "ShowAccounts");
        "ShowAccounts");
   String customerId = (String)session.getValue("customerId");
// Setting the command bean in the HTTP request
    setRequestAttribute("showAccounts_CMD", showAccounts_CMD, request);
// Initialize the input value of the bean: the customerId property from the parameters
    showAccounts_CMD.setCustomerId(customerId);
// Call the perform method to execute the business logic.
    showAccounts_CMD.perform();

// Call the JSP output page on successful completion.
// If the output page is not passed as part of the URL, the default page is called.
    callPage(getPageNameFromRequest(request), request, response);
  catch(Throwable theException)
  {
// call the error JSP page if an error condition occurs
   handleError(request, response, theException);
   }
}
```

*Figure 31. Code example: Loading an enterprise bean-specific command bean*

The JSP code that references this part of the servlet is shown in Figure 32 on page 130.

## JavaServer Pages

JavaServer Pages (JSP) are used to display the results of client requests. They dynamically generate HTML that can be displayed in the client browser.

The JSP pages that are used in the sample application are created by using the WebSphere Studio JSP wizard and comply with version 1.0 of the JSP specification. Information is retrieved from the command bean by using a jsp:UseBean tag. This tag defines the reference to the command bean; the reference to the bean is read as the request is passed to the JSP page. Later, the properties of the command bean can be directly accessed from the bean itself.

Figure 32 on page 130 shows how the jsp:UseBean tag is used to retrieve the properties of the command bean. The key parts of the code example are in bold font, and show how the JSP page gets information from the command bean.

```
<jsp:UseBean id="showAccounts_CMD" scope="request" type="Commands.ShowAccounts_CMD">
  <CENTER>
...
<!--METADATA type="DynamicData" startspan
--><%
try {
   // throws an exception if empty
  java.lang.String _p0 = showAccounts_CMD.getAccountId(0); .
  java.lang.String _p0_0 = showAccounts_CMD.getAccountType(0);
  java.lang.String _p0_1 = showAccounts_CMD.getBalance(0); %>
  <TABLE border="1" width="600">
    <CAPTION><B><FONT size="+1">Accounts</FONT></B></CAPTION>
    <TBODY>
      <TR>
        <TH>Id</TH>
        <TH>Type</TH>
        <TH>Balance</TH>
        <TH>History</TH>
      </TR><%
      for (int _i0 = 0; ; ) { %>
        <TR>
          <TD align="center" class="TBL_ODD"><%= _p0 %></TD>
          <TD align="center" class="TBL_EVEN"><%= _p0_0 %></TD>
          <TD align="right" class="TBL_ODD">$<%= _p0_1 %></TD>
          <TD align="center" class="TBL_EVEN">
              <FORM action="/servlet/Commands.ShowHistory" method="POST"
                      target="_self" name="history<%= _i0 %>">
                <A href="javascript:invoke('history<%= _i0 %>');"
                      onmouseout="if(hover)mout('history',
                      <%= _i0 %>)" onmouseover="if(hover)mover('history',<%= _i0 %>)">
                   <IMG border="0" height="50" name="history<%= _i0 %>"
                      src="file:///D:/Studio/Projects/WSFamily/WSFamily/history_o.gif"
                      width="50">
                   <INPUT name="accountId" type="hidden" value="<%= _p0 %>">
                </A>
             </FORM>
          </TD>
        </TR><%
        _i0++;
        try {
          _p0 = showAccounts_CMD.getAccountId(_i0);
          _p0_0 = showAccounts_CMD.getAccountType(_i0);
          _p0_1 = showAccounts_CMD.getBalance(_i0);
        }
        catch (java.lang.ArrayIndexOutOfBoundsException _e0) {
          break;
        }
      } %>
    </TBODY>
  </TABLE><%
}
catch (java.lang.ArrayIndexOutOfBoundsException _e0) {
} %><!--METADATA type="DynamicData" endspan-->
```

*Figure 32. Code example: Using the <BEAN> tag*

The jsp:UseBean tag references the JavaBeans component showAccounts_CMD that is contained in the HTTP request. This is indicated by the value of the jsp:UseBean tag parameter `scope="request"`.

