WebSphere™ Application Server

Getting Started with WebSphere Application Server

Version 4.0

SC09-4581-00

WebSphere™ Application Server

Getting Started with WebSphere Application Server

Version 4.0

SC09-4581-00

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 105.

First Edition (March 2001)

This edition replaces SC09-4430-01.

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

Contents

Figures

Tables

About this book

This document is intended to help you get started using IBM® WebSphere™ Application Server and familiarize you with the components that make up this product. Although this book focuses on the WebSphere Application Server Enterprise Edition, it also contains information on the WebSphere Application Server Standard Edition and the WebSphere Application Server Advanced Edition.

Who should read this book

This document is intended for use by installers, system administrators, developers, system architects, and other information technology professionals who want to gain an understanding of WebSphere Application Server. A minimal level of familiarity with distributed computing and Web computing is assumed.

Document organization

This document has the following organization:

- "Chapter 1. Introducing the IBM WebSphere family" on page 1 provides a high-level introduction to the IBM e-business strategy and how the WebSphere Application Server and associated products implement this strategy.
- "Chapter 2. Distributed computing and WebSphere Application Server" on page 11 describes distributed computing, transactions, and security.
- "Chapter 3. Overview of component technology" on page 17 describes the various component models and related technologies used in WebSphere Application Server.
- "Chapter 4. Adapting business models to WebSphere Application Server" on page 35 discusses the value that WebSphere Application Server offers to an organization.
- "Chapter 5. Introduction to WebSphere Application Server, Standard and Advanced Editions" on page 39 describes the main components and concepts of the Advanced Application Server and the Standard Application Server, with emphasis on the former.
- "Chapter 6. WebSphere Application Server Enterprise Edition" on page 49 introduces the Enterprise Application Server and associated concepts.
- "Chapter 7. Introduction to Component Broker" on page 55 provides more detailed information on Component Broker.

- "Chapter 8. Introduction to TXSeries" on page 67 provides more detailed information on TXSeries™.
- "Chapter 9. Sample topologies and configurations" on page 79 discusses examples of WebSphere Application Server system configurations.
- "Appendix. The library for WebSphere Application Server" on page 93 provides a complete list of the documentation available with WebSphere Application Server.

Related information

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

- *Building Business Solutions with WebSphere*
- *CICS Application Programming Guide*
- Component Broker *Application Development Tools Guide*
- Component Broker *Planning, Performance, and Installation Guide*
- Component Broker *Programming Guide*
- TXSeries *Planning and Installation Guide*
- *Writing Encina Applications*
- *Writing Encina Applications on Windows Systems*
- *Writing Enterprise Beans in WebSphere*

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.
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.

Table 1. Conventions used in this book (continued)

Convention	Meaning
%	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: 1. Expand the Management Zones folder 2. Expand the management zone named Sample Cell and Work Group Zone 3. Expand the Configurations folder **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.
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.)

Table 1. Conventions used in this book (continued)

Convention	Meaning
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).

Chapter 1. Introducing the IBM WebSphere family

This chapter examines the IBM approach to e-business and discusses how the products in the WebSphere family provide solutions to your e-business challenges. It also looks at some of the other tools available with the WebSphere family and how they fit into IBM's overall approach. The final section of this chapter provides guidance on where to get more information about WebSphere family products.

IBM and e-business

The popularity of the World Wide Web (the Web) among both individuals and businesses has grown rapidly. Although individuals use the Web for many different purposes, businesses use the Web primarily to provide products, services, and information to their customers, suppliers, strategic partners, and employees.

When the first businesses moved onto the Web, it was enough for them to provide a few static Web pages that listed products and services for sale and provided a telephone number or address to order those products and services. Businesses that provided information services (such as software companies) were among the first to enter this new frontier, and they often made information and software products available for downloading.

As new technologies were developed, static Web pages were no longer sufficient. In response, businesses built active Web sites where customers could order products directly, customers and suppliers could communicate with the business, and employees could communicate with each other.

While the Web side of many businesses was changing rapidly, non-Web business information systems also went through major changes as application development spread into distributed systems from mainframe systems. The Open Group's Distributed Computing Environment (DCE) and the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) were two major technologies that provided the infrastructure for these types of systems.

Until recently, Web and non-Web business information systems remained largely detached from one another. The IBM e-business initiative and the WebSphere family have changed that. WebSphere enables businesses to integrate their Web-based systems with their non-Web systems, producing a single enterprise-wide business system.

The WebSphere family: providing e-business solutions

The IBM WebSphere family was designed to help users realize the promise of e-business. It is a set of software products that helps customers develop and manage high-performance Web sites and integrate those Web sites with new or existing non-Web business information systems. It focuses on the following general types of businesses:

- Businesses that want to use the latest technologies to establish a powerful Web presence or upgrade their current Web presence
- Businesses that want to develop distributed, enterprise-wide business systems and applications
- Businesses that want to integrate their Web presence with their existing computer systems and applications

The WebSphere family consists of the WebSphere Application Server and other WebSphere family software that is tightly integrated with the WebSphere Application Server and enhances its performance.

WebSphere Application Server is fully compliant with Sun Microsystem's Java™ 2, Enterprise Edition (J2EE™) standard and supports other common industry standards such as CORBA, XA, secure sockets layer (SSL), Kerberos, Logical Unit (LU) 6.2, and lightweight directory access protocol (LDAP).

WebSphere Application Server: three editions for different customer needs

To enable customers to achieve their e-business goals, WebSphere is available in three editions:

- The WebSphere Application Server *Standard Edition* (also called the Standard Application Server) combines 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. It includes server capabilities for applications built to the Enterprise JavaBeans™ specification from Sun Microsystems. It also enables Web applications to interact with enterprise databases and transaction systems.
- The WebSphere Application Server *Advanced Edition* (also called the Advanced Application Server) builds on the Standard Application Server. It provides support for distributed server environments and includes the underlying services for managing these environments. It also offers closer integration with non-Web business systems.
- The WebSphere Application Server *Enterprise Edition* (also called the Enterprise Application Server) builds on the Advanced Application Server to expand its capabilities to enterprise-wide information systems. It combines TXSeries™, IBM's world-class transactional application environment (consisting of both Encina® and CICS®), with the full

distributed object and business-process integration capabilities of Component Broker. The Enterprise Application Server contains a complete version of the Advanced Application Server.

All three editions are available on the IBM AIX®, Sun Microsystems Solaris, and Microsoft® Windows NT® platforms. WebSphere Standard and Advanced Editions are available on the Linux, IBM AS/400®, and Novell Netware platforms. WebSphere Standard and Advanced editions and the TXSeries component of Enterprise Edition are available on the Hewlett-Packard HP-UX platform. For a complete list of supported platforms, see the WebSphere Application Server web site, www.ibm.com/software/webservers/appserv.

WebSphere Application Server for OS/390® is available in Standard and Enterprise Editions. On this platform, Standard Edition comes with the IBM JDK and WebSphere Site Analyzer. It also supports connections to CICS and IMS™ on the OS/390 platform through the Common Connector Framework. Enterprise Edition consists of the Standard Edition products plus Component Broker for the OS/390.

WebSphere Application Server is primarily a runtime environment on the OS/390 platform. To write applications, you can use the WebSphere development tools available on the Windows and Unix platform, such as VisualAge™ for Java or the Component Broker Object Development Toolkit.

Which edition should you use?

The three editions of WebSphere Application Server are intended to support the needs of different types of customers. Table 2 gives some recommendations about which edition might fit the needs of your organization.

Table 2. Selecting a WebSphere Application Server edition

Edition	Features	Recommended for
Standard	• Single-machine application server • Supports JSP™ pages, servlets, enterprise beans • Offers Web site development tools	Web sites with relatively low traffic, hosted on a single application server; small organizations.
Advanced	• Multimachine application server • Supports JSP pages, servlets, enterprise beans • Offers workload management, enhanced security, Web site and Java development tools, better access to back-end resources	Web sites with moderate to high traffic, hosted on multiple application servers; medium to large organizations.

Table 2. Selecting a WebSphere Application Server edition (continued)

Enterprise	All features of the Advanced Application Server plus Component Broker and TXSeries	Organizations that need robust, enterprise-wide transaction servers that integrate with Web servers.

WebSphere Edge Server

IBM WebSphere Edge Server includes software tools to reduce Web server congestion, increase content availability, and improve Web server performance. It is intended to provide better service to users who access Web-based content over the Internet or a corporate intranet. The name Edge Server indicates that the software usually runs on machines that are close (in a network configuration sense) to the boundary between an enterprise's intranet and the Internet.

The Edge Server can be used in conjunction with WebSphere Application Server to control client access to Web servers. It has two components: the Caching Proxy and the Network Dispatcher. Both are described in this section.

The Caching Proxy

The Caching Proxy component intercepts data requests from end users, retrieves the requested information from content-hosting machines, and delivers it back to the end users. Most commonly, the requests are for documents stored on Web server machines (also called origin servers or content hosts) and delivered via the HyperText Transfer Protocol (HTTP). However, you can configure the Caching Proxy to handle other protocols, such as File Transfer Protocol (FTP).

When retrieving certain types of content, the Caching Proxy stores it in a local cache before delivering it to the requester. The most prominent example of cacheable content is static Web pages (those without portions that are dynamically generated at access time). Caching enables the Caching Proxy to satisfy subsequent requests for the same content directly from the cache, which is much quicker than retrieving it again from the content host.

The Caching Proxy can be useful both when hosting Web-accessible content and when providing Internet access:

- When used by content hosts, the Caching Proxy is installed as a reverse proxy between the Internet and the enterprise's content hosts. It intercepts user requests arriving from the Internet, forwards them to the appropriate content host, caches the returned data, and delivers it to the users across the Internet.
- When used by Internet access providers, the Caching Proxy is installed as a forward proxy between an enterprise's end users and the Internet. It

forwards users' requests to content hosts located across the Internet, caching and delivering the retrieved data to users.

The Caching Proxy was called Web Traffic Express in earlier versions of WebSphere.

Network Dispatcher

The Network Dispatcher also intercepts data requests from end users, but rather than actually retrieving data, it forwards the request to the server machine that is currently best able to fill the request. In other words, it distributes incoming requests among a defined set of machines that service the same type of requests.

The Network Dispatcher can distribute requests to many types of servers, including both HTTP origin servers and Caching Proxy machines. If desired, you can write rules that specify the criteria used by the Network Dispatcher when determining which server can best handle a request.

Like the Caching Proxy, the Network Dispatcher can be useful both when hosting Web-accessible content and when providing Internet access.

- When used by content hosts, the Network Dispatcher is installed between the Internet and the enterprise's backend servers, which can be content hosts, Caching Proxy machines, or mail server machines that service the POP3 or IMAP protocols. The Network Dispatcher acts as the enterprise's single point-of-presence on the Internet, even if the enterprise uses multiple backend servers because of high demand or a large amount of content.
- If the Network Dispatcher's Content Based Routing (CBR) module is installed together with the Caching Proxy, HTTP requests can even be distributed based on URL or other administrator-determined characteristics, eliminating the need to store identical content on all backend servers.
- When used by Internet access providers, the Network Dispatcher is installed between an enterprise's end users and two or more Caching Proxy machines in the enterprise's intranet, to balance the load between them. Balancing loads on multiple Caching Proxy machines provides highly reliable access to the Internet even in the face of high demand. You can also guarantee high availability by installing a backup Network Dispatcher to take over if the primary one fails temporarily.

WebSphere Studio

WebSphere Studio is a suite of tools that brings all aspects of Web site development into a common interface. Content authors, graphic artists, programmers, and Webmasters can all work on the same projects, each having access to the files they need. With the WebSphere Studio, it is easier than ever to cooperatively create, assemble, publish, and maintain dynamic interactive Web applications.

WebSphere Studio is composed of the Workbench, the Page Designer, the Remote Debugger, and wizards, and it comes with companion Web development products. WebSphere Studio enables you to do everything you need to create interactive Web sites that support your advanced business functions, including the following:

- Create Java beans, database queries, and Java servlets by using the Studio wizards. You don't have to be a programmer to generate server-side logic that can add powerful functions to your Web sites. The wizards make it easy to produce the input forms, the output pages, and the Java code that makes it all work. Your Web pages won't just sit there; now they will be able to do real work for your business.
- Group Web site files into projects and folders. You decide what organization makes sense and group your files accordingly. Filters and global search capabilities let you find just the files you need. With familiar objects that behave the way you expect them to, you can add speed and efficiency to your routine tasks.
- Maintain the files individually or in a shared version control system. You can store your files locally, on your own workstation, on other systems in your network, or in a full-function version control system (VCS). You can confidently work in a collaborative manner and know your file integrity is guaranteed.
- Edit and update the files with your preferred tools. You get to choose which editing and viewing programs to use for each file type. When opening a Studio file, you can quickly launch your default selection or you can choose one of your alternative tools.
- Quickly assess file relationships and find broken links. The Relationship view provides a visual representation of how your files link to each other. You can quickly see how many other files link to a particular file, which files have broken links, and which files are not linked at all.
- Publish your Web site during any stage of development. Go directly from site development to site publishing, right within the Studio Workbench. You can publish all or part of a Web site during any stage of development and quickly view and test the results of your work. When the Web site is ready for release, you can easily transfer the files to your final stage and publish them to your production servers.

The Studio Workbench helps you manage and maintain your Web site applications and files. It provides the following capabilities:

- A graphical display of the link relationships between the files in a project.
- The automatic updating of links whenever files change or move.
- The ability to register multiple authoring tools, giving you a choice each time you edit your Web site files.

- The ability to stage your Web site production cycle and publish various stages to different (and to multiple) servers.
- An import wizard that simplifies the transfer of existing site content directly into a Studio project.
- A quick way to archive Web sites or subsites in a single file.
- The ability to easily integrate third-party tools right into the Workbench environment.
- An enhanced team environment with a common view of work in progress, through the integration of popular source control-management software such as IBM VisualAge Team Connection®, Microsoft SourceSafe®, PVCS, Rational ClearCase, and Lotus® Domino™.

The Studio Page Designer provides a visual design environment that enables you to create JavaServer Pages (JSP), Java servlets, and other Java-based Web tools. For example, you can use the visual environment to drag and drop Java beans into JSP applications. The Studio Page Designer can also be used to create DHTML and HTML pages and includes the capability to easily edit and toggle between the HTML or DHTML source and the browser view. This capability is based on the new IBM HomePage Builder product.

The Studio Remote Debugger provides source level debugging of JSP files and Java servlets within the Studio environment. Remote debugging is possible on any machine that contains WebSphere Application Server 3.0 or above.

VisualAge for Java

VisualAge for Java is an integrated development environment that supports the complete cycle of Java program development. VisualAge for Java is tightly integrated with the WebSphere Application Server. This integration enables VisualAge developers to develop, deploy, and test their Java programs without leaving VisualAge. It also provides developers with an environment that helps to manage the complexity common in 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 the Visual Composition Editor, you point and click to do the following:

- Design the user interface for your program
- Specify the behavior of the user interface elements
- Define the relationship between the user interface and the rest of your program

VisualAge for Java generates the Java code to implement what you visually specify in the Visual Composition Editor. In many cases you can design and run complete programs without writing any Java code.

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

VisualAge for Java gives you the programming tools that you need to develop industrial-strength code. Specifically, you can:

- Use the completely integrated visual debugger to examine and update code while it is running
- Build, modify, and use Java beans and build, modify, deploy, and test enterprise beans
- Browse your code at the level of project, package, class, or method

VisualAge for Java also has a code management system that makes it easy for you to maintain multiple version of programs.

How do I get more information about WebSphere Application Server?

The remainder of this document contains descriptions of the three editions of WebSphere Application Server. In addition, you can get more information by reading the documents described in the rest of this section.

All WebSphere documentation is available from the WebSphere Application Server library page at www.software.ibm.com/webservers/appserv/library.html. These documents are available as Hypertext markup language (HTML) and Adobe Portable Document Format (PDF) files.

Installers and system administrators

If you need to install, configure, or manage a version of the WebSphere Application Server, read one or more of the following:

- To learn the basics of installing and configuring the Standard or Advanced Application Servers, see the InfoCenter for the product you wish to install. This online documentation center is designed for first-time users who want to get a simple system up and running quickly. Both versions of the InfoCenter are available from the WebSphere Application Server library page at www.software.ibm.com/webservers/appserv/library.html.
- To learn about managing the Standard and Advanced Application Servers, see the following:
  - The online help available within the WebSphere Administrative Console.
  - The online documentation in the Standard Edition or Advanced Edition InfoCenter.

- To learn about installing, configuring, and managing a system with the Enterprise Application Server, start with the *Planning, Performance, and Installation Guide* for Component Broker or the *Planning and Installation Guide* for TXSeries.

Application developers and system architects

If you need to design business systems or develop applications using a version of the WebSphere Application Server, read one or more of the following documents:

- To learn about planning, developing, and troubleshooting applications in the Standard and Advanced Application Servers, see the Standard Edition and Advanced Edition InfoCenters, which are available from the WebSphere Application Server library page at www.software.ibm.com/webservers/appserv/library.html.
- To learn the basics of developing enterprise beans and related components in compliance with the Sun Microsystems Enterprise JavaBeans specification, start with *Writing Enterprise Beans in WebSphere*. This document provides instructions for developing enterprise beans in both the Advanced Application Server and the Enterprise Application Server.
- To learn about the broader issues involved in designing and developing systems and applications in the WebSphere family, read *Building Business Solutions with WebSphere*. Many of the technologies, issues and topologies mentioned in *Getting Started with WebSphere Application Server* are described in more detail in *Building Business Solutions with WebSphere*.
- To learn about developing applications in Component Broker, start by reading the *Application Development Tools Guide* and then read the Component Broker *Programming Guide*. Programmers on OS/390 should also read *OS/390 Component Broker Programming: Assembling Applications*.
- To learn about developing applications in TXSeries CICS, start by reading the *CICS Application Programming Guide*.
- To learn about developing applications in TXSeries Encina, start by reading *Writing Encina Applications* (for general development) or *Writing Encina Applications on Windows Systems* (for development on Microsoft Windows platforms).

For a complete list of the documentation available with the Enterprise Application Server, see "Appendix. The library for WebSphere Application Server" on page 93.

Chapter 2. Distributed computing and WebSphere Application Server

WebSphere Application Server provides an environment for open distributed computing. Users and processes on a wide variety of platforms can interact by using the facilities provided by WebSphere. Both the Advanced Application Server and the Enterprise Application Server provide a distributed computing environment.

This section provides an overview of the basic concepts of distributed computing, transaction processing and security. If you are familiar with these concepts, you can skip to "Chapter 3. Overview of component technology" on page 17 or "Chapter 4. Adapting business models to WebSphere Application Server" on page 35.

Three-tiered client/server computing

A common way of organizing software to run on distributed systems is to separate functionality into two parts: clients and servers. A *client* is a program that uses services provided by other programs called *servers*. The client makes a request for a service, and a server performs that service. Server functionality often involves some sort of resource management, in which a server synchronizes and manages access to the resource, responding to client requests with either data or status information. Client programs typically handle user interactions and often request data or initiate some data modification on behalf of a user.

For example, a client can provide a form on which a user (a person using a Web browser, for example) can enter orders for a product. The client sends this order information to the server, which checks the product database and performs tasks needed for billing and shipping. A single server is typically used by multiple clients. For example, dozens or hundreds of clients can interact with a handful of servers that control database access.

A common design of client/server systems uses three tiers: a client that interacts with the user, an application server that contains the business logic of the application, and a resource manager that stores data. This approach is shown in Figure 1 on page 12. In this model, the client is isolated from having to know anything about the actual resource manager. If you change the database you are using, the server might have to be modified, but the client does not need to be modified. Because there are usually fewer copies of the server than the client, and because the servers are often in locations that are

easier to update (for example, on central machines rather than on PCs running on users' desks), the update procedure is also simplified. Furthermore, this approach provides additional security. Only the servers, not the clients, need access to the data controlled by the resource manager.

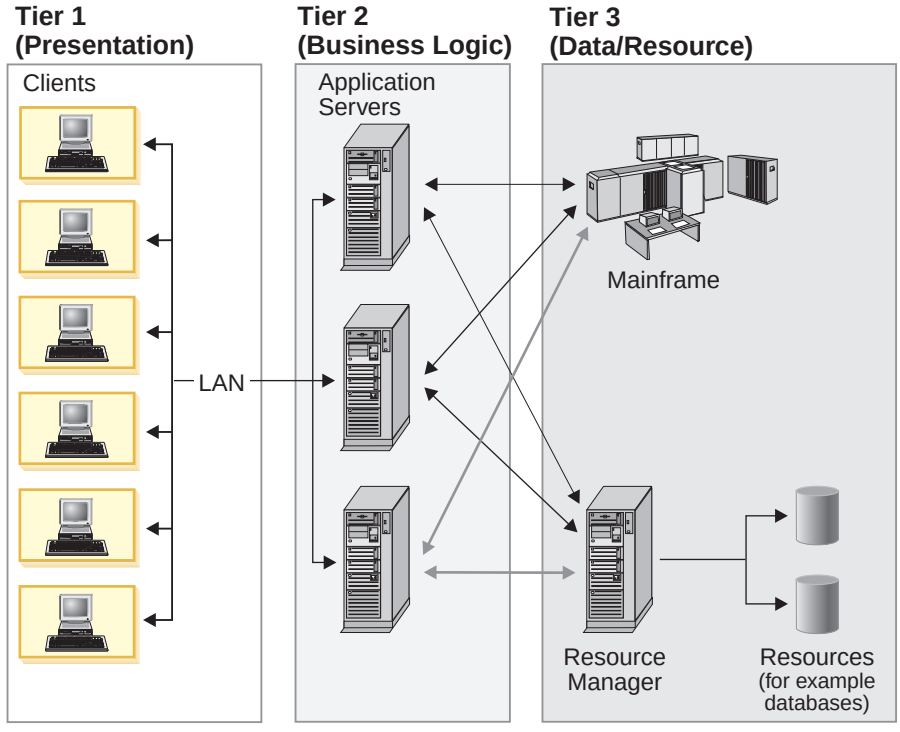


Figure 1. Three-tiered client/server architecture

WebSphere Application Server provides the middle tier in this architecture, allowing clients—applets, Visual Basic® clients, C++ clients, and so on—to interact with data resources (relational databases, MQSeries®, and so on) as well as with existing applications. This architecture is also used by two major components of the Enterprise Application Server: Component Broker and TXSeries.

Transactions: ensuring data consistency and permanence in a distributed environment

A *transaction* is a set of operations that transforms data from one consistent state to another. This set of operations is an indivisible unit of work, and in some contexts, a transaction is referred to as a *logical unit of work* (LUW). A transaction is a tool for distributed systems programming that simplifies failure scenarios.

Transactions provide the *ACID properties*:

- *Atomicity*: A transaction's changes are atomic: either all operations that are part of the transaction happen, or none happen.
- *Consistency*: A transaction moves data between consistent states.
- *Isolation*: Even though transactions can run (or be executed) concurrently, no transaction sees another's work in progress. The transactions appear to run serially.
- *Durability*: After a transaction completes successfully, its changes survive subsequent failures.

As an example, consider a transaction that transfers money from one account to another. Such a transfer involves deducting money from one account and depositing it in another. Withdrawing the money from one account and depositing it in the other account are two parts of an *atomic* transaction: if both parts cannot be completed, neither must happen. If multiple requests are processed against an account at the same time, they must be *isolated* so that only a single transaction can affect the account at one time. If the bank's server fails just after the transfer, the correct balance must still be shown when the system becomes available again: the change must be *durable*. Note that *consistency* is a function of the application; if money is to be transferred from one account to another, the application must subtract the same amount of money from one account that it adds to the other account.

Transactions can be completed in one of two ways: they can commit or roll back. A successful transaction is said to *commit*. An unsuccessful transaction is said to *roll back*. Any data modifications made by a rolled back transaction must be completely undone. In the above example, if money is withdrawn from one account but a failure prevents the money from being deposited in the other account, any changes made to the first account must be completely undone. The next time any source queries the account balance, the correct balance must be shown.

A *distributed transaction* is one that runs in multiple processes, usually on several machines. Each process works for the transaction.

Distributed transactions, like local transactions, must adhere to the ACID properties. However, maintaining these properties is greatly complicated for distributed transactions because a failure can occur in any process, yet even in the event of such a failure, each process must undo any work already done on behalf of the transaction.

A distributed transaction processing system maintains the ACID properties in distributed transactions by using two features:

- *Recoverable processes*: Recoverable processes log their actions and thus can restore earlier states if a failure occurs.

- *A commit protocol*: A commit protocol enables multiple processes to coordinate the committing or aborting of a transaction. The most common commit protocol, and the one used throughout WebSphere Application Server, is the two-phase commit protocol.

Security: ensuring authorized use only

When enterprise computing was handled solely by a few powerful mainframes located in information systems (IS) sites, ensuring that only authorized users obtained access to computing services and information was a fairly straightforward task. In distributed computing systems, where users, application servers, and resource managers can be spread out across the world, securing computing system resources has become a much more complicated task.

Although there are many issues associated with providing security in a distributed computing system, the underlying issues have not changed significantly. A good security service provides two main functions: authentication and authorization.

Authentication takes place when a *principal* (a user or a computer process) initially attempts to gain access to a computing resource. At that point, the security service challenges the principal to prove that the principal is who it claims to be. Human users typically prove who they are by entering their user IDs and passwords; a process normally presents an encrypted key. If the password or key is valid, the security service gives the user a *token* or *ticket* that identifies the principal and indicates that the principal has been authenticated.

After a principal is authenticated, it can then attempt to use any of the resources within the boundaries of the computing system protected by the security service; however, a principal can use a particular computing resource only if it has been authorized to do so. *Authorization* takes place when an authenticated principal requests the use of a resource and the security service determines whether the user has been granted the privilege of using that resource. Typically, authorization is handled by associating access control lists (ACLs) with resources that define which users or processes (or groups of users or processes) are authorized to use the resource. If the principal is authorized, the principal gains access to the resource.

In a distributed computing environment, principals and resources must be mutually suspicious of each other's identity until both have proven that they are who they say they are. This is necessary because a principal can attempt to fake its identity to get access to a resource, and a resource can be a trojan horse, attempting to get valuable information from the principal. To solve this

problem, the security service contains a security server that acts as a *trusted third party*, authenticating principals and resources so that these entities can prove their identities to each other.

Chapter 3. Overview of component technology

Business information systems must be flexible enough to respond to shifts in markets, organizational structures, and business methods. They must be reliable and easily manageable. They must be able to scale up to larger sizes and run on a wide variety of hardware and software platforms. Above all, they must be designed, coded, tested, and put into production quickly.

Component technology is the foundation of the IBM WebSphere product family. It can help an enterprise meet these information technology goals. Component technology provides a common-sense, building-block approach to application development. Applications are built from reusable blocks of software that perform specific functions. By using component technology, businesses can develop applications more quickly and simplify their maintenance.

This section examines the basic concepts of component technology. It includes the following topics:

This chapter is intended for readers who have little or no experience with object-oriented programming and component technology. If you are familiar with these concepts, you can skip to "Chapter 4. Adapting business models to WebSphere Application Server" on page 35.

Objects

Objects are used to model conceptual and physical entities. They are used in user interfaces, embedded systems, and many other venues including complex business systems. The model must capture both the state and behavior of the entity in a software packet, or *object*. The object must know things about the entity and be able to perform the actions associated with the entity. The things that an object knows are referred to as its *state*, *properties*, or *attributes*. The things that an object does are referred to as its *behavior*, *operations*, or *methods*. Collectively, the things that an object knows and does constitute its *interface* to the rest of the application.

The way in which an object implements its interface is an internal matter. Its implementation is hidden, or *encapsulated*, from the rest of the software system. Hiding implementation details — for example, the way that an

object's state is represented internally — prevents the rest of the system from becoming dependent upon them. This makes it possible to change an object's implementation without disrupting the rest of the system.

Business objects contain the data and tasks performed by individual parts of a business system. They enable software developers to design software systems that directly reflect the business information that needs to be delivered and the tasks that need to be performed. As a result, crucial business requirements are less likely to be overlooked. If used properly, business objects can make information systems more manageable and extend their lifetimes.

Business objects do not exist in isolation. Instead, they collaborate to perform the operations that are necessary for an organization to run smoothly. Each object in an object-oriented system has its own set of responsibilities to fulfill. Defining and allocating these responsibilities is the task of the system designer.

Objects also have relationships to each other. These can be transient relationships, where one object learns about another one as part of an incoming request or as the result of a request. Relationships can also be structural, with objects specifically designed to work with each other.

Building objects through composition

Object-oriented systems often use *composition* — the process of building objects from other objects — to implement a business model. Building objects through composition is like building a structure out of blocks. Individual blocks (objects) can be reused in many different contexts. Older blocks can be replaced by newer ones. Blocks can be combined to form larger units. These units can then be used in a variety of ways.

One of the promises of object technology is that the power and simplicity of the building block concept can be used to develop and maintain business applications. An information system can be designed as a set of reusable business objects that grow and evolve with the organization.

Interface versus implementation

One of the most powerful features of object-oriented systems is the notion of object interfaces (described under "Objects" on page 17). An object's interface (or more specifically, the interface provided by its class) specifies certain requirements and guarantees that apply to the object.

The client of an object can be obliged to meet certain preconditions as a basis for using that object's interface. Preconditions can include restrictions on the order in which methods can be called, and requirements for the types of input data that the client provides.

The object that serves the client must perform certain tasks in response to any message that it is sent. The interface stipulates only what is guaranteed to occur, including any exceptions that must be accounted for. It says nothing about how the object will carry out its tasks.

The *implementation* of the interface specifies how the object fulfills its requests. The implementation is always private. Although the implementation of an interface can change, the interface itself is unaffected.

Classes

Object-oriented systems provide a mechanism called a *class* that serves as a template for creating objects. A class describes an object's interface, the implementation details that support the interface, and the specific information that describes the object's state. Each object, in turn, is an *instance* of a class.

Inheritance

At any time, there can be multiple implementations of an interface in a system. Clients are able to interact uniformly and transparently with each implementation of an interface. This concept is unique to object-oriented systems. What makes it possible is the idea of *inheritance*. Inheritance enables classes to share, or inherit, interfaces and implementations from other classes. The class that passes on its attributes and behaviors to other classes is known as the *parent class* or the *base class*. The class that inherits attributes and behaviors is known as the *child class* or the *derived class*. A child class can inherit from one or more parent classes.

Inheritance allows you to design by addition. You can create a common parent class that packages the attributes and behaviors of a family of related types. The child classes inherit the parent's interface, and can add their own attributes and behaviors to it. (Attributes and methods cannot be subtracted from the parent interface.)

Polymorphism

Inheritance allows objects to be used again and again. Reusing an interface is known as *interface inheritance*; reusing the implementation of an interface is known as *implementation inheritance*. Although interfaces can be reused, the implementation of an interface cannot always be reused. For example, the way in which a balance is adjusted can be different for checking accounts and savings accounts. In practice, reusing interface designs is much more important than reusing implementation code. The details of the private implementation do not matter to other objects in the system, but the details of the public interface do.

Polymorphism is a powerful tool for reusing interfaces. It enables objects of different classes related by inheritance to respond in a unique way to the same function call. Although their interfaces all contain the same method, the method's implementation varies from object to object. The method's

implementation in the child class can override its original implementation in the parent class. When the method is invoked, objects derived from each class respond in a way that is defined by their particular implementation.

Polymorphism also makes it possible to reuse client code. A client refers to an object only through its interface. It can communicate with any object that implements that interface. Each object that is contacted by the client responds in a manner appropriate to its own implementation. From the client's perspective, an object can be substituted for any of the others in the system.

Object code can be reused as well. In general, objects with simple interfaces have a better chance of being reused. For instance, the Account object's interface can be designed in a way that enables the object to be used in a variety of contexts, such as queries and reports, settlement and audit processing, and interest calculations for monthly statements.

Achieving reuse often requires many design iterations over the course of several projects. Software developers must place a high priority on designing reusable objects if object-oriented systems are to achieve their potential.

Components

Components are objects with additional capabilities that enable them to function in large-scale information systems. These additional features include the ability to do the following tasks:

- Create and destroy objects
- Provide an object with a location in a distributed network
- Establish a system identity for an object
- Store the state of an object in a resource manager
- Handle the activation and passivation (or paging) of an object
- Map transaction semantics to an object's state transitions
- Coordinate an object's locks with its underlying data store
- Control access to an object
- Search for an object
- Notify an object that an event has occurred
- Transport the state of an object across a distributed network
- Assign a human-readable name to an object

An object that has been transformed into a component can be used in traditional client/server application environments, since it supports distributed processing, remote procedure calls, persistence management, integrated security, transactions, concurrency control, and dynamic query capabilities.