## WebCommands

WebCommands are JavaBeans components that encapsulate interactions with the application's business logic. They provide a standardized way of handing off information for processing. Instead of directly invoking business logic methods, a servlet invokes the WebCommands, which then execute the appropriate methods on the components. This process allows the developers of the Web site, servlets, and business logic to work independently. The WebCommands are created manually by using a text editor.

### WebCommand structure

Figure 33 on page 132 shows how the WebCommands are structured.

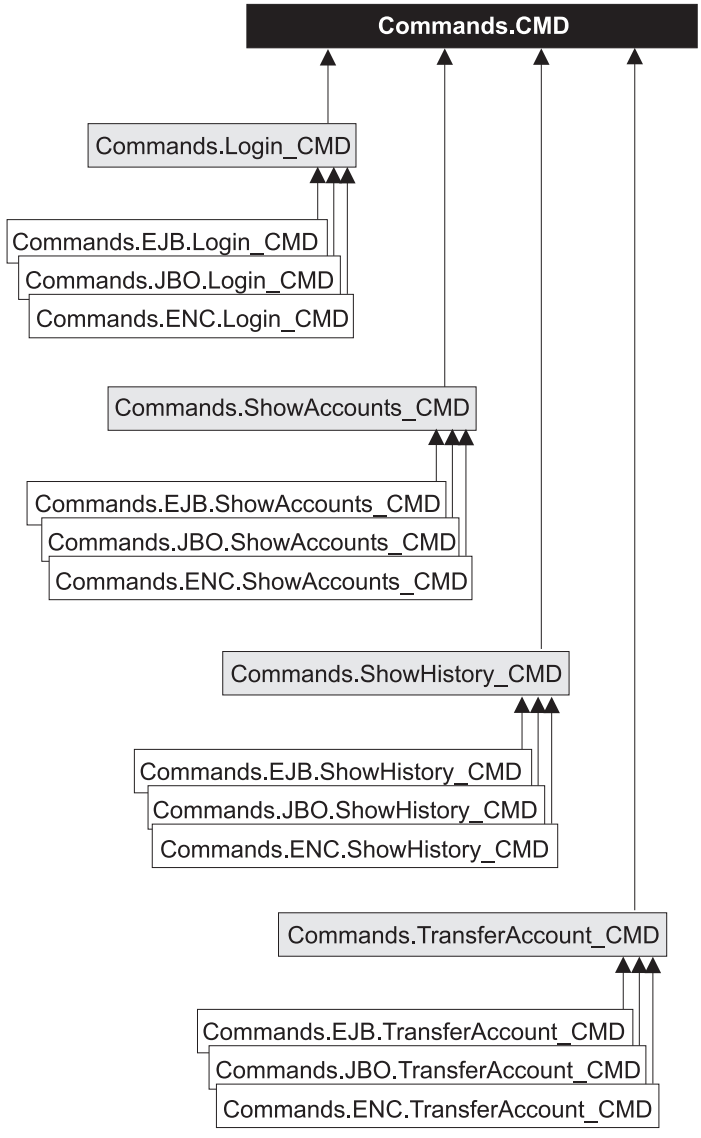


*Figure 33. WebCommand inheritance structure*

All of the WebCommands are derived from the base JavaBeans component, Commands.CMD. This component includes a method called perform that communicates with the business logic. Each type of WebCommand executes a particular task, such as showing a customer's accounts (Commands.ShowAccounts_CMD) or transferring funds between accounts (Commands.TransferAccount_CMD). The WebCommands add the attributes

necessary for input and output for their particular tasks. The subclasses of each WebCommand have different implementations of the perform method for each version of the business logic.

**Commands.EJB.***command***_CMD**
: Used with the enterprise bean implementation of the business logic in the Advanced Application Server.

**Commands.ENC.***command***_CMD**
: Used with the enterprise bean and Encina++ implementation of the business logic.

For example, Commands.EJB.ShowHistory_CMD retrieves an account's transaction history from the Advanced Application Server enterprise bean implementation of the business logic. The business logic-specific WebCommand subclasses use the input attributes of the parent class. They set the output attributes on the parent class as part of the implementation of the perform method. The servlet loads the appropriate WebCommand for the business logic implementation, as described in "Servlets" on page 126.