Component models and related technologies

A *component model*, or *component architecture*, specifies the way in which objects can be put together within the software system that surrounds them. It is a model for object interoperability. This section discusses some of the component models and related technologies that can be used to design and implement business applications in WebSphere Application Server. It includes:

Java 2™ Platform Enterprise Edition (J2EE™)

WebSphere Application Server complies with the Java™ 2 Platform, Enterprise Edition (J2EE™) specification. J2EE defines a standard architecture for designing and implementing multi-tiered applications. It is comprised of the following elements:

J2EE application components and runtime environment

The J2EE platform is a standard platform for hosting J2EE applications. Its runtime environment consists of the following:

- Application components, which are the building blocks for creating J2EE-compliant programs. The J2EE programming model defines four types of components that an application must support:
  - Application clients, which are typically Java programs that run on desktop computers.
  - Applets, which run in a Web browser and provide the user interface for J2EE application.
  - Servlets and JSP pages, which run on a Web server and respond to HTTP requests from application clients.
  - Enterprise beans, which run on an EJB server and contain the application's business logic and handle transactions.
- Containers, which provide the runtime support for the application components. They offer services such as transaction management, security, and state management to application components.
- Resource manager drivers, which connect J2EE applications to external resources.
- A database which stores business data.

J2EE application model and development team roles

The J2EE application model defines a standard application model for developing multi-tier, thin client services. Applications are assembled from components such as servlets, JSP pages and enterprise beans, then deployed on the J2EE runtime platform. J2EE application development team roles include:

- Product provider — Implements a J2EE product, including component containers, J2EE platform APIs, and other J2EE features.
- System administrator — Configures and administers deployed J2EE applications and the corresponding computing and network infrastructure.
- Tool provider — Provides tools for developing and packaging application components.
- Application component provider — Designs and implements J2EE application components.
- Application assembler — Assembles application components into a complete J2EE application.
- Deployer — Deploys applications.

J2EE standard services

The J2EE standard services include the following services and protocols:

- HTTP and HTTPS
- Java Transaction API (JTA)
- RMI-IIOP. Both the Java native RMI protocol and the CORBA IIOP protocol are supported.
- JavIDL
- Java Database Connectivity (JDBC™) API
- Java Messaging Service (JMS) API
- Java Naming and Directory Interface™ (JNDI)
- JavaMail™
- JavaBeans™ Activation Framework (JAF)
- Java API for XML Parsing (JAXP)
- J2EE Connector architecture
- Java Authentication and Authorization Service (JAAS)

J2EE compatibility test suite and reference implementation

The J2EE compatibility test suite is a suite of compatibility tests for verifying that an application conforms with the J2EE platform standard.

The J2EE reference implementation demonstrates the capabilities of J2EE and provides an operationa definition of the J2EE platform.

JavaBeans components

A number of programming languages can be used to implement components. However, the Java programming language is well suited for cross-platform component development. Java is an object-oriented language created by Sun Microsystems. It is simpler to use than older object-oriented programming languages like C++.

JavaBeans technology is Sun's component architecture for the Java application environment. It extends the idea of writing once and running anywhere to reusable component development. A *bean* is a reusable software component built using the JavaBeans specification. Visual programming tools can combine beans to create an application.

A bean implements an additional set of interfaces from an ordinary Java object. The interfaces specific to a bean are:

- **Introspection** — Allows a visual programming tool to analyze how a bean works, allowing developers to connect beans.
- **Customization** — Enables developers to customize the appearance and behavior of a bean, using a property sheet provided by a visual programming tool.
- **Events** — Enables beans to communicate by using a notification mechanism.
- **Properties** — Describes the bean's attributes and broadcasts a notification when an attribute changes.
- **Persistence** — Allows developers to customize beans and package them into a Java Archive (JAR) file. Also known as *serialization*.

Figure 2 shows how two JavaBeans components can be combined during the development of a simple applet. When a user presses the **Close** button in this example, an event occurs. The event is communicated to the AccountProxy bean, which invokes a method. A visual Java development tool can be used to link the beans.

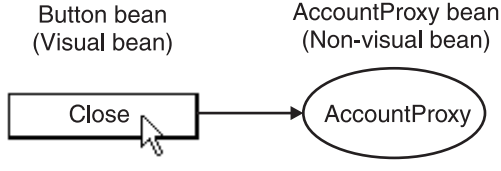


Figure 2. Beans make it possible to program visually

Beans can also interoperate with ActiveX controls. (ActiveX is Microsoft's version of downloadable, interoperable objects.) Bridge software makes it

possible for beans to be deployed in ActiveX environments such as Internet Explorer, Visual Basic®, and Microsoft Word.

Enterprise beans

The *Enterprise JavaBeans* (EJB) specification enables developers to create server-side business components in Java. Enterprise beans are the standard component architecture for building distributed object-oriented business applications in the Java programming language. They are designed to be installed on a server and accessed remotely from a client. Enterprise beans are platform independent and can be deployed in any run-time environment that supports the EJB specification.

This section discusses the following enterprise bean-related topics:

The EJB architecture

Figure 3 illustrates some of the details of the EJB specification.

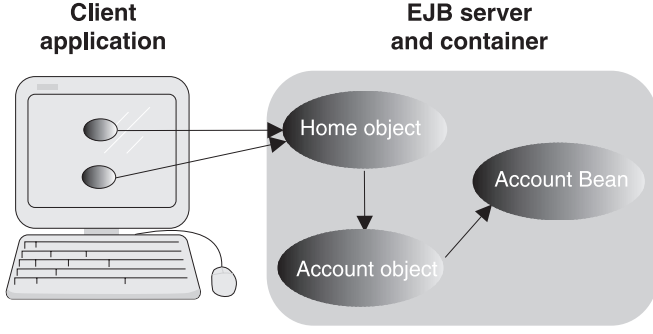


Figure 3. EJB architecture

The client application communicates with a remote object called a *home.* The home object is used to create a business object — for example, an Account object — that is accessed remotely from the client. This object provides all of the attributes and behaviors that pertain to accounts (such as methods for adjusting an account balance and opening an account).

The Account object forwards the requests it receives to an enterprise bean that provides the real business implementation. By intercepting incoming requests, the Account object controls access to the enterprise bean and handles any requirements for executing business methods. For example, the Account object must establish the appropriate transactional context before money can be moved into or out of an account.

On the server side of Figure 3 on page 24, a *container* is the run-time component of an EJB environment. It provides essential services like transactional support, security support, and memory management. The Account object collaborates with the container in order to complete its tasks.

Home collections and containers are native features of WebSphere Application Server Enterprise Edition. They provide simultaneous server support for C++ business objects and enterprise beans. No layered run-time service is required.

Persistence

Enterprise beans exhibit two different types of *persistence*.

- *Session beans* represent tasks and operations. They are transient and do not correspond to data in persistent storage. However, a session bean can maintain state information, such as a list of actions that have been performed during its lifetime. State information that is associated with a session bean is not shared between clients.
- *Entity beans* represent persistent data. Each entity bean is mapped to a data store that is shared with other enterprise beans. For instance, entity beans can correspond to rows in a relational database. In most cases, an entity bean must be accessed in some transactional manner. Instances of an entity bean are unique, and they can be accessed by multiple users. Entity beans do not store state information and can be shared between clients.

Management of the persistent state of an entity bean can be done in two ways:

- In *bean-managed persistence* (BMP), the bean directly accesses persistent storage. Access to persistent storage must be implemented by the software developer.
- In *container-managed persistence* (CMP), the bean relies on its container to provide transparent access to persistent storage. The software developer does not need to explicitly implement access to persistent storage.

Deploying enterprise beans

Enterprise beans are deployed into a container that provides run-time services. A *deployment descriptor* is used to specify how an enterprise bean is managed by its container. Deployment descriptors are set during application assembly or deployment. They define life cycle, persistence, transactions, and security settings for enterprise beans.

In addition, an enterprise bean's *environment properties* allow developers to customize the bean based on the needs of the application. For instance, an environment property might specify the location of a database. By using deployment descriptors and environment properties, you can customize applications without accessing the underlying source code.

Applets and servlets

Java *applets* and *servlets* are complementary technologies to beans and enterprise beans. They are not components themselves, but Java programs tailored to perform specific client-server tasks.

Applets

Applets run in a client machine's Java-capable browser. The business logic in an applet is downloaded from a central World Wide Web server and run on a client, ensuring that the client machine always gets the latest copy of the code. Administration costs are lower than traditional desktop environments, where applications must be updated manually. The downloadable nature of applets also reduces hardware requirements. For these reasons, many companies are deploying Java applets internally.

Applets can be designed by using the standard packages found in the Java2™ Software Development Kit (SDK) or by using the components of the Java Foundation Classes (JFC). They use the Java Virtual Machine (JVM) supplied by a Web browser. The JVM is a run-time environment that interprets compiled Java programs according to the underlying operating system. Because JVMs have been developed for most operating systems, applets can be run on many different operating systems without any special porting requirements.

Applets also enable a business to extend its reach to a larger set of clients. The Internet gives customers a new entry point into an enterprise, opening up opportunities for electronic commerce. Applets can be downloaded over the Internet at any time to enable customers to place orders, retrieve information, and perform many other tasks.

Figure 4 on page 27 shows a scenario for using applets to access a database. A Web server downloads a page of information in *Hypertext Markup Language* (*HTML*) to a Web browser (1). The page includes a command to download a Java applet packaged into a *Java Archive* (JAR) file from the server (2). The applet uses Java's *Remote Method Invocation* (RMI) protocol to communicate with other objects residing on the server (3). These objects then access the database (4).

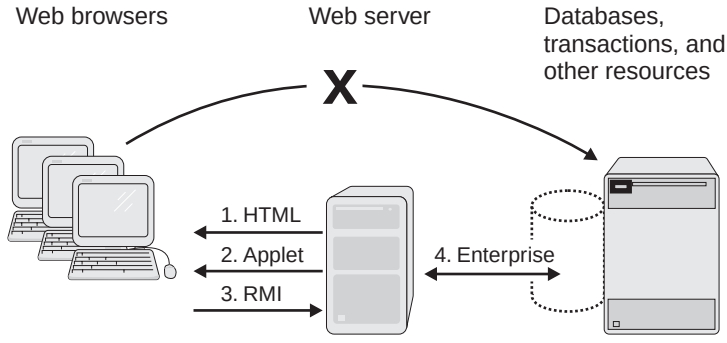


Figure 4. An applet-based business solution

The applet cannot directly access the resource. Java security prevents the applet from communicating directly with the file system on the machine where the browser runs or with other machines than the server.

Applets have two drawbacks:

- Depending on their size and the type of network connection, they can take a considerable amount of time to download.
- They cannot run properly if the browser's JVM does not support the functionality of the Java code in the applet. Although the correct JVM can be attached to the browser dynamically, this does not solve the underlying incompatibility problem and can degrade applet performance.

Servlets

Java servlets are a complementary technology to applets; many applications use both. A servlet is a Java program that runs on a World Wide Web server. A special interface allows it to receive requests from a Web browser, such as a user-initiated submission from an HTML form. Servlets typically access enterprise resources in response to an incoming request, then format new HTML pages dynamically for transmission back to the browser. To simplify HTML generation, servlets can use (but are not limited to) *Java Server Side Includes* (JSSIs) and *JavaServer Page* (JSP) scripting.

Servlets use the Java Servlet API and its associated classes and methods. Servlets can also use Java class packages that extend and add to the Java Servlet API. Like applets, servlets are part of the Java standard and are designed to run across different platforms. However, unlike applets, Java servlets do not use the JVM supported by a browser. Servlets run on a Web server that is controlled from within your enterprise. This makes their behavior more reliable and predictable.

Figure 5 on page 28 shows how to access a database by using only a servlet. The HTML file that is generated by the servlet contains embedded *JavaScript*

commands (1). JavaScript is a scripting language that is used to write simple programs that run on a client machine. JavaScript does not require a JVM and is faster to download than an applet. The browser uses HTTP to communicate directly with the servlet (2). The servlet can then access databases and other resources (3).

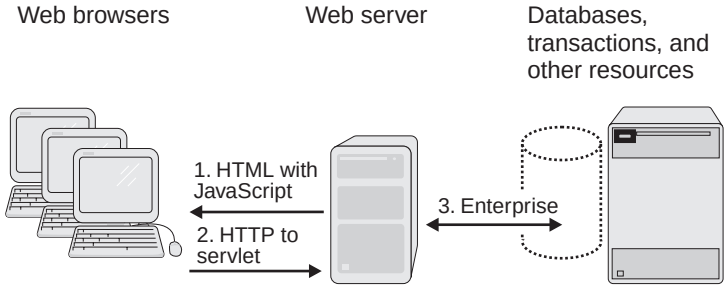


Figure 5. A servlet-based business solution

The servlet-based application retains Internet access and the ability to process logic locally at the client. It reduces the costs of administration and client hardware. Compared to the applet-based application shown in Figure 4 on page 27, the servlet-based application avoids the overhead associated with downloading an applet. It reduces concerns related to the capabilities of the client browser, since it does not require a JVM. It also avoids restrictions for communicating through a firewall.

Unlike the widely-used Common Gateway Interface (CGI) programs, which require an entire process to handle user requests, servlets can handle user requests by using threads. This capability makes servlets much more efficient than CGI programs.

A servlet can be loaded automatically when the Web server is started, or it can be loaded the first time a client requests its services. After being loaded, a servlet continues to run, waiting for additional client requests.

JavaServer Pages

WebSphere Application Server supports a powerful approach to dynamic Web page content: JavaServer Pages (JSP). The JSP function in the Application Server is based on the Sun Microsystems JavaServer Pages Specification.

JSP files are similar in some ways to server-side includes in static HTML because both embed servlet functionality into the Web page. However, in a server-side include, a call to a servlet is embedded within a special servlet tag; in JSP pages, Java servlet code (or other Java code) is embedded directly into the HTML page.

One of the many advantages of JSP pages is that they enable you to effectively separate the HTML coding from the business logic in your Web pages. You can use JSP pages to access reusable components, such as servlets, Java beans, enterprise beans, and Java-based Web applications.

CORBA

The Common Object Request Broker Architecture (CORBA) specification defines communications between distributed objects and integrated object services. CORBA provides a common framework for developing applications that use components. It defines the software services that enable objects to communicate transparently across a distributed computer network. CORBA is language and platform independent and can handle objects that are implemented with different vendor packages, located on different machines, coded in different programming languages, and run on different operating systems. Both TXSeries and Component Broker can be used to create CORBA-compliant applications.

Distributed object communications

Distributed objects communicate by using an *Object Request Broker* (*ORB*). The ORB sends local client requests across the network by using the *Internet Inter-ORB Protocol* (*IIOP*), which is a TCP/IP-based protocol with CORBA-defined message exchanges. IIOP allows ORBs to communicate with each other and enables them to use the Internet for distributed object communication.

Figure 6 on page 30 shows client/server communication through an ORB. The client uses a *proxy object* that represents a remote object located in another part of the network. The client does not need to know where the real object resides, since the proxy object handles communication with the remote object. The proxy object works with the ORB to translate client requests into a format that can be sent over the network. An *object adaptor* on the server finds the remote object and dispatches the request for processing. It is the primary interface between the ORB and the object implementation code. The object adaptor then sends the results back to the proxy object, which returns them to the client that initiated the request.

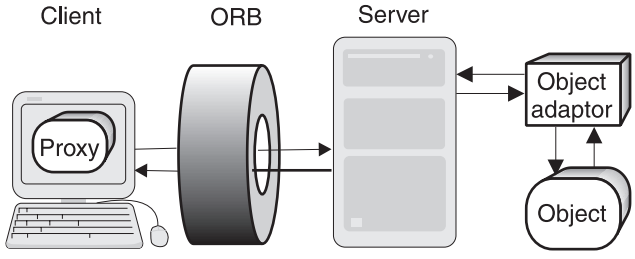


Figure 6. Clients and servers communicate by using an ORB

The proxy object and the real object are synchronized by using the *Interface Definition Language* (*IDL*). IDL specifies the behaviors and attributes that are provided by the remote object. During application development, the IDL specification of an interface is compiled to generate the code that implements the *remote interface* for the client and server. The client communicates with a remote object through its remote interface, and can interact with any remote object that supports that interface. The component interface is created in a CORBA IDL file, which is then compiled by using the CORBA **idl** compiler. This produces most of the stub and skeleton files required for creating a distributed application.