## Interaction with access beans

The `perform` methods of the WebCommands act as clients to the business logic implementation on the EJB server. The actual business processing takes place on the server. However, the WebCommands do not directly access the enterprise beans or managed objects. To minimize the number of remote method calls and simplify looking up remote methods, the WebCommands interact with either access beans or CopyHelpers.

Figure 34 on page 134 shows a code excerpt from the `perform` method on Commands.EJB.ShowAccounts_CMD. This WebCommand is used with the enterprise bean implementation in the Advanced Application Server. It shows the accounts owned by a customer. The customer ID serves as input; the list of accounts is returned as output.

```
public void perform() throws CustNotFoundException, java.lang.Exception {
  btab = new BankTasksAccessBean();
  if (nameServiceTypeName() != null)
     btab.setInit_NameServiceTypeName(nameServiceTypeName());
  if (nameServiceURLName() != null)
     btab.setInit_NameServiceURLName(nameServiceURLName());
  try {baabt = btab.lookupAccounts(customerId);
  }
  catch (javax.ejb.FinderException fe) {
       throw new CustNotFoundException("id " + customerId
              + " does not exist in our records....");
  } finally {
    btab = null;
  }
}
```

*Figure 34. Code example: Using access beans with WebCommands*

## Enterprise beans (Advanced Edition)

Enterprise beans handle back-end processing for the Advanced Application Server version of the sample application.

The business logic for the Web site resides in enterprise beans that run on an application server. The EJB server can be configured to do the following:

- Run in the same process as the servlet engine
- Run in another process on the same machine
- Run in another process on a different machine in the environment

The business logic is accessed by calling methods on stateless session beans. These stateless session beans scope the unit of work and manipulate the entity beans accordingly. The entity beans update data in the DB2 database and handle some of the business processing.

The customer does not directly access the business logic from the Web site. Instead, a customer request is passed to a servlet, which uses the appropriate set of WebCommands to access the business logic. See “WebCommands” on page 131 for details.

### Session bean implementation

One stateless session bean, BankTasks, contains most of the business logic which the customer requires. It has the following methods:

- lookupCustomer, which returns information about a customer.
- lookupAccounts, which returns a list of a customer’s accounts.
- lookupHistory, which returns the transaction history of a given account.

- deposit, which credits funds to an account and updates its transaction history. This method is not available on the Web site.
- withdraw, which debits funds from an account and updates its transaction history. This method is not available on the Web site.
- transfer, which transfers funds between accounts and updates their transaction history.

The session bean is deployed with the attribute TX_REQUIRED (transaction required) into a container. When the session bean is invoked, the container starts a new transaction, processes the customer request, and then commits the transaction when processing is finished.

See "Session beans" on page 38 for more information on this type of enterprise bean.

### Entity bean implementation

Three entity beans — Customer, BankAccount, and TranRecord — represent the different entities in the bank. The Customer and TranRecord entity beans update rows in the underlying database. They do not contain any additional business logic. The BankAccount entity bean also updates rows in the underlying database. However, it contains methods for depositing and withdrawing money from the account. It throws errors in the case of zero or negative amounts and insufficient funds available for withdrawals.

**Note:** The difference between the withdraw method in the session bean BankTasks and the withdraw method in the entity bean BankAccount is that the session bean also generates transaction records of the changes to the accounts via the TranRecords bean.

See "Entity beans" on page 41 for more information on this type of enterprise bean.

### Access bean implementation

All of the enterprise beans are wrapped with access beans. Access beans simplify the programming model for the user by concealing an enterprise bean's home and remote interfaces, allowing the bean to be treated as a local JavaBeans component. For example, the WebCommands in the sample application do not directly access the enterprise beans. Instead, they use access beans to simplify their interactions with the enterprise beans. Three different types of access beans — wrapped bean, copy helper, and rowset — are used in the enterprise bean implementations of the sample application.

The session bean, BankTasks, uses a wrapped bean. This type of access bean is used with a session bean instance. The following code example shows how a client can access methods on a session bean that uses a wrapped bean access bean.

```
BankTasksAccessBean bt = new BankTasksAccessBean();
CustomerAccessBean cab = bt.lookupCustomer("1234");
System.out.println("Customer title is " + cab.getTitle());
System.out.println("Customer first name is " + cab.getFirstName());
System.out.println("Customer last name is " + cab.getLastName());
```

The entity Beans, Customer, TranRecord, and BankAccount all use CopyHelper access beans. This type of access bean maintains a local copy of the enterprise bean, reducing the number of remote requests when attributes are retrieved from the object. Figure 35 shows how a client can get information from an entity bean that is wrapped with a CopyHelper access bean.

The BankAccount and TranRecord objects also use a rowset access bean. This

```
public BankAccountAccessBean lookupAccount(String accountId)
throws javax.naming.NamingException, java.rmi.RemoteException, FinderException {
  try {
       BankAccountAccessBean baab = new BankAccountAccessBean();
       baab.setInitKey_accountId(accountId);
       baab.refreshCopyHelper();
       return baab;
      } catch (CreateException ce) {
        throw new java.rmi.RemoteException(ce.getMessage());
      }
}
```

*Figure 35. Code example: CopyHelper wrapper on an entity bean*

type of access bean allows you to work with multiple instances of an entity bean without having to instantiate them individually. It also maintains local copies of the enterprise beans. The access bean returns indexed results to make it easier for a JSP page to display the returned data. Figure 36 shows how a JSP client can loop through a rowset access bean.

```
BankAccountAccessBeanTable baabt = bt.lookupAccounts("1234");
   for (int i=0; i < baabt.numberOfRows(); i++) {
       BankAccountAccessBean baab = (BankAccountAccessBean)
              baabt.getBankAccountAccessBean(i);
       System.out.println("Balance is " + baab.getBalance());
       System.out.println("Account Id is " +
              ((BankAccountKey)baab.__getKey()).accountId);
       System.out.println("Account Type is " + baab.getAccountType());
}
```

*Figure 36. Code example: Using a rowset access bean*

For more information on access beans, see "Access beans" on page 45. The VisualAge for Java documentation describes how to create and use access beans.

### Associations between enterprise beans

Associations define the relationships between enterprise beans. There are several one-to-many (1:m) associations between the entity beans in the sample application. A customer can have many bank accounts; therefore, a Customer entity bean can be associated with many BankAccount beans. Similarly, each bank account has multiple transaction records, one for each transaction made on that account; a BankAccount bean can therefore have many TranRecord beans associated with it.

In the sample applications, these associations are hand-defined to be very portable. (Alternatively, the EJB tools in VisualAge for Java can be used to define associations between CMP entity beans.) A method looks up the home interface for an associated entity bean once by using the Java Network Directory Interface (JNDI). It then stores that information for subsequent lookup requests.

In the example where the Customer bean is associated with many BankAccounts beans, the Customer entity bean has two additional methods, getBankAccounts and getBankAccountHome. These methods are used as follows:

- The getBankAccountHome method does the JNDI lookup and returns a reference to the home for the BankAccount bean.
- The getBankAccounts method returns an enumeration of individual instances of BankAccount objects. It does so by calling a custom finder method on the BankAccount Home interface and passing in the primary key information from the Customer bean. For example,

```
findBankAccountsByCustomerId(((CustomerKey)
     entityContext.getPrimaryKey()).customerId)
```

For more information about associations between entity beans, see "Association" on page 48.

### Deployment

The enterprise beans are deployed into a container on the EJB server. The container manages transactions and handles all interaction with the server and the persistence mechanism.

## Enterprise beans, the Encina bridge server, and Encina++

The Encina++ implementation of the sample application (described in "Enterprise Edition implementation with enterprise beans and TXSeries Encina++" on page 121) consists of enterprise beans, an Encina bridge server, and an Encina++ server. These components and their interfaces are described in this section.

### Enterprise beans

The enterprise beans in the Enterprise Application Server/Encina++ implementation of the sample application are similar to the ones in the Advanced Application Server implementation of the sample application. Both implementations use the following:

- A session bean, BankTasks, which contains the business logic.
- A set of three entity beans, Customer, BankAccount, and TranRecord, which represent the entities involved in banking operations.