CORBA business objects implement interfaces described by proxies. The client proxy object represents an abstract interface. The server implementation of this object inherits this interface and implements the object's business logic.

Remote calls

CORBA remote calls enable communications between client proxy objects and server-side objects. CORBA servers export implementation objects to which client proxy objects bind in order to obtain a service. Objects that participate in transactions or make transactional requests on other objects are called *transactional objects*.

A remote call in CORBA occurs when a client creates a proxy object and uses that object to invoke a method on a corresponding implementation object at the server. This remote call is similar in most respects to an RPC.

Common object services

CORBA specifies common object services, which are used to manage distributed components. These services include naming, security, life cycle, event, externalization, transaction, persistence, and more. The CORBA specification also includes interfaces that support specific domains and common facilities.

Microsoft COM

Microsoft Component Object Model (COM) is a component architecture that allows applications to be built from binary software components. Using COM,

Windows applications can be developed as multiple components instead of as a single entity. The individual components of each application can be transparently and separately upgraded and applications can be distributed more easily. COM can be used to create applications in any programming language.

A COM component is a piece of compiled code that performs an action in an application. (In contrast, components in other object-oriented programming models are defined in the application's source code.) Once instantiated in a client, a COM component acts as a proxy for the client, marshalling and unmarshalling data to contact a server and returning data and status information back to the client. This encapsulates data from processing.

COM provides the following:

- A standardized component interface
- Communication between components
- Shared memory management between components
- Error and status reporting
- Dynamic loading of components

Managed Object Framework

Component Broker has its own object model. Server objects are derived from the Managed Object Framework (MOFW), which enables objects to work together and behave as a single component. Each component consists of a set of objects that are managed by an application adaptor. From the client's perspective, the component acts as a single object, even though it is implemented as a set of objects on the server.

Components are made up of three types of basic objects: business objects, managed objects, and data objects. Persistent components add a fourth type of object, persistent objects, which provides the code to access resource managers or data stores. Key objects and copy helper objects aid in locating and creating the component.

The objects that make up a component are described in the rest of this section.

Business objects

Business objects define the component's interface — that is, the attributes and methods that contain its business logic. The CORBA Interface Definition Language (IDL) is used to define a business object's interface. IDL specifies an object's interfaces, independent of operating system and programming language.

A business object can be implemented in the C++ or Java programming languages. Because the business object is derived from the Managed Object

Framework, the only code you need to provide is the implementation for any application-specific methods you defined. When you create a business object by using Component Broker tools, the framework is extended for you. Attributes that are part of the state of the object can be cached or delegated to the data object.

Data objects

Data objects manage the persistence of components' essential state information (state data). They provide an interface for the business object to get and set state data.

A data object isolates its business object from these actions:

- Knowing which data store to use
- Knowing how to access the data store
- Managing access to the data store

Managed objects

Managed objects provide the component with management by an application adaptor. Because this management capability is provided in a separate object, the type of service provided can be changed without affecting the component interface.

Some examples of managed services are:

- Persistence, transaction, and security
- Workload management and availability management over multiple servers
- Object-oriented access to existing databases and applications

Persistent objects

Persistent objects are C++ objects that provide a mechanism for storing a component's state in a data store. Every persistent object has an identifier or a key that is used for locating its corresponding record within the data store. There are two main kinds of persistent objects:

- Those used for accessing database data (mapping to a database schema).
- Those used for accessing procedural data (mapping to a CICS or IMS bean, or Procedural Adaptor bean (PA bean)).

Key objects and copy helper objects

A component's *key object* defines which attributes are to be used to find a particular instance of the component on the server. The key consists of one or more of the business object attributes, which must contain enough information to uniquely identify an instance.

A *copy helper object* provides an efficient way for the client application to create new instances of the component on the server. Its use is optional. The copy helper contains the same attributes as the business object (or a subset of

them). Without a copy helper, the client might need to make many calls to the server for each new instance: one call to create the instance, and then an additional call to initialize each of the instance's attributes. With a copy helper, the client can create a local instance of the copy helper, set values for its attributes, and then create the server component and initialize its attributes in one call by passing it the copy helper.

Assembling components

When you assemble a component, you can start from the business object, data object, or schema. Configuring the objects into a unified component involves selecting which objects form a particular component on the server. The deployed component is accessed through its managed object, by client applications or other components that require access to the server data.

Component instantiation and execution

Each component is instantiated and later located by using a home object. A *home object* is a server object that allows clients and other components to locate and create components of a certain type. After they are created, the components run in a container, which acts as an application adaptor for the component, providing management services to the component through its managed object. From the point of view of the calling program, only the business object interface (which acts as the interface to the whole component) is visible. Even the managed object is hidden.

Chapter 4. Adapting business models to WebSphere Application Server

Just as people are highly valued resources in a business, software also provides value to a business. It must meet the same expectations that are placed upon other business resources. A change driven by external influences such as government regulation, new technology, or competition can be analyzed in terms of business policies and processes. The corresponding information systems model can then be designed and adapted.

Component technology

Software must work well within existing business processes without requiring excessive accommodation. Reworking existing business processes is undesirable, as is lost productivity. Components enable a new approach to building distributed applications, where the business model drives the software implementation. This is because:

- The software model and implementation is described in business terms.
- Components have business names and embody behavior and state consistent with their business role.
- Relationships between components are established by business process.
- Models are cooperatively created and maintained by business domain experts with support from information technology specialists.

These characteristics of component-based systems provide the flexibility required by today's fast changing business climate. Consider the advantage when a change in business objectives can be mapped directly to the systems model, enabling a more rapid introduction of new components and relationships.

- New components can be introduced without unnecessary changes to existing components.
- New relationships can be established without having to modify existing (and still useful) relationships.
- Relationships can be changed without having to rewrite the entire system.
- Component-based applications can interface with existing business information systems, enabling new development to proceed without making existing investments in technology obsolete.

Common software approach

A high-level programming model allows software developers to produce useful business components that can be used directly in your system. If other businesses follow the same programming model, organizations can achieve a higher level of intercompany system integration to support collaborative engagements, mergers and acquisitions. The benefits of a common approach to object-oriented design and implementation are extraordinary and, in many respects, essential to achieve a greater return on a software investment.

Several important industry trends are supporting the adoption of a common software approach:

- Industry-wide support for component-based architectures.
- The rapid adoption of the Java programming language, the Internet, and corporate intranets.
- Distributed software topologies supported by robust enterprise servers.

The convergence of these trends has produced unprecedented opportunities based on new models of computing. At the same time, businesses must be able to use the substantial investments they have made in existing application systems and data. It would be far too costly for them to rebuild from scratch.

Software availability

IBM has been the industry leader in computing infrastructure for nearly 40 years, and has vast experience in technologies such as transactions, compilers, operating systems, communications, and databases. This technical knowledge has been utilized in WebSphere. It is available on common, industry-wide platforms and supports a wide range of operating systems, communication protocols, databases, and other resources.

Businesses improve their reliability by cross-training people to accommodate unexpected outages. Similarly, systems can improve their reliability by incorporating redundancies and respond to unexpected outages by shifting work to available resources. WebSphere incorporates automated workload management and failover features that eliminate single points of failure in systems and maintain the availability of the system. This allows the development of highly available, fault-tolerant systems.

Software reuse

Component-based systems offer the promise of reusability where individual components can be used again and again without being reworked. However, this promise has largely remained unrealized, primarily because there has been no infrastructure to provide necessary services to object systems. As a

result, support for operating systems, data management mechanisms, and communication mechanisms has had to be explicitly coded into applications.

WebSphere provides the distributed object infrastructure necessary to support reusable components. It allows developers to concentrate on the development of components that implement business functionality. Components can be reused with a variety of platforms, operating systems, databases, communication protocols, and programming languages.

Dividing up software development tasks

The WebSphere programming model breaks down application development into a number of different roles. The specific role a developer plays depends on the type of application that is being developed. These roles enable development team members to concentrate on tasks that provide the most business value without being concerned with the low-level technical details of other parts of the system. For instance, Java programmers can concentrate on developing enterprise beans, servlets, and other types of components without having to worry about how these components are used in a browser-based client. Similarly, Web site developers can concentrate on coding references to these components into Web pages without having to worry about the details of their implementation.

Role-based development allows the creation of highly productive teams, with different team members concentrating on analysis, design, development, testing, and deploying the system. Each team member can concentrate on the parts of their role that directly benefit the business.

Tool sets

Distributed object systems have recurring implementation patterns. These patterns are relatively few in number, are highly repetitive, and have little variability. This makes it possible to create a set of tools that use these patterns to generate systems. These patterns constitute a programming model, and define the ″best practices″ for developing a distributed object system.

The WebSphere development tools help to improve developer productivity. They can determine what information is needed based on the patterns that are being utilized, locate and manage this information, and then use it to generate, deploy and manage the resulting system. This frees software developers to concentrate on providing information that is not contained in the tool: the design and implementation of business logic.

Scalability

Productivity can be threatened by growth. As a business grows, it must add employees to handle the increased workload. The same holds true for computer systems that support growing businesses. As they become saturated, the service level they provide to the business reaches a plateau, and in some cases actually decreases. This can damage the growth of the business, and must be remedied as quickly as possible.

WebSphere allows systems to grow by adding more computing power, either by adding more computers or by migrating functions to more powerful platforms. It monitors executing systems so that signs of saturation are visible before they have an adverse impact. It also allows dynamic configurations, so that conditions can be responded to without making the system unavailable. This is particularly important if problems occur at times when the system's availability is most critical.

Open standards and investment protection

Systems that cannot communicate with other systems or require large changes in other systems present enormous challenges to businesses. This can cause problems ranging from lost productivity to abandoning investments in non-working systems.

WebSphere can communicate with other systems in a variety of ways with little or no impact on those systems. This allows WebSphere-based systems to be as productive as possible, and enables the other systems to maintain their productivity.

Open standards define ways in which various segments of the computer industry have agreed to implement common functions. WebSphere supports open standards such as the Java 2, Enterprise Edition (J2EE) specification, Common Object Request Broker Architecture (CORBA), XA, secure sockets layer (SSL), Kerberos, Logical Unit (LU) 6.2, and other industry-wide standards.

Businesses invest heavily in their information technology systems. Distributed object systems must be designed to work with existing systems. WebSphere protects existing information technology investments by providing ways to communicate and interoperate with existing systems.

Chapter 5. Introduction to WebSphere Application Server, Standard and Advanced Editions

IBM's 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. It is available in three editions: Standard, Advanced, and Enterprise.

What is the difference between the Standard and Advanced Application Servers?

The Standard and Advanced Application Servers have many similar features and both can be used to create robust business information systems. However, there are several major difference between the Standard Application Server and the Advanced Application Server:

- The Standard Application Server supports only a single-machine server environment. The Advanced Application Server supports a multiple-machine server environment and includes underlying support for services such as naming, transaction, and workload management. However, both editions can support access from multiple client machines.
- The Advanced Application Server supports the replication of EJB server models that makes it easy to clone EJB servers across multiple nodes, enabling workload management and improving availability. The Standard Application Server does not permit replication.
- The administrative interfaces to the two application servers differ somewhat as a result of the differences in functionality. The interface to the Advanced Application Server cannot be used to administer a Standard Application Server environment and the interface to the Standard Application Server cannot be used to administer an Advanced Application Server. The Advanced Application Server stores configuration information in a database repository; the Standard Application Server stores it in XML files.

The Enterprise Application Server includes the Advanced Application Server. So if you purchase the Enterprise Application Server, you can use any of the products in any of the three Application Servers to implement your e-business solutions.

The rest of this chapter focuses on the Advanced Application Server because it contains everything that is in the Standard Application Server and more.

Introduction to the Advanced Application Server

The WebSphere Application Server Advanced Edition provides the following major functionality:

- Tools for developing active Web sites through the use of Java servlets and JavaServer Pages (JSP). This functionality is also available in the Standard Application Server.
- Tight integration with tools for developing and deploying enterprise beans written to the EJB specification. Enterprise beans can act as a bridge between your Web site and your non-Web computer systems.
- A graphical user interface (GUI), the WebSphere Administrative Console, for administering the components of the Advanced Application Server environment. This functionality is also available in the Standard Application Server.
- A set of application programming interfaces (APIs) for generating, validating, parsing, and presenting extensible markup language (XML) documents. (This functionality is also available in the Standard Application Server.)

The Advanced Application Server environment

Figure 7 shows the components that make up the Advanced Application Server.

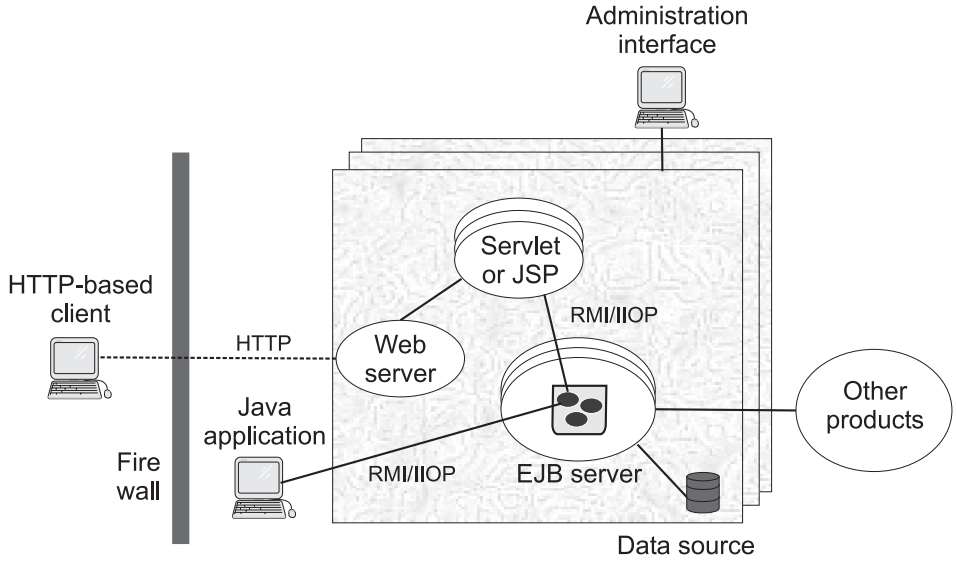


Figure 7. The components of the Advanced Application Server environment

They can be combined to create a powerful, Java-centered, three-tiered system that puts heavy emphasis on a customer's Web site. Each part of the Advanced Application Server is explained as follows.

Administration server

The *Administration server* and the *administrative interface* enable application servers and processes to be monitored and controlled centrally. For more information, see "The administration model in Advanced Application Server" on page 43.

Browser-based clients

Clients of applications that run on the Advanced Application Server generally run in Java-enabled browsers. They send and receive information from a Web server by using HTTP. Browser-based clients can include applets, described in "Applets and servlets" on page 26, and JavaServer Pages (JSP), described in "JavaServer Pages" on page 28.

Web servers

Except for stand-alone Java applets, which are restricted by built-in Java security, browser-based client applications require that a Web server be installed on at least one machine in your Advanced Application Server environment. The Web server provides the communications link between browser-based applications and the other components of Advanced Application Server.

WebSphere Application Server supports many of the most widely used Web servers. The IBM HTTP Server, which is a modified version of the Apache server, comes with the Advanced Application Server. For information on the supported Web servers, refer to the IBM WebSphere Application Server site at www.ibm.com/software/webservers/appserv.

Servlet engine

The Advanced Application Server contains a Java-based servlet engine that is independent of both your Web server and its underlying operating system. Java servlets extend the Web server's capabilities by creating a framework for providing request and response services over the Web. For more information on servlets, see "Applets and servlets" on page 26.

Enterprise beans

Advanced Edition provides full support for enterprise beans. An enterprise bean is a Java component that can be combined with other enterprise beans and other Java components to create a distributed, three-tiered application. An *EJB server* provides the run-time environment for enterprise beans, handling low-level programming tasks like transaction management, naming, and security. Java applications can interact directly with an EJB server by using Java remote method invocation over the Internet Inter-ORB Protocol (RMI/IIOP).

For more information on enterprise beans, EJB servers, and containers, see "Enterprise beans" on page 24.