- Access beans that wrap each enterprise bean.

See "Enterprise beans (Advanced Edition)" on page 134 for more information on how the enterprise beans and access beans are implemented.

The main difference between the entity bean implementations in the sample application is in how they manage persistence. The Encina++ enterprise bean implementation uses entity beans with BMP; the Advanced Edition enterprise bean implementation uses entity beans with CMP.

The Encina++ implementation of persistence is shown in Figure 37 .
The entity beans refer requests for access to persistent data to JavaBeans

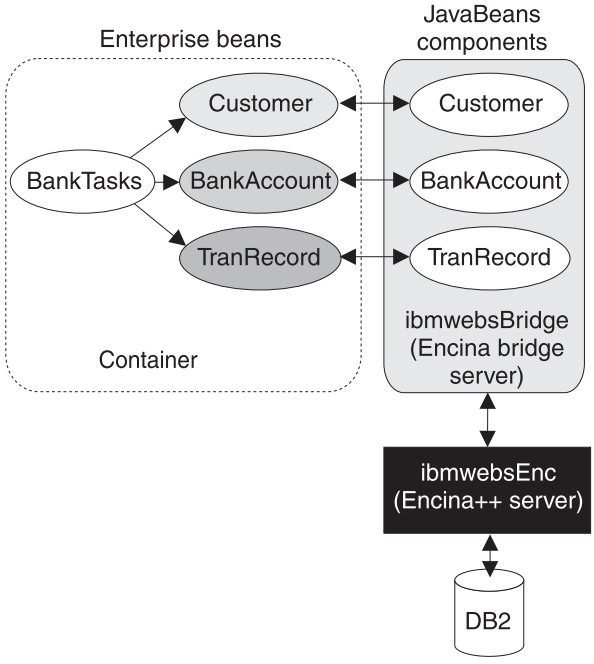


*Figure 37. Managing persistence*

components that reside on the Encina bridge server. The JavaBeans components contain methods that correspond to the operations available on the Encina++ server. The bridge server passes on the persistence requests to an Encina++ server, which transactionally updates a DB2 database.

The entity bean methods correspond to methods in the Encina++ application. These methods are defined in a Transactional Interface Definition Language (TIDL) file. The JavaBeans components and the files that make up the Encina bridge server are generated by the **wstidl** command.

## Defining interfaces by using wstidl

The business logic methods encapsulated in the enterprise beans are first written in TIDL. (In general, the TIDL files originally created for an Encina or Encina++ application are used to define the corresponding interfaces in the Java application.) From the interfaces defined in the TIDL file, the **wstidl** command generates the following:

- A CORBA IDL file with home and bridge object interfaces. The home interface creates and returns an object, using specific create methods that

correspond to Encina++ binding types. The bridge object interface and methods correspond to the TIDL interfaces and operations.

- C++ home and bridge object implementations (the C++ bridge layer). This layer communicates with the Encina++ application.
- JavaBeans components (Java bridge layer), which encapsulate the client-side use of the CORBA home and bridge object interfaces. The entity beans refer persistence requests to these components. (The **wstidl** command can optionally generate an enterprise bean implementation to serve as the client side of the Java bridge layer.)
- Additional Java classes for handling the out parameters and return value of operations and application exceptions.
- A main function for the bridge server.

The entity bean interfaces are manually mapped to corresponding TIDL files. A TIDL file is associated with each entity bean, because a TIDL file can contain only one set of interfaces. Table 3 lists the TIDL input files that are used in the sample application.

*Table 3. wstidl files for entity beans*

| **Entity bean** | **TIDL file** |
|---|---|
| BankAccount | ibmwebsAcctIf.tidl |
| Customer | ibmwebsCustIf.tidl |
| TranRecord | ibmwebsTranIf.tidl |

In addition to the TIDL files, an Interface Definition Language (IDL) file, common.idl, defines the data structures for the interfaces. It is included in the TIDL files listed in Table 3.

Figure 38 on page 141 shows a TIDL file, ibmwebsTranIf.tidl, that contains the interface definition for the TranRecord entity bean.