Application model

Advanced Application Server applications consist of object-oriented business logic that use relational database systems for data storage. Applications are usually integrated with Web clients (either thick or thin); they can also be integrated with existing procedural applications running in application servers.

An application consists of the following components, each performing a different function:

- HTML and JSP pages provide the user interface and program flow.
- Enterprise beans contain the application's business logic and handle transactional operations and access to databases.
- Servlets coordinate work between the other components of the application. They also can dynamically generate Web page contents.
- JavaBeans components enable the other types of components to work together.
- Relational databases implement persistence and query functions for enterprise beans. Either new or existing databases can be used in an application.

WebSphere Programming Model Extensions

The Programming Model Extensions provide reusable business logic for Java programs. There are two sets of tools provided in the Advanced Application Server environment for Java programmers: the command package and the distributed-exception package. These tools are documented in *Writing Enterprise Beans in WebSphere*, but their use is not restricted to enterprise beans.

The command package provides a way for distributed applications to bundle remote requests together, reducing the number of individual remote invocations. Remote invocations are expensive, so the command package can help you improve the performance of distributed applications. In addition, the command package provides a generic way of making requests. The package provides a common way to issue a command, locally or remotely, and independently of a server's implementation. Any server (an enterprise bean, a JDBC server, and so on) can be the target of a command.

The distributed-exception package helps you manage exceptions in distributed applications. When writing complex distributed applications, you face a choice in handling exceptions. One option is to manage each exception explicitly, catching and rethrowing each by name. This ensures that the information about the original exception is not lost, but it can lead to

unmanageable code as the number of exceptions increases. The other option is to adopt a strategy of throwing one exception when you catch any of a group. This choice allows you to keep the number of exceptions manageable, but you lose information as exceptions pass through an application. The distributed-exception package allows you to chain a sequence of exceptions into a throwable object. With an exception chain, you can throw one exception in response to another, without losing the previous exceptions. You can also retrieve exceptions from the chain.

The administration model in Advanced Application Server

WebSphere Application Server provides central administration of EJB servers and other resources. The administration server manages servlets, JSP files, enterprise beans, and EJB servers. This management is directed by the WebSphere Application Server administrator who uses the WebSphere Administrative Console, which is the interface to the administration server.

In WebSphere Application Server, an *administrative domain* is a collection of host machines called managed nodes. Each managed node runs an *administration server* (administration servers are also EJB servers). A node's administration server is responsible for configuring, monitoring, and managing the resources on that node. Resources include live objects such as EJB servers, containers, deployed beans, JSP files, Java servlets, and applications. Resources also include objects such as method groups or policies that are used to define security for resources in the domain.

Resource beans are container-managed persistent (CMP) entity beans. The persistent data associated with a resource (for example, the name, current state, and executable file of an EJB server) is stored in a central data repository. The administration server communicates with a repository server to access, define, and modify resource information stored in the repository. An administration server also communicates with other (remote) administration servers to delegate tasks and to respond to requests. The IBM DB2 relational database, which is packaged with the Advanced Application Server, acts as the repository server. You can also use other databases such as Oracle, Sybase, or InstantDB.

Administration takes place through method calls to resource beans in the repository server. The WebSphere Administrative Console makes requests to an administration server to access or modify a resource in the domain. In the administration server, session beans invoke methods on the resource beans. Each resource bean has an associated attribute class that contains methods for getting and setting attribute values.

All administration servers in a domain share the central storage for resources in that domain. Regardless of the node it is running on, any administration

server can view and modify the characteristics or status of resources on other nodes. If an administration server calls a method on a resource that is running on a different remote node, the method is delegated from the local administration server to the remote administration server.

Resources are modeled in an object type hierarchy that relates the object types to each other. Other object types represent entities such as server groups. Related objects inherit methods from objects above them in the tree hierarchy.

Certain objects in the administrative domain, such as EJB servers, can be copied (*modeled*) to create replicas (*clones*) that perform identical functions to the object from which they are replicated. This enables the administrator to duplicate server functionality across multiple nodes, improving availability and efficiency. After you clone a resource, modifying the model automatically propagates the same changes to all of the clones. You can efficiently administer several copies of a server or other resource by administering its model.

Resources that can be cloned include the following:

- Application servers
- EJB containers
- Enterprise beans
- Servlets
- Servlet engines
- Web applications

Administration tools

The WebSphere Administrative Console is the administrative interface to the Advanced Application Server. It can be used for a range of administrative tasks—from configuring resources and setting security policies, to starting servers and deploying beans, to identifying and responding to system failures and monitoring usage patterns. The tasks supported by the WebSphere Administrative Console fall into six categories: configuration, operation, security, troubleshooting, performance, and data storage.

The WebSphere Administrative Console provides a centralized hierarchical view of all resources in an administrative domain, guides for performing administrative operations, forms for viewing and modifying a resource's attributes, a central browsing facility for JAR files, a messages window for monitoring critical events, and context-sensitive help. The WebSphere Administrative Console modifies information in the repository in response to user commands and reflects any changes to the configuration and status of the administrative domain.

The extensible markup language (XML)

XML is designed to make it easy to exchange documents (and other structured information) over the Internet. In simple terms, it is a standard for defining document markup languages. A markup language is a set of elements (frequently called tags) that have one or more of the following functions:

- Describing the structure of the document.
- Describing the content of the document.
- Controlling how the document is presented to the user.

HTML is the most widely used markup language for Web-based documents, but it has many limitations. HTML tags describe the visual presentation of Web pages and do not really specify their logical structure. HTML users are restricted to a relatively small set of tags and cannot create their own tags because commercially-available Web browsers do not support tags that are not part of the HTML standard.

HTML is further limited because the tags that control presentation are in the same file with tags that describe the document content. Although HTML 4 and Cascading Style Sheets enable HTML authors to separate content from presentation, HTML 4 is limited in its ability to describe the content of a document.

XML overcomes these limitations. XML users can define their own custom tag set, or use tags from any publicly available document type definition (DTD). XML tags specify the content and structure of a document. Presentation is separated from the document's content.

The *XML Document Structure Service* contained in the Advanced Application Server enables users to develop servlets and applications that implement server-side XML processing. It includes a set of APIs for setting servlet configuration parameters without using the administration interface. This alternative method involves creating an XML servlet configuration file (which is an XML document named servlet_instance_name.servlet) that contains the following:

- The name of the servlet class file.
- A description of the servlet.
- The servlet initialization parameters.
- A page list that contains the universal resource identifiers (URIs) of each JSP file that the servlet can call. The page list can include a default page, an error page, and one or more target pages that are loaded if their name appears in the HTTP request.

Services used by the Advanced Application Server

Although the Advanced Application Server is primarily concerned with the Web side of your business, the EJB server can act as a bridge to connect your Web and the non-Web applications to span all of your business systems. This section looks at some of the generic tasks that must be accomplished to enable the development and use of distributed applications. It also describes the tools used to approach each of these tasks in the Advanced Application Server.

Naming service

In an object-oriented distributed computing environment, clients must have a mechanism to locate and identify the objects as if the clients and objects were all on the same machine. A naming service provides this mechanism. In the EJB server environment, the Java Naming and Directory Interface (JNDI) is used to provide a common front end to the naming service.

JNDI provides naming and directory functionality to Java applications, but the API is independent of any specific implementation of a naming and directory service. This independence ensures that different naming and directory services can be used by accessing it behind the JNDI API. Therefore, Java applications can use many existing naming and directory services, for example, the Lightweight Directory Access Protocol (LDAP) or the Domain Name System (DNS).

Transaction service

A *transaction* is a set of operations that transforms data from one consistent state to another. The EJB server manages transactions for EJB applications by using the mechanism defined in the Java Transaction API (JTA).

For most purposes, enterprise bean developers can delegate the tasks involved in managing a transaction to the EJB server. The developer performs this delegation by setting the deployment descriptor attributes for transactions. The enterprise bean code itself does not need to contain transactional logic.

For more information on transactions, see "Transactions: ensuring data consistency and permanence in a distributed environment" on page 12.

Security service

In the Advanced Application Server environment, the main component of the security service is an EJB server that contains security enterprise beans. When system administrators administer the security service, they manipulate the security beans.

After an EJB client is authenticated, it can attempt to invoke methods on the enterprise beans that it manipulates. A method is successfully invoked if the principal associated with the method invocation has the required permissions to invoke the method. These permissions can be set by application (an

administrator-defined set of Web and object resources) and by method group (an administrator-defined set of Java interface-method pairs). An application can contain multiple method groups.

In general, the principal under which a method is invoked is associated with that invocation across multiple Web servers and EJB servers (this association is known as *delegation*). Delegating the method invocations in this way ensures that the user of an EJB client needs to authenticate only once. HTTP cookies (identification files stored on a client computer's hard drive) are used to propagate a user's authentication information across multiple Web servers. These cookies have a lifetime equal to the life of the browser session.

Workload management service

Workload management (WLM) optimizes the distribution of processing tasks in the WebSphere Application Server environment. Incoming requests are distributed to the application servers and other objects that can most effectively process the requests.

The workload management service ensures that client requests are distributed according to the capacity of each machine in the system, making it easier to scale up systems to meet higher client loads. It also provides failover support by redirecting requests and simplifies maintenance and administration.

The WebSphere Application Server environment provides several ways to manage workloads:

- Clones and models can be created for application objects, from individual enterprise beans and servlets to entire application servers. The Advanced Application Server's WLM service distributes processing requests among enterprise bean and EJB server clones.
- Servlet redirection is used to route client requests to servlet clones.
- EJB servers can be combined into EJB server groups. Clients then access these EJB server groups as if they were a single server.
- WLM-enabled administration servers provide failover and load balancing for administration requests.

Development environment

WebSphere Studio is the WebSphere Application Server application development environment. It 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 suite for developing HTML content and can be integrated with other content development tools.

VisualAge for Java is an integrated development environment that supports the complete cycle of Java program development. Although it is not formally

a part of the Standard or Advanced Application Servers, 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.

Documentation

The Standard and Advanced Edition InfoCenters provide information that can help you to install and configure WebSphere Application Server and plan, develop, deploy, and troubleshoot applications. The InfoCenters for both Standard and Advanced Edition can be accessed online from the WebSphere Application Server library page at www.software.ibm.com/webservers/appserv/library.html.

Chapter 6. WebSphere Application Server Enterprise Edition

This chapter provides a brief discussion of the contents of the WebSphere Application Server Enterprise Edition. It also discusses the environments in which the Enterprise Application Server products run and the additional products packaged with the Enterprise Application Server.

Why use Enterprise Application Server?

The Enterprise Application Server contains all of the products found in the Advanced Edition and adds the following major product components:

- Component Broker, which is an enterprise solution for distributed computing, providing a scalable, manageable run time for developing and deploying distributed component-based solutions. It is a complete and integrated implementation of the open standards contained in the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA). In addition, Component Broker contains a separate implementation of the EJB Specification, which can be used with or instead of the implementation contained in the Advanced Edition.
- TXSeries, which contains two popular middleware packages (CICS and Encina) that support and simplify the creation of transactional applications that can span multiple platforms in diverse and complex networks. In addition to offering cross-enterprise integration, TXSeries applications provide high levels of scalability, availability, integrity, longevity, and security.

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

The rest of this chapter explains some of the major underlying environments and services on which Enterprise Application Server runs. It also briefly discusses some of the other products that are licensed for use with the Enterprise Application Server. The following two chapters examine the components of the Enterprise Application Server and explain what each of these components do best.

Low-level services used in Enterprise Application Server products

Many of the Enterprise Application Server products rely on one or more of the following services to handle low-level tasks such as security, naming, and remote procedure calls:

- The Open Group's Distributed Computing Environment (DCE). For more information, see "Distributed Computing Environment (DCE)".
- The Object Management Group's (OMG) Component Object Request Broker Architecture (CORBA). For more information, see "Common Object Request Broker Architecture (CORBA)" on page 52.
- Microsoft Corporation's Component Object Model (COM). For more information, see "Component Object Model (COM)" on page 52.

The Enterprise Application Server also contains an implementation of the EJB Specification (and related Java specifications) that is built into the Component Broker product. Information about this implementation is provided in the general discussion of Component Broker; for more information see "Chapter 7. Introduction to Component Broker" on page 55.

Distributed Computing Environment (DCE)

DCE enables distributed transaction processing environments using Component Broker or TXSeries to run seamlessly across machines that differ in hardware, operating system, network transport, and application software. The DCE layer extends the basic operating systems of the separate machines to provide a common infrastructure for distributed computing. By using the standard interfaces provided by DCE, applications can interoperate with and be ported to other DCE platforms. The following sections describe the DCE services used by Enterprise Application Server products.

Remote procedure call (RPC)

At the core of DCE support is the *RPC*. RPCs provide a form of network-transparent communication between processes in a distributed system. Processes use RPCs to communicate in exactly the same way regardless of whether they are on the same machine or different machines in an administrative unit known as a cell. The DCE Security Service can be used to authenticate RPCs. *Authenticated RPCs* can be checked for tampering and can be encrypted for privacy. DCE uses *multithreading* to enable a client to have multiple concurrent RPC conversations with servers and to enable a server to handle many concurrent client requests.

Cell Directory Service (CDS)

The CDS provides a *namespace* within which network resources can be accessed by name only. Applications do not need to know the addresses of resources. (Typical network resources are servers, users, files, disks, or print queues.) Further, if a resource is moved, it can still be located by the same name; application code does not need to be changed.

The CDS Server manages a database, called a *clearinghouse*, which contains the names and attributes (including locations) of network resources in the DCE cell. When a request is made for a network resource, the CDS Server locates the resource.

The DCE Directory Service also supports a global name service for identifying resources outside a cell.

DCE Security Service

The DCE Security Service provides secure communications and controlled access to network resources in a DCE cell. It verifies the identity of DCE *principals* (users, servers, and DCE-enabled clients) and allows them to access only the network resources that they are authorized to use. The DCE security service does the following:

- Manages a central source of security information in the cell's security database.
- Validates the identity of interactive principals, such as users, by enabling them to log into DCE. This is known as establishing a login context.
- Grants *tickets* to principals and services so their communications are secure.
- Certifies the credentials of principals to control principals' access rights to resources.
- Validates the identity of noninteractive principals, such as CICS regions, by enabling them to perform the equivalent of an interactive user login. In this way, they can establish their own login context rather than running under the identity of the principal that started them.
- Controls the authorization that principals have to network resources in the DCE cell. Each object in the DCE cell can have an associated *access control list (ACL)* that specifies which operations can be performed by which users. ACLs can be associated with files, directories, and registry objects, and can be implemented by arbitrary applications to control access to their internal objects.

The DCE Security Service is implemented as a security server, which maintains a store of security information about network resources in its *security database* (also known as the DCE *registry database*).

Distributed Time Service (DTS)

To compensate for natural drifts in system clocks, the DCE DTS ensures that all system clocks of the servers in a DCE cell are synchronized. This is especially important where servers are in different time zones. A time service is also essential to ensure the reliable operation of authentication and authorization services.

Common Object Request Broker Architecture (CORBA)

The OMG created the CORBA specification to facilitate the development of object-oriented applications across a network. CORBA objects are standard software objects implemented in any object-oriented programming language. An Object Request Broker (ORB) mediates the transfer of messages between client programs and objects. When a client program invokes a method on an object, the ORB intercepts the request and finds an object implementing that method. The result of the method invocation is returned to the client program by the ORB. From the programmer's point of view, all of the work appears to be done on one computer system.

The Internet Inter-ORB Protocol (IIOP) enables communications between different ORB implementations. The IIOP is based on TCP/IP and includes additional message-exchange protocols defined by CORBA.

Various Enterprise Application Server products support one of the following ORBs:

- The IONA Orbix ORB
- The IBM Component Broker ORB
- The IBM Java ORB

For more information, see "CORBA" on page 29.

Component Object Model (COM)

Microsoft COM is a model for developing applications composed of individual components that can be transparently and separately upgraded. When COM is used, Windows applications can be developed as multiple components rather than as a single entity; this enables the application to be distributed and maintained more easily.

Both TXSeries Encina and Component Broker provide COM functionality on Windows NT systems.

- Encina applications can use COM components to transparently handle the functions of Encina clients. Clients can contact Encina servers, manage transactions, and perform other tasks by instantiating Encina COM components and calling standard member functions on these components.