```
/* WebSphere Family Sample TIDL file
ibmwebsTranInterface */
[
  uuid(c7cc49d0-572f-11d3-9da4-0008c7b2e356),
  version(1.0)
]
interface ibmwebsTranIf
{
  import "common.idl";
 /* get TransactionRecord given transaction id */
 /* this should cover all get TransactionRecord EJB get methods */
 [nontransactional] void getTranRec
 (
   [in] estring_t tranId,
   [out] transactionRecord_t *txRecord
 );
 /* get all transactions of all banking accounts of the
  * specified customer using customer id */
 [nontransactional] void getAllTR
 (
   [in] estring_t customerId,
   [out] transactionRecordArray_t *arrayP
 );
 /* create new transaction record if txRecord.id == null
  * otherwise update other fields */
 /* this should cover all TransactionRecord EJB set and add methods */
 [transactional] void updateTR
 (
   [in, out] transactionRecord_t *txRecord
 );
 /* remove transaction given tranId */
 [transactional] void removeTR
 (
   [in] estring_t tranId
 );
}
```

*Figure 38. Code example: TIDL file containing interfaces for TranRecord entity bean*

This TIDL file defines four interfaces that are used to communicate with the Encina++ server:

- getTranRec, which retrieves a transaction record from the database
- getAllTR, which retrieves records of all transactions associated with a customer's accounts
- updateTR, which creates a new transaction record or updates an existing record. Notice that this method executes within the context of a transaction (indicated by the [transactional] specification in the TIDL file).
- removeTR, which deletes a transaction. This method also executes within the context of a transaction.

Each of these interfaces corresponds to a method defined on the TranRecord entity bean and the Encina++ server. The TranRecord entity bean implements these interfaces. It refers requests to obtain or update persistent data to the JavaBeans component that was generated from the TIDL file. The Encina bridge server passes the request to the Encina++ application server, which executes the method specified in the TIDL file. Java serializable local classes are used to avoid remote calls where only one TIDL call is needed.

The following **wstidl** command is used to generate the files for the bridge server:

```
wstidl -Ic:\opt\encina\include -main \
  -module CustomerIf -javabean  \
  -pkgPrefix com.ibm.ibmwebs.ibmwebsCustIf.tidl
```

Each of the TIDL files is compiled with this command; substitute the appropriate file name and module. The -javabean option tells **wstidl** to generate JavaBeans components as the client layer. Because only one main function needs to be generated for the bridge server, use the -main option only for the first TIDL file.

The common.idl file is run through the IDL compiler to generate the corresponding header file. The generated header file is linked in with the other generated files to produce the bridge server.

### Encina bridge server

The Encina bridge server, ibmwebsBridge, is actually a specialized Monitor application server (MAS) that acts as a bridge between the Java-based sample application and an Encina++/DCE application. It is built from the files generated by the **wstidl** command. The generated main function includes the header file for the TIDL interface and initializes the interface. Since the bridge server exports multiple interfaces, you must edit the file to include the header files for all three TIDL interfaces and the common.idl interface. You must also initialize each interface.

### Encina++ server

The Encina++ server, ibmwebsEnc, is created by using the Encina Server wizard, which is part of the TXSeries application development kit. It incorporates all three TIDL files defined for the sample application (see Table 3 on page 140).

The ibmwebsEnc server transactionally accesses a DB2 database. Data requests from the entity beans are passed through the JavaBeans components and the bridge server to the Encina++ server. The Encina++ server then queries and transactionally updates the database by using embedded Structured Query Language (SQL) statements. Each SQL statement (some of which have multiple joins) roughly corresponds to an entity bean method.

### Managing transactions

Transactions are initiated by the entity beans and are managed by the container. The container works with the Encina bridge server to coordinate transactions that involve the Encina++ application server. Transactions pass through the bridge server to access the back-end resources controlled by the Encina++ application server. Transactional methods on the Encina++ server are identified in the TIDL files.

### Deployment

The enterprise beans are deployed with a deployment descriptor of TX_MANDATORY. (See “Managing transactions” for more information on how the application manages transactions.) Because the implementation uses entity beans with BMP, the container does not handle persistence.

# Chapter 12. Extending the sample application

This section describes some ways in which the sample application can be extended. Its purpose is to present ideas on how other IBM and WebSphere products can be used with WebSphere Application Server. It is not intended to be an exhaustive list of extensions or a detailed guide for implementing these extensions.

The following topics are included:

- "Connecting Java applications to MQSeries"
- "Using the WebSphere Edge Server with the sample application" on page 147

## Connecting Java applications to MQSeries

MQSeries can be used to extend the capabilities of the Advanced Application Server by enabling Java applications (including servlets, applets, and applications based on the JavaBeans and EJB component architectures) to exchange data in the form of messages. This gives these applications the ability to communicate with MQSeries applications on a wide variety of platforms.

To enable the sample application to communicate with MQSeries, you can modify the sample's servlets or enterprise beans to connect to an MQSeries queue manager and exchange data in message format. MQSeries supports three Java APIs that can be used to implement messaging:

- The *MQSeries classes for Java* (MQ base Java) *Java Bindings* package (com.ibm.mqbind) provides a full Java API for communicating with MQSeries applications. Java applications can use this API to connect directly to an MQSeries queue manager, which needs to reside on the same host as the application. This API provides a high-speed connection to a queue manager on the local host, but cannot be used to connect to queue managers on different hosts.
- The MQ base Java *Java Client* package (com.ibm.mq) provides a full Java API for communicating through a client connection to an MQSeries server channel. Applications can use this API to communicate with MQSeries queue managers on any host in the network. However, the Java Client API provides a slower connection to a local queue manager than the Java Bindings API because the Java application connects through a channel instead of through a direct connection.
- The *Java Messaging Service* (JMS) API is the standard for implementing messaging applications in the Java programming language. It provides a

portable, vendor-independent messaging API. The MQSeries implementation of the JMS interfaces lies beneath the JMS API. Objects that represent MQSeries queues and queue managers are defined to JMS by using a directory naming service (the MQSeries message service). Applications can use the JMS API to connect to MQSeries queue managers on any host in the network. JMS also provides functionality that is not available in the MQ base Java APIs, including:

- Asynchronous messaging
- Message selectors
- Support for MQSeries Publish/Subscribe messages
- Structured message classes

These three APIs enable Java applications to issue calls and queries to MQSeries (for instance, over the Internet) without requiring MQSeries to be installed on the client machine. They provide a infrastructure for accessing enterprise applications and developing Web-based applications. By placing the MQSeries queue closer to the user in network terms, system loading can be bypassed and responses can be sent more quickly. In addition, MQSeries messaging enables back-end resources to be accessed transactionally.

In a WebSphere Application Server environment, MQSeries can be used to implement an integration server — that is, a server that acts as an integration point for multiple presentation tiers. An integration server enables first tier clients to share the applications and resources on the second and third tiers. It communicates with databases, transaction processing monitors, applications, and back-end data stores to retrieve and manipulate enterprise information before it is processed by other applications and presented to end users. The flexible MQSeries messaging format enables an MQSeries-based integration server to communicate with a wide variety of resources on many different types of systems.

The MQ base Java and JMS APIs are available in MQSeries SupportPac MA88, which can be downloaded from the MQSeries support Web site. For more information on using these APIs, see the following documents:

- *MQSeries Using Java*, which is available from the MQSeries support Web site.
- The JMS specification, which is available from the Sun Microsystems Java Web site, www.java.sun.com.

## Using the WebSphere Edge Server with the sample application

WebSphere Edge Server can be used in conjunction with WebSphere Application Server. The Edge Server acts as a front-end to a WebSphere Application Server system, as shown in Figure 39.

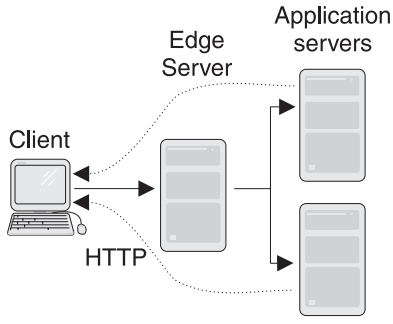


*Figure 39. WebSphere Edge Server and Application Server*

It can monitor server loading, verify application and database integrity, create server clusters, provide failover support, and offload static content from the Web server.