  COM components can be created from standard Encina Transactional Interface Definition Language (TIDL) files. The resulting dynamic linked library (DLL) can be used with the prepackaged Encina COM DLLs and other, non-Encina COM DLLs to develop Encina clients.
- Component Broker applications can use COM components to access managed objects. A managed object is wrapped at run time by a COM component, which serves as a proxy for accessing the object. COM components can also be used to access remote CORBA objects.

COM components can be created from standard Interface Definition Language (IDL) files. The resulting DLL can be used with other COM DLLs to develop Component Broker clients. COM components for some Component Broker object services are also provided.

For more information on COM components, see "Microsoft COM" on page 30.

Other tools available with the Enterprise Application Server

The Enterprise Application Server includes the following additional products that are required by (or recommended for use with) the main tools of the Enterprise Application Server:

- IBM DB2—DB2 is a distributed relational database that can be used as a resource manager in conjunction with TXSeries and Component Broker. DB2 can be used by the EJB administration servers contained in the Advanced Application Server and must be used by the EJB administration servers contained in Component Broker. 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.
- IBM HTTP Server—The IBM HTTP Server is a powerful Web server that is based on the popular Apache Web server. In addition to providing the full range of Web server features, it provides enhanced SSL for secure transactions. The Advanced Application Server also provides plug-ins for the most popular Web servers, enabling your Web server to extend into a Java application server.
- IBM MQSeries—The IBM MQSeries® range of products enables application programs to communicate in a nonserial, asynchronous manner by using messages and queues. At the heart of MQSeries is the Message Queue Interface (MQI), a high-level programming interface that enables applications to communicate transparently across various platforms. MQI takes care of network interfaces, assures delivery of messages, deals with communications protocols, and handles recovery from system problems.
- IBM VisualAge for Java Enterprise Edition—VisualAge for Java is a powerful integrated development environment (IDE) that contains many features for building Java-based business information systems. This powerful IDE provides support for developing and testing Java applications and components written to the Enterprise JavaBeans and JavaBeans specifications. For more information, see "VisualAge for Java" on page 7.
- IBM VisualAge C++ Professional Edition—VisualAge C++ provides a rich environment and toolset for multiplatform object-oriented application development. 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.

Chapter 7. Introduction to Component Broker

Component Broker provides a scalable, manageable environment for developing and deploying distributed component-based solutions. It is an integrated implementation of the open standards contained in the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) initiative. Many of the low-level details of the CORBA interface are hidden by Component Broker's easy-to-use framework. It is available on the Windows NT, AIX, Solaris, and OS/390 platforms.

This overview provides an introduction to the following Component Broker features:

Component Broker features

Component Broker is primarily an object server. It comes with a development environment that is optimized for creating business objects that run in the Component Broker server. This server consists of both a run-time package called the Component Broker Connector (CBConnector) and the VisualAge Component Development Toolkit (CB Tools). The run-time package provides a server in which business object components run and are managed through a set of management tools.

As an object server, Component Broker provides an application environment that lets clients access back-end systems through object-oriented middleware. This system provides an infrastructure scalable enough to include everything from desktops to the largest cluster of mainframes.

The Component Broker run-time environment supports the execution of C++ and Java-based business logic that follows the CORBA model or the model of the Enterprise JavaBeans (EJB) Specification from Sun Microsystems. The Enterprise JavaBeans Specification provides a portable, platform-independent reusable component architecture. These components run in a robust, multithreaded server that provides easy access to a wide variety of services and capabilities.

Component Broker's object services are available in an integrated fashion based on the OMG model and the Component Broker Managed Object Framework (MOFW), a set of configurable and extensible object interfaces. Key object services provided include Data Cache and Prefetch, Concurrency Control, Event, Externalization, Object Identity, LifeCycle, Naming, Query, Security, and Transaction. Application adaptors extend and specialize these interfaces and provide a home and container for Managed Objects, similar to an object-oriented database

Component Broker contains a CORBA 2.0–compliant Object Request Broker (ORB). This ORB is largely encapsulated by the Managed Object Framework and the object services provided by Component Broker. The ORB facilitates interoperability with other Internet Inter-ORB Protocol (IIOP) servers and with clients. Component Broker supports client access by using the Internet Inter-ORB Protocol (IIOP) directly or by using Remote Method Invocation over IIOP (RMI/IIOP). This enables Java clients (applet, application, or servlet), C++ clients, and Microsoft ActiveX® or Visual Basic clients to access business objects running in the server.

Component Broker applications can access resources on DB2, Oracle, CICS, IMS, and MQSeries. Business objects that have data objects backed by these resource managers can participate in distributed transactions. Component Broker acts as an external commit coordinator for all of these resource managers through the implementation of the CORBA Object Transaction Service (OTS).

System Management tools enable administrators to control the distributed-object computing environment. These tools enable the modeling of the deployment configuration and then the actual management of the abstractions that are introduced by the distributed-object paradigm. Administrators can enlarge configurations by adding servers and server groups. They can alter qualities-of-service through container management, and they can scale up environments by adding additional computing resources into the object server pool.

Development of applications that run on Component Broker can be done in three ways:

- Component Broker provides the Object Builder tools for building CORBA-based applications. Designs for systems can be imported into Object Builder from the Rational Rose visual modeling tool. After these designs are imported, the template for the implementation is available for use. Developers fill in the business logic within the implementation framework. Object Builder takes care of the rest. It generates the code, makefiles, and application configuration information necessary to test the application on a Component Broker server. Object Builder also supports development of large-scale systems by teams of developers.

- Enterprise beans created in other tools (such as VisualAge for Java) can be deployed in Component Broker, which can also act as an EJB server. Developers can build enterprise applications by using the EJB programming model. Object Builder serves as the deployment tool for enterprise beans that run in the Component Broker run time and facilitates the mapping of entity beans with container-managed persistence (CMP) to databases and existing applications. This support is based on the same technology used to map CORBA-based business objects to existing resource managers. Enterprise beans also benefit from the same implementations of object services that are available to CORBA-based business objects.
- Distributed object applications that combine enterprise beans and Component Broker business objects can be easily constructed, tested, and deployed using the Object Builder tool and the Component Broker run time.

Component Broker applications can also make use of the WebSphere Programming Model Extensions. For more information, see "WebSphere Programming Model Extensions" on page 42.

Application architecture

Component Broker applications are designed as three-tiered applications (as described in "Three-tiered client/server computing" on page 11). The content of each tier is summarized below:

- *First tier*—Client applications can be C++ programs, Java applets, or Visual Basic programs that a user interacts with directly; they can also be Web servers or application servers with their own clients. Clients can access components on the server through proxy objects over a CORBA-compliant ORB. The Component Broker client programming model also supports access to components from clients that use programming models such as ActiveX/COM objects.
- *Middle tier*—A run-time environment in which components containing the application's business logic are deployed on a Component Broker server. Components can be CORBA-based business objects or enterprise beans. The server can have components on one machine or on many different machines.
- *Third tier*—Data in various resource managers, including DB2, Oracle, CICS, and IMS. Application adaptors in the middle tier allow components to access third-tier resources. The third tier can be running on many different physical hosts, and can use application logic that itself provides additional physical tiers.

Development of three-tiered applications is made possible by the Managed Object Framework. The framework is supported by a suite of development

tools, which allow you to make use of the framework to create components without going into details of inheritance and framework implementation.

Components are deployed in the middle tier, connecting the first tier (client) with the third tier (databases and other resources). The components are implemented as CORBA-based business objects or enterprise beans. They allow application logic to run on high-powered servers, and insulate client applications from the complexities of the various resource managers. The client works with the components through a CORBA-compliant ORB, and the components work with the resource managers, using whatever communications protocols are supported by the target resource manager (for example, TCP/IP). For an overview of Component Broker's component architecture, see "Managed Object Framework" on page 31.

A Component Broker server process provides a complete execution environment and partial implementation for managed objects. Method requests are routed from the ORB to the server. The server, in conjunction with the container and adaptor, ensures that method requests are dispatched.

For more information, see the Component Broker *Programming Guide*.

Application adaptors

An application adaptor is similar to an object-oriented database. It provides a place for the managed objects of components, similar to the way in which a database system provides a home for data or records. Application adaptors are responsible for providing systems capabilities (for example, identity, caching, and persistence) for their managed objects. Component Broker provides the following application adaptors:

- The Procedural Application Adaptor (PAA), which enables Component Broker applications to access procedural resources such as CICS, IMS, or SAP.
- The MQSeries application adaptor, which integrates Component Broker applications with MQSeries applications.
- Database application adaptors, which are used to access relational databases such as DB2, Oracle, and Informix.

To learn how to use the adaptors available with Component Broker, see the following documents:

- *MQSeries Application Adaptor Development Guide*
- *Procedural Application Adaptor Development Guide*
- *Database Application Adaptors Development Guide*

Object services

The Component Broker application server provides a number of services to business objects or enterprise beans. Some of these object services are administrative in nature; their behavior is configured through the management tools. Others are presented as interfaces, and still others are built into the infrastructure and work on behalf of the business logic. For more information on these services, see the Component Broker *Advanced Programming Guide*.

Concurrency Control Service

The Concurrency Control Service is intended for use in a transactional environment. It consists of a set of interfaces that allow an application to coordinate access by multiple transactions or threads to a shared resource. When multiple transactions or threads try to access a single resource at the same time, any conflicting actions are reconciled so that the resource remains in a consistent state.

Event Service

The Event Service enables objects to communicate asynchronously. There are two defined roles: supplier objects and consumer objects. Suppliers produce events, and consumers process events. Component Broker uses the Event Service to uniquely identify the occurrence of an event. The event message does not convey anything more about the event instance except that it occurred.

Notification Service

The Notification Service enables objects to register (and unregister) their interest in certain events. It contains event channels that allow supplier objects to communicate with consumer objects asynchronously, without affecting the low-level object processes.

Externalization Service

The Externalization Service enables objects to save and restore their states in a nonobject form. This allows the object's state to exist independently of the object itself. The state can be maintained indefinitely without regard to the continued existence of the original object or the ORB process in which it existed.

Identity Service

The Identity Service provides unique identity information for each object. It derives an object's identity from relative information that positions the object within its container, server, host, and domain.

LifeCycle Services

The LifeCycle Services provide operations for creating, copying, moving, and deleting objects in a distributed environment. Clients need to perform LifeCycle operations on objects in various locations.

Naming Service

The Component Broker Naming Service allows you to create naming hierarchies to easily locate objects. In conjunction with other services, clients can navigate through different naming context trees to locate specific objects. Component Broker Naming Service handles both absolute and relative paths.

Security Service

Component Broker supports a variety of security services. It provides the mechanisms and technologies to secure your distributed system. However, your distributed system is no more secure than you make it. The Component Broker run time (specifically the application adaptor to which the object is configured) is responsible for establishing the authenticity of any requests made on the data system for that object's persistent data. These mechanisms include the following:

- Authentication
- Message protection
- Authorization

The Security Service is used primarily to prevent end users from accessing information and resources that they are not authorized to use. This predominantly covers distributed business objects, but by extension includes any of the information and resources from other nonobject-oriented or nondistributed sources used by those business objects.

In many cases, Component Broker is used to wrap legacy information system resources, such as business applications and enterprise data. Often, those resources have been centralized and held in a physically secure environment or with restricted access over controlled access channels.

Transaction Service

The Transaction Service provides standard, distributed object interfaces that can be used by programmers to implement transactions. Component Broker uses the Transaction Service to ensure that transactional data is updated consistently. If an application uses the Transaction Service in conjunction with the Concurrency Service, these updates are not affected by updates being performed for other tasks.

For more information on transactions, see "Transactions: ensuring data consistency and permanence in a distributed environment" on page 12.

Session Service

The Session Service enables applications to control the extent of a session and provides information on the application profile and session properties that are relevant within the scope of that session.

The scope of the session is defined to exist between the point when the session starts and the point when it ends. Each session provides the ability, outside of a transaction context, to isolate particular instances of objects from other sessions; each session can have its own view of object data.

Query Service

The Query Service locates objects in a Component Broker collection based on a set of conditions described with an object-oriented structure query language (OOSQL). OOSQL is a extension of SQL with features for handling object collections, object attributes, and methods in query statements. It supports complex search criteria.

The Query Service can return either a list of object references or a list of object attribute values. It uses the search capabilities and indexes in the underlying database to make searching for objects more efficient.

Cache Service

The Cache Service enhances concurrency and performance by supporting optimistic and pessimistic caching of data. In optimistic caching, frequently used data is cached in the memory of the Component Broker server and not reread from the database on each transaction. Cached data is invalidated based on a time-out value. Pessimistic caching is used when the application must be guaranteed current data and uses a higher degree of isolation to guarantee serializability of transactions. You configure the caching mode by using the System Management End User Interface on each object type.

Workload Management

As more clients use an application, the amount of work increases and the load on the servers increases. The Workload Management capability allows the Component Broker run time to dynamically allocate an application server to process a request. This minimizes the response time for client requests and maximizes server dispatch by reducing routing between objects in the distributed network. The key to workload management in Component Broker is the use of a server group to define multiple application servers with a common configuration.

System management

Component Broker provides a range of system management functions. On the UNIX and Windows NT platforms, system management is handled through a graphical user interface (GUI). This GUI helps you administer and operate your enterprise easily and effectively. The System Manager user interface displays and acts on the entities in your enterprise as system management objects. On the OS/390 platform, system management is handled by using the Administration Application and the Operations Application.

As shown in Figure 8, the System Manager controls the applications, servers, and clients within the Component Broker network. All servers, clients, applications, and their resources are managed as system management objects. Information about those objects is stored in the central configuration data maintained by the System Manager. The System Manager interacts with the system management objects on each managed host in its network through the System Manager agent that runs on the managed host. It presents administrators with data about objects in its network through the System Manager user interface.

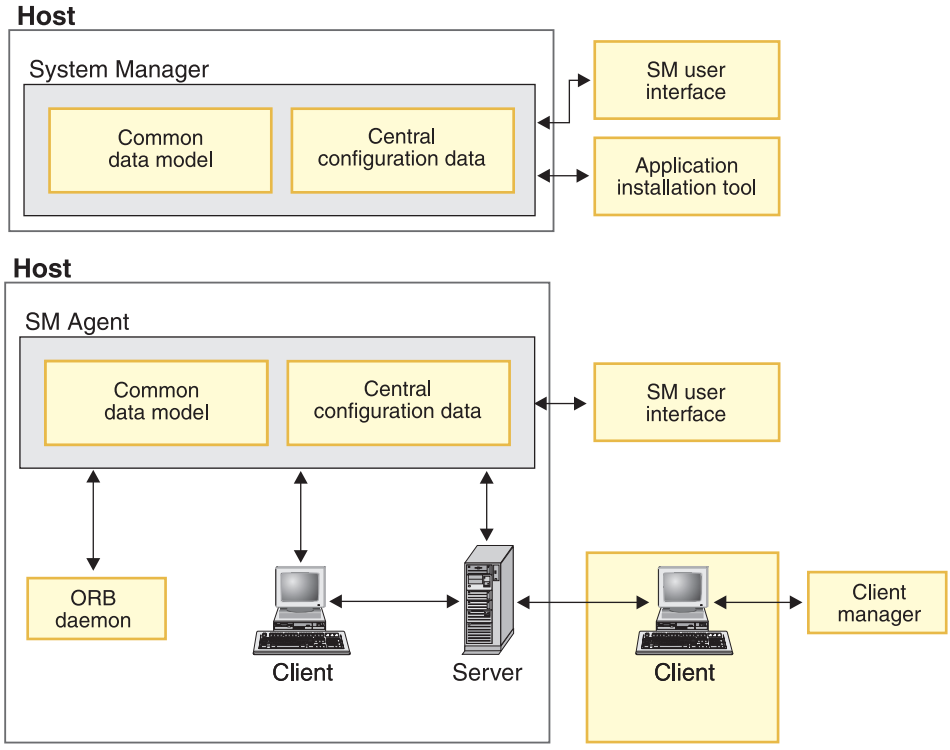


Figure 8. The Component Broker System Manager

For effective systems management, organize your systems and resources according to your business needs. For example, you can use the System Manager to do the following:

- Group your host computers into cells and workgroups.
- Configure your enterprise as one or more management zones to be managed as separate units. The hosts in your enterprise are automatically configured into one management zone, the network zone. Business applications and their clients and servers can be configured into one or more other application management zones.