Both components of the Edge Server can be used with WebSphere applications such as the sample application.

- The Caching Proxy component intercepts data requests from end users, retrieves the requested information, and delivers it back to the end users. Most commonly, the requests are for documents stored on Web servers and delivered via HTTP. The Caching Proxy can be used to cache the sample's static Web pages, enabling them to load more swiftly.
- The Network Dispatcher intercepts data requests and forwards them to the server machine that is currently best able to fill the requests. It distributes incoming requests among a set of machines that handle the same type of requests. The Network Dispatcher can be used as a load balancer for the Web server — for instance, if the sample application was replicated over several servers. It can evenly distribute requests over the different machines and provide a reliable connection between the client and the application server.

For more information on using the Edge Server, see the product documentation.

# Notices

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will

be incorporated in new editions of the document. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

**For Component Broker:**
IBM Corporation
Department LZKS
11400 Burnet Road
Austin, TX 78758
U.S.A.

**For TXSeries:**
IBM Corporation
ATTN: Software Licensing
11 Stanwix Street
Pittsburgh, PA 15222
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Trademarks and service marks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States, other countries, or both:

Advanced Peer-to-Peer Networking
AFS
AIX
APPN
AS/400
CICS
CICS OS/2
CICS/400
CICS/6000
CICS/ESA
CICS/MVS
CICS/VSE
CICSPlex
DB2
DCE Encina Lightweight Client
DFS
Encina
IBM
IBM System Application Architecture
IMS
IMS/ESA
Language Environment
MQSeries
MVS/ESA
NetView
Open Class
OS/2
OS/390
OS/400
Parallel Sysplex
PowerPC
RACF
RAMAO
RMF
RISC System/6000
RS/6000
S/390
SAA
SecureWay
TeamConnection
Transarc
TXSeries
VSE/ESA
VTAM
VisualAge
WebSphere

Domino, Lotus, and LotusScript are trademarks or registered trademarks of Lotus Development Corporation in the United States, other countries, or both.

Tivoli is a registered trademark of Tivoli Systems, Inc. in the United States, other countries, or both.

ActiveX, Microsoft, Visual Basic, Visual C++, Visual J++, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Some of this documentation is based on material from Object Management Group bearing the following copyright notices:

Copyright 1995, 1996 AT&T/NCR
Copyright 1995, 1996 BNR Europe Ltd.
Copyright 1991, 1992, 1995, 1996 by Digital Equipment Corporation
Copyright 1996 Gradient Technologies, Inc.
Copyright 1995, 1996 Groupe Bull
Copyright 1995, 1996 Expersoft Corporation
Copyright 1996 FUJITSU LIMITED
Copyright 1996 Genesis Development Corporation
Copyright 1989, 1990, 1991, 1992, 1995, 1996 by Hewlett-Packard Company
Copyright 1991, 1992, 1995, 1996 by HyperDesk Corporation
Copyright 1995, 1996 IBM Corporation
Copyright 1995, 1996 ICL, plc
Copyright 1995, 1996 Ing. C. Olivetti &C.Sp
Copyright 1997 International Computers Limited
Copyright 1995, 1996 IONA Technologies, Ltd.
Copyright 1995, 1996 Itasca Systems, Inc.
Copyright 1991, 1992, 1995, 1996 by NCR Corporation
Copyright 1997 Netscape Communications Corporation
Copyright 1997 Northern Telecom Limited
Copyright 1995, 1996 Novell USG
Copyright 1995, 1996 02 Technolgies
Copyright 1991, 1992, 1995, 1996 by Object Design, Inc.
Copyright 1991, 1992, 1995, 1996 Object Management Group, Inc.
Copyright 1995, 1996 Objectivity, Inc.
Copyright 1995, 1996 Oracle Corporation
Copyright 1995, 1996 Persistence Software


This software contains RSA encryption code.


---



**Java Compatible™ Enterprise Edition**

Other company, product, and service names may be trademarks or service marks of others.

# Index

## F



## H



## I



## J



## L



## M



## N



## O



## P



## Q



## R

## S



## T



## V



## W

## X