- Create alternative configurations of each management zone to provide support for changes in your business needs. For example, you can have configurations that support different applications or geographical areas, use different servers, or even use different host computers.
- Configure workload management of your applications across controlled server groups.
- Verify that your configurations are complete, properly defined, and ready for activation.
- Activate an entire configuration of a management zone with one action to update all or part of your business enterprise.

You can do all this with minimal definition of system management objects. When you verify that a configuration can be activated, the System Manager determines what system management objects and relationships are needed and, if possible, creates those objects and relationships.

You can also use Component Broker system management to operate your enterprise and to perform functions on run-time objects. For example, you can do the following:
- Start and stop individual servers and applications.
- Display and act on the status of managed objects.
- Display and change the attributes and relationships of managed objects.
- Change workload management across server groups.

Development tools

The CB Tools can be used with VisualAge for Java and VisualAge C++ to generate multitier applications.

Object Builder

Object Builder is the development environment for Component Broker. You can use Object Builder to:
- Develop new applications.
- Accumulate existing applications.
- Add new functionality to existing applications.
- Package an application.
- Prepare enterprise beans for execution in the Component Broker run time.

It is used to develop applications from start to finish. You can also design in Rational Rose, then import the design into Object Builder and add the final objects and program logic. Object Builder supports the CORBA programming model using IDL with implementations in both Java and C++. Complete working applications can be generated, including unit test versions and full client/server packages complete with server setup scripts.

The CB Tools integrate with the Object Builder to create the input to compile code and emitters. This integration defines Object Builder as the development environment for the Component Broker product. You can use it to develop your application from start to finish, or you can import architectural engineering designs into Object Builder, where you add the final objects and program logic.

You can select platforms for which to generate your application. For each platform you select, an equivalent subdirectory is added to the working directory of the application project. Every time you generate code, you generate that code for all selected platforms; however, only code that needs to be regenerated because of changes to the model since the last code generation are rebuilt.

You can define constraints to ensure that the components you develop are deployable on your target platforms. These constraints can be set when you create the object, or later by editing its properties. For example, if your platform constraints are set to AIX and OS/390, you can select to apply only OS/390 constraints to develop an OS/390-specific version of the object.

The Object Builder user interface provides access to different views of your application. To build the application DLL files defined in Object Builder, you must have the Component Broker Server software development kit (SDK) installed, as well as any prerequisite application development software.

The model for your application is constructed from components. You can use Object Builder to develop components for deployment on Windows NT, AIX, Solaris, or OS/390 servers. Most development options are the same for all platforms: the main differences appear when you generate the code for your components. For example, you can filter inheritance options, framework methods, and framework method implementations for the selected platform. The information for all platform views is stored in the same project model; you can switch between views at any time.

Component Broker also provides tools for deploying portable enterprise beans to run as Component Broker business objects. Deployment of both session beans and entity beans is supported. Command-line and GUI interfaces for bean deployment are provided, including support for deploying enterprise beans from Object Builder or from VisualAge for Java. Deployment of session beans and entity beans with bean-managed persistence can be done unattended in batch mode or from a makefile. Deployment of entity beans with CMP involves using Object Builder to define the mapping between the bean's CMP fields and a persistent data store. This mapping can be done either by using a legacy data store or by defining a new database, and can utilize the full variety of persistent back ends that are supported by

Component Broker. The mapping can also take advantage of the rich set of database mapping helpers that Component Broker provides.

For more information on Object Builder, see the Component Broker *Application Development Tools Guide*.

Chapter 8. Introduction to 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 combines the technologies of IBM's market-leading Customer Information Control System (CICS) and IBM's Encina transaction processing products. Customers can use it to create a distributed, client/server environment with the reliability, availability, and data integrity required for online transaction processing.

TXSeries supports standard protocols and gateways to the Internet. You can link your transaction environment to key applications for groupware and database management. TXSeries also enables you to run business transactions over the Internet securely and reliably; for example, you can run order entry, customer record updates, and inventory maintenance applications.

TXSeries provides transaction-based access to data stored in DB2, Oracle, Microsoft SQL Server, Informix, and Sybase relational databases.

TXSeries 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, which is part of WebSphere Enterprise Edition, runs on AIX, HP-UX, Solaris, and Windows NT systems; other versions of CICS run on OS/390, AS/400, VMS, and other platforms.

CICS handles security, data integrity, and resource scheduling. It integrates basic business software services required by online transaction processing applications. Typical transaction processing applications that use CICS include:

- Retail distribution systems
- Banking, insurance, and brokering systems
- Order entry and processing systems
- General ledger systems
- Payroll systems
- Automatic teller machines
- Airline reservation systems
- Process control systems

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

Basic CICS concepts

An instance of a CICS system is called a CICS *region*. A region is configured and administered as a unit and controls a common set of resources. Multiple CICS regions often run 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 interregion (also called intersystem) communication.

User interactions with a CICS region involve one or more transactions. In CICS terms, a *transaction* is a basic operation that is offered to the user. For example, a banking application can include a query transaction, a debit transaction, a funds-transfer transaction, and so on. This use of the term is a bit different from the use of the term in other contexts (such as within 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. 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.

CICS uses a file manager—either the Encina Structured File Server (SFS) or DB2—to store transient data and Virtual Storage Access Method (VSAM) files.

The CICS application programming interface

CICS provides a rich application programming interface (API) to enable developers to create transaction application programs. The API is made up of a number of CICS commands. The commands are embedded in an application program written in a high-level language (such as COBOL, C, C++, PL/I, or Java). The developer simply precedes the CICS command by the phrase EXEC CICS as shown in this 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:

- Reading, writing, and updating files

- Allocating and freeing memory
- Passing control between CICS programs
- Reading from and writing to temporary storage queues
- Reading from and writing to transient data queues
- Using timers
- Writing journals for audit trails, change records, and so forth
- Sending and receiving data from 3270 terminals
- Communicating using SNA LU 6.2 communications
- Controlling the CICS internal dispatcher
- Handling logical units of work
- Security and authentication
- Batched data exchange
- Monitoring and diagnostics

Commands are also provided for a number of other functions. On some platforms, CICS API commands are also implemented as C++ and Java methods.

CICS also provides *user exits*, places in CICS modules 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.

Relational database support

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. The databases can fully participate in CICS LUWs by using a full two-phase commit process if needed. CICS provides support for the following relational database management systems:

- DB2 UDB
- Oracle
- Informix
- Sybase
- Microsoft SQL Server

CICS also supports the use of VSAM.

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: transient data queues and temporary storage queues. 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.

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.

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 following CICS intercommunication facilities are available:

- *Distributed program link (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.
- *Function shipping* enables an application program to access files, transient data queues, and temporary storage queues belonging to another CICS system.
- *Transaction routing* enables transactions to be executed 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* extends the EXEC CICS START command to enable 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)* uses additional EXEC CICS commands that enable two applications running on different systems to pass information between themselves. These EXEC CICS commands map to the LU 6.2 mapped conversation verbs defined in the SNA Architecture.

  DTP can be used to communicate with non-CICS applications that use the *advanced program-to-program communications (APPC)* protocol.

CICS SNA support

CICS regions and Encina PPC-based applications can communicate across SNA with any system that supports APPC; for example, IBM mainframe-based CICS and APPC workstations.

TXSeries can use the following two methods of SNA communication:

- CICS local SNA support, which supports synchronization levels 0 and 1.
- An Encina PPC Gateway server, which supports synchronization levels 0, 1, and 2.

Both methods support all the CICS intercommunication facilities to other CICS regions, and DTP is supported to non-CICS regions (such as Encina). Also, CICS can use local SNA support to communicate with IBM CICS Universal Clients.

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. A CICS client can communicate with multiple CICS regions.

CICS Transaction Gateway

The CICS Transaction Gateway is an integration of the functionality of the CICS Internet Gateway and the CICS Gateway for Java. The gateway enables any Web browser, network computer, or Internet-enabled consumer device to access business applications running on CICS servers. When the CICS Transaction Gateway is used, a Web browser can be used to run a CICS 3270 terminal session, a Java applet, or a CORBA-based client.

The CICS Transaction Gateway runs on the Windows NT, AIX, Solaris, and OS/2 operating systems. In addition, application development is supported for the Windows 95 and 98 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 SFS, DB2, and the Distributed Computing Environment (DCE).

The administrative tool used to configure and manage CICS depends on the operating system you are using. For example, you can use the TXSeries Administration Tool on Windows NT or the System Management Interface

Tool (SMIT) for CICS on AIX. The tools simplify and automate 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.

TXSeries Encina

Encina is designed to help develop and manage open distributed systems. Encina is a family of software products for building and running large-scale, distributed client/server systems. It uses and enhances the facilities provided by DCE and CORBA. It provides support for distributed transactions across multiple platforms, using multiple resource managers. It can also interact with other transaction processing products, including CICS, and coordinate transactions that span platforms and resource managers.

Encina runs on AIX, Sun Solaris, HP-UX, and Windows NT. Using facilities such as the Encina DE-Light Gateway and Encina's support for such standards as CORBA and Microsoft COM, clients can also run on a number of other platforms and in Web browsers.

Figure 9 provides an overview of the Encina architecture. Descriptions of the components follow.

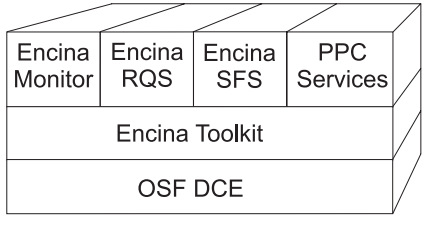


Figure 9. Architecture of Encina

Encina Monitor

The Encina Monitor provides an infrastructure for developing, running, and administering 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. The Monitor includes:

- A full-featured API that shields the programmer from the complexities of distributed computing
- A reliable execution environment that delivers load balancing, scheduling, and fault tolerance across heterogeneous environments to provide high performance and transactional integrity
- A comprehensive management environment that enables widely distributed Monitor-based systems to be administered as a single, logically defined system

The Monitor and its programming interface are described in more detail in the *Encina Monitor Programming Guide*. For more information on the administrative interfaces to the Encina Monitor, see *Encina Administration Guide Volume 1: Basic Administration*.

The Recoverable Queueing Service (RQS)

The Recoverable Queueing Service (RQS) enables applications to queue transactional work for later processing. Applications can then commit their transactions with the assurance that the queued work will be completed transactionally at a later time.

RQS supports transactional applications that must offload all or part of a task for later processing. An application can store data related to a task in a queue. This data can be subsequently processed by another program. This offloading can be desirable when use of a resource incurs an unacceptable time penalty during peak usage hours, when one part of a transaction takes much longer than other parts, or when a resource is temporarily unavailable. For example, the confirmation of a sale can be completed in real time, and the data associated with the sale can be stored in an RQS queue for later processing.

RQS programming is described in the *Encina RQS Programming Guide*. RQS administration is described in the *Encina Administration Guide Volume 2: Server Administration*.

The Structured File Server (SFS)

The SFS is a record-oriented file system that provides transactional integrity, log-based recovery, and broad scalability. Many operating systems support only byte-stream access to data: all input and output data, regardless of its source, is treated as an unformatted stream of bytes. SFS uses *structured files*, which are composed of records. The records themselves are made up of fields. For example, each record possibly contains information about an employee, with fields for the name, employee number, and salary.

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 programming is described in the *Encina SFS Programming Guide*. SFS administration is described in the *Encina Administration Guide Volume 2: Server Administration*.

The Peer-to-Peer Communications (PPC) Services

The Encina Peer-to-Peer Communications Services (PPC Services) enable Encina transaction processing systems to interoperate with systems that have System Network Architecture (SNA) LU 6.2 communications interfaces. This enables integration of mainframe and Encina environments.

PPC Services enable bidirectional communications, so that both applications and data can be shared between mainframes and Encina, with either side initiating communications. Communications are routed through a gateway server that is part of both a DCE cell and a SNA network, establishing a virtual link between an Encina and mainframe application.

PPC Services support the following interfaces:

- Distributed Program Link (DPL)
- Both the X/Open Common Programming Interface Communications (CPI-C) and the IBM System Application Architecture (SAA) CPI-C
- SAA Common Programming Interface Resource Recovery (CPI-RR)

These communications interfaces and this distributed transaction processing model operate within the Encina environment.

PPC programming is described in the *Encina PPC Services Programming Guide*. PPC administration is described in the *Encina Administration Guide Volume 2: Server Administration*.

The DE-Light Gateway

The DCE Encina Lightweight Client™ (DE-Light) is a set of APIs and a gateway server that you can use to extend the power of DCE and Encina to personal computers and other systems that are not running as DCE clients.

DE-Light enables you to access DCE and Encina from systems that do not support DCE, but that do support Java. DE-Light clients require less effort to create, are simpler to administer, and do not use the network resource require by DCE or Encina clients. Yet DE-Light clients can still take advantage of the benefits that were formerly available only to full DCE and Encina clients.

DE-Light is composed of the following:

- A 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).
- A C API used to develop clients for the Microsoft Windows NT and Windows 98 environments. DE-Light C clients use TCP/IP to communicate with gateways at known endpoints.
- A DE-Light Gateway server that enables communications between DE-Light clients and DCE or Encina.

DE-Light Java supports two types of clients:

- A Java applet embedded into a Hypertext Markup Language (HTML) document residing on a Web server.
- A standalone Java application.

DE-Light programming is described in the *Encina DE-Light Programming Guide*. DE-Light Gateway administration is described in the *Encina Administration Guide Volume 2: Server Administration*.

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.

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

Encina++

Encina++ is an object-oriented API for Encina. It is composed of classes that access many Encina components. Encina++ supports the development of object-oriented applications that are based on DCE (Encina++/DCE), CORBA (Encina++/CORBA), and a mixture of both DCE and CORBA (Encina++/CORBA-DCE). The latter type of application is often referred to as a mixed application.

Table 3 lists the interfaces that make up Encina++. Some can be used only with Encina++/DCE or only with Encina++/CORBA, and some can be used by both.

Table 3. Components of Encina++

Interface	Purpose	Language	Used with

Table 3. Components of Encina++ (continued)

Encina++	Creates and manages client/server applications.	C++	DCE and CORBA
Transactional-C++ (Tran-C++)	Distributed transaction processing.	C++	
Object Management Group Object Transaction Service (OMG OTS)	Distributed transaction processing; implements the OMG *Object Transaction Service* specification (*OMG document 94.8.4*).	C++	
RQS++	Transactionally enqueues and dequeues data at an RQS server	C++	DCE only
SFS++	Manipulates data stored at an SFS server.	C++	
Object Concurrency Control Service (OCCS)	Enables multiple clients to coordinate access to shared resources; implements the OMG *Concurrency Control Service Proposal*	C++	CORBA only
Java OTS Client	Enables Java clients to use distributed transactions; implements the OMG *Object Transaction Service* specification.	Java	

For more information on the various Encina++ interfaces, see the following documents:

- *Encina Object-Oriented Programming Guide*
- *Encina Transactional Programming Guide*
- *Encina RQS++ and SFS++ Programming Guide*

Encina tools available only on Windows platforms

On both Windows NT and Windows 95/98 systems, Encina comes with additional programming and diagnostic tools that are not available on other platforms. For more information on how to use these tools to create Encina applications, see *Writing Encina Applications on Windows Systems*.

Programming tools

The following Encina tools aid developers in the creation of distributed client/server applications:

- Encina Server Wizard—This wizard, which can be used to create Encina and Encina++ servers, 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.
- Encina COM Wizard—This 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.

WinTrace

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.

Interoperability with WebSphere Application Server Advanced Edition

Encina applications can interoperate with Advanced Edition applications. Java-based Advanced Edition applications can act as clients to Encina servers. They can connect to Encina servers by using a specialized Monitor application server that acts as a bridge between Java-based systems and Encina/DCE systems, Encina++/DCE systems, or both. The bridge server propagates transactions originating at the client to the server.

Figure 10 illustrates the general architecture of a distributed system that uses a bridge server.

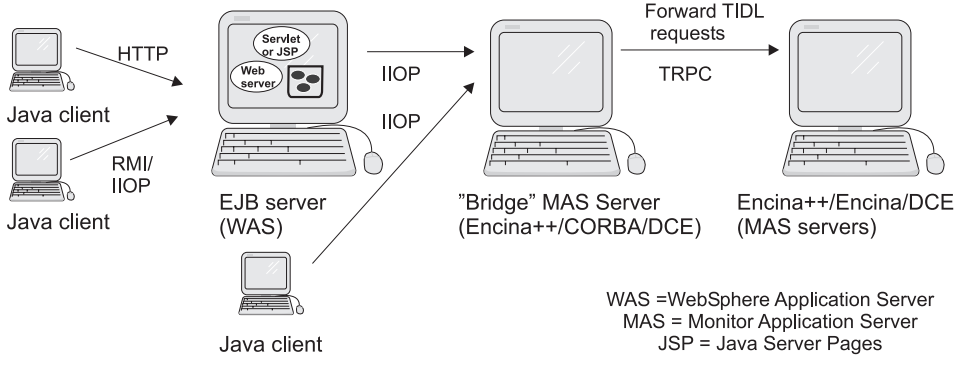


Figure 10. Interoperability between Java applications and Encina/Encina++ servers

Java-based clients can access Encina/Encina++/DCE applications with minimum development effort. Encina provides a tool (the **wstidl** command) that takes a TIDL file as input and generates the required CORBA IDL interfaces, Java classes, server main function, and other files used to achieve connectivity. The generated Java bean or enterprise bean can be used as part of Advanced Application Server applications. For example, an enterprise bean in a WebSphere EJB server can use a generated bean (or a generated enterprise bean) to communicate with the Encina application.

For an example of an application that uses the Encina bridge server, see *Building Business Solutions with WebSphere*.

Chapter 9. Sample topologies and configurations

This section discusses typical examples of topologies and configurations for WebSphere Application Server. It is intended to be a sampling of configurations that can be set up using the WebSphere Application Server Standard, Advanced, and Enterprise editions, not an exhaustive set of configurations.

The following examples are included:

For a detailed look at other WebSphere Application Server topologies, see *Building Business Solutions with WebSphere*.

Client topologies

Traditional client/server models place the logical second tier onto the physical first tier — that is, the client implements some or all of the application's business logic. This architecture often results in clients that require larger and more powerful desktop machines than those required by clients in other distributed architectures. (This type of client is sometimes called a *fat* client.) In addition to requiring more expensive equipment, maintaining the business logic directly on the user's desktop can be difficult and expensive to administer.

Many designers of business information systems are implementing computing models in which the essential business logic is more centrally maintained. This approach is easier to administer and enables the information system to be updated more quickly. WebSphere Application Server can be used to implement various types of clients where business processes are handled more centrally.

Thick client

A *thick* client depends on a local application for its user interface. Its business logic is handled on a remote server in a distributed computing environment.

For instance, a thick client can use desktop applications such as word processors and spreadsheets to provide a user interface.

Figure 11 shows a typical thick client. This figure depicts only the user interface and business logic portions of the architecture.

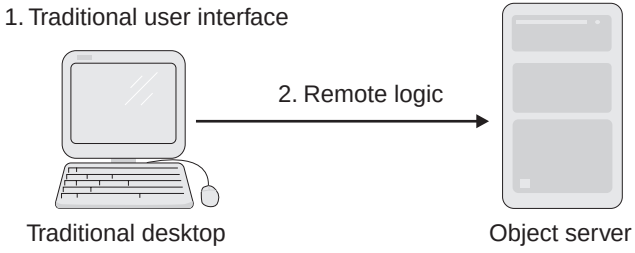


Figure 11. A thick client has a local user interface and remote business logic

Using a thick client to implement a business solution has less administrative overhead than using a traditional client. The application's business logic is centrally located and easier to maintain. However, a thick client still requires a relatively powerful desktop computer.

Thin client

Using a Java applet can reduce client size. Figure 12 shows how this type of *thin* client can be implemented. In this example, the applet is downloaded over a corporate intranet into a browser where it runs locally on the client machine.

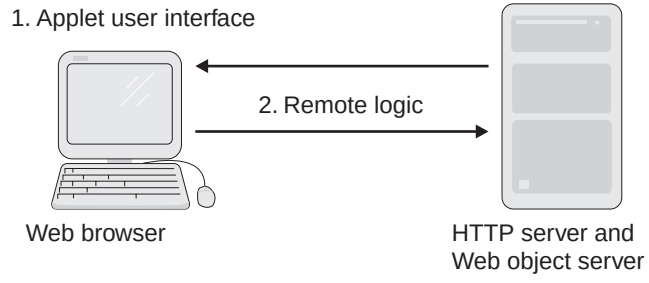


Figure 12. A thin client has an applet user interface and remote business logic

Thin clients make more efficient use of computer resources since a large number of high-powered machines are not needed to run the client. They also have less administrative overhead than thick clients. A fresh copy of the applet is downloaded every time it is accessed, which makes updating the client automatic. This is a huge advantage in an environment where many machines are running a client. Thin clients also do not need to run on a powerful desktop computer. The only requirement is that the client machine's

browser must support the Java Virtual Machine (JVM) level required by the applet. Thin client applets can have longer download times than other types of clients, however, since the client user interface must be downloaded over the network.

Thinner client

A thin client can be made even thinner by using HTML with JavaScript, as shown in Figure 13. User requests are submitted to a Java servlet. The servlet initiates remote processing requests from the Web server and dynamically generates new HTML pages as required.

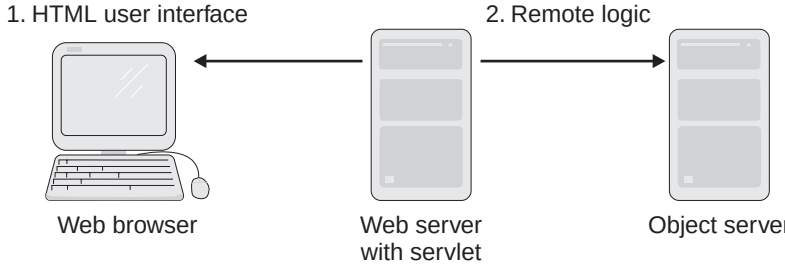


Figure 13. A thinner client used with a servlet

The thinner client described in Figure 13 has a much shorter download time than a thin client implemented with a Java applet. The thinner client also does not use JVM, eliminating incompatibility problems in this area. It retains the applet benefits of reduced hardware requirements and reduced client administration. At the same time, it can improve client performance and reliability. The use of HTTP between the browser and Web server also enables clients to securely access the servlet through a firewall.

The drawback of using a thinner client is that it offers only a simplified set of client functions. In particular, it cannot make use of Java's sophisticated user interface functions. However, it is a good choice for clients that do not require a complex user interface.

Server topologies

The thick, thin, and thinner client examples focused on the first and second tiers of a three-tiered distributed computing architecture. WebSphere Application Server allows you to organize the servers and resources of the second and third tiers along several different models. The client topologies described in "Client topologies" on page 79 can be used with all of these server models.

Distributed servers

The Advanced and Enterprise Application Servers support distributed servers. Figure 14 shows several middle-tier and back-end machines in a physically distributed network. In this example, the machines in the middle tier have smaller, slower, and less expensive processors than those in the third tier. If a sufficient number of these smaller servers are clustered together, the middle tier can gain enough processing power to handle its workload. This includes compensating for the overhead imposed by distributed network traffic. Clusters also improve the availability of servers because processing can be done by other machines if a member of the cluster goes offline. Another benefit of clustered servers is that computing capacity can be added incrementally without disrupting clients.

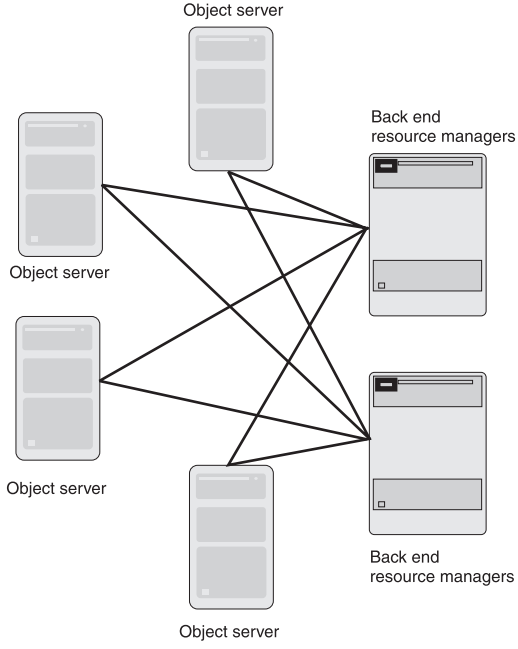


Figure 14. A cluster of distributed servers connected to resource managers

For this approach to work, the servers must be easily replicated and have transparent, manageable load balancing and failure support. In addition, the business logic that runs in the middle-tier environment must be consistent across the cluster.

Cloned servers

In this topology, each machine in the middle tier hosts an exact replica (or clone) of a server. Cloning servers improve the availability of servers because processing can be done by other machines if one of the servers goes offline.

The Advanced and Enterprise Application Servers provide a workload management service to evenly distribute processing requests across the clones of a server and provide failover services if a server unexpectedly goes offline. Workload management applies to administrative servers as well as application servers, ensuring that there are no disruptions to management of the application server cluster.

Consolidated servers

Many companies limit their physical architecture to two tiers, regardless of whether they implement a three-tiered logical architecture. The operational costs of a three-tiered physical architecture can be very high. In Figure 15, the second and third tiers are physically consolidated.

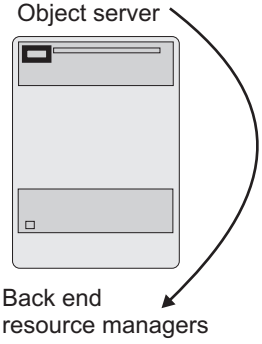


Figure 15. Physical consolidation of the middle and back-end tiers

This configuration improves performance by reducing remote communications and eliminating the overhead of network traffic between the middle tier application server and its associated back-end systems. Its centralized implementation is easier to administer than clustered servers.

Increasing the scale of a system that implements a consolidated server architecture depends on the computing platform on which it is implemented. High-end platforms (such as the IBM OS/390) that use this architecture can easily scale up to large business systems.

Standard Application Server topology

The Standard Application Server is a single-machine version of WebSphere Application Server. Figure 16 on page 84 shows one example of how the Standard Application Server can be configured.

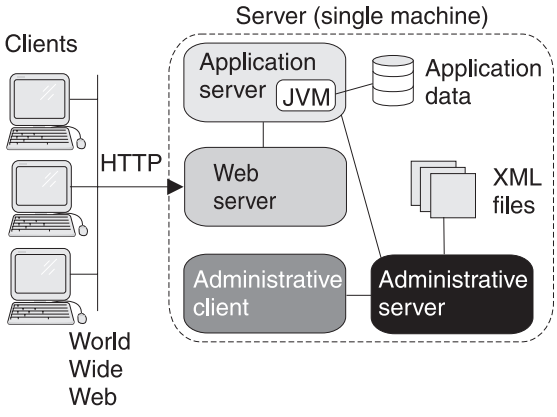


Figure 16. A simple Standard Application Server configuration

Clients access the Web server via HTTP requests over the World Wide Web. Processing requests are referred to the application server, which accesses a database that is located on the same machine as the application server. The administration server and administration client manage the Standard Application Server environment.

Advanced Application Server topologies

The Advanced Application Server can be configured in a wide variety of topologies depending on the needs of an organization. The following sections illustrate some sample topologies for the Advanced Application Server.

Simple configuration

A simple Advanced Application Server configuration is shown in Figure 17 on page 85.

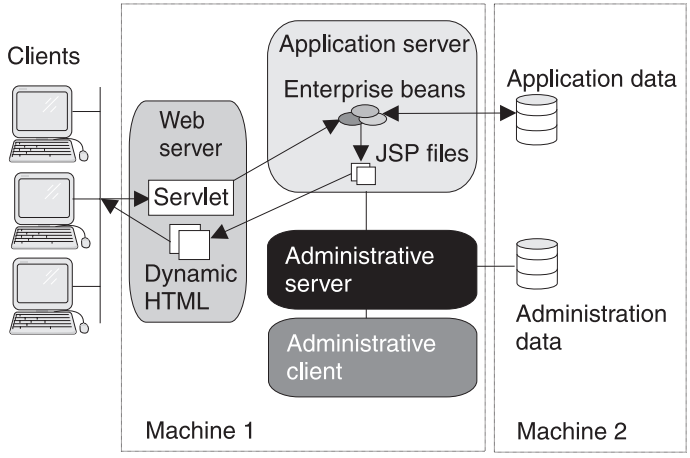


Figure 17. Simple Advanced Application Server configuration

Clients access a Web-based business application hosted on the Advanced Application Server. For simplicity's sake, a single application server is shown in this configuration; however, the application can be hosted on multiple servers.

The client accesses a servlet that runs on the Web server. The servlet relays client processing requests to enterprise beans that transactionally access a database on another machine. (Administration data is also stored in a database on this machine.) The results of the operation are displayed to the client using HTML that is dynamically generated via JavaServer Pages (JSP) files.

DMZ configuration

The demilitarized zone (DMZ) topology, shown in Figure 18 on page 86, provides enhanced security for business applications that are accessed from the Internet.

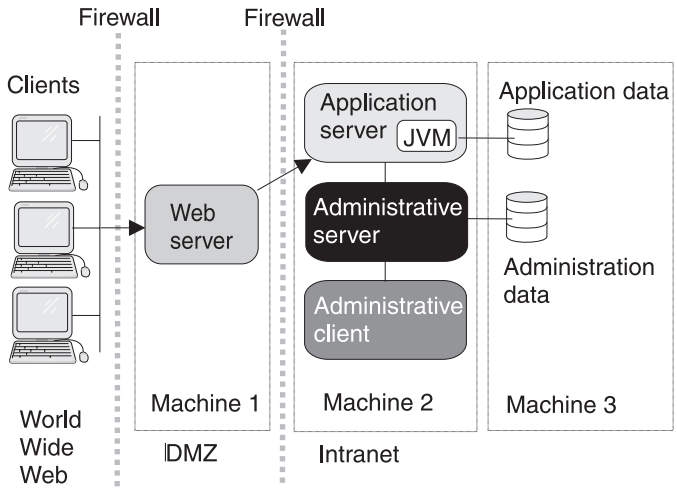


Figure 18. DMZ configuration in the Advanced Application Server

In this configuration, the Web server is isolated by firewalls both from its clients and from the application servers, administration servers, and back end data stores that the system uses. Clients accessing the application from the Internet must pass though the secure Web server in the DMZ in order to gain access to the application servers, databases, and other resources on the organization's protected intranet. This minimizes the risk to sensitive enterprise resources.

TXSeries configurations

The following sections illustrate some example configurations for CICS and Encina.

A simple CICS configuration

A simple distributed CICS environment has a CICS client on one machine and a CICS region on another machine, as shown in Figure 19 on page 87. This configuration is recommended for first-time CICS installations because it is the easiest to install, test, and maintain.

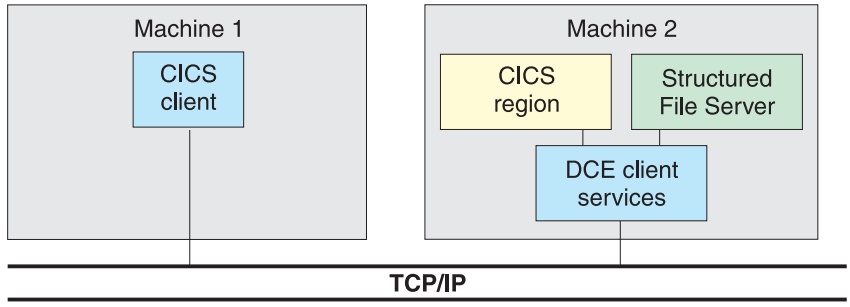


Figure 19. A simple distributed configuration using CICS in a non-DCE cell environment

The example configuration shows CICS in an RPC-only environment. In an RPC-only environment, internal CICS security and directory services are used. The DCE RPC for endpoint mapping and transmitting transactional data is the only DCE service used.

The example configuration shows the following:

- The SFS server is used for CICS region files and queues, and can be used to store user data.
- Communications between the CICS region and the SFS server use DCE RPCs provided by the DCE RPC daemon. Other DCE client services are not used.
- The CICS client provides immediate 3270 terminal access and program access to the CICS region.

A simple CICS configuration within a DCE cell

A simple distributed CICS environment within a Distributed Computing Environment (DCE) cell consists of a client on one machine and a CICS region running on another machine, as shown in Figure 20 on page 88. This configuration requires the DCE CDS and DCE Security Service to be installed on machines in the DCE cell.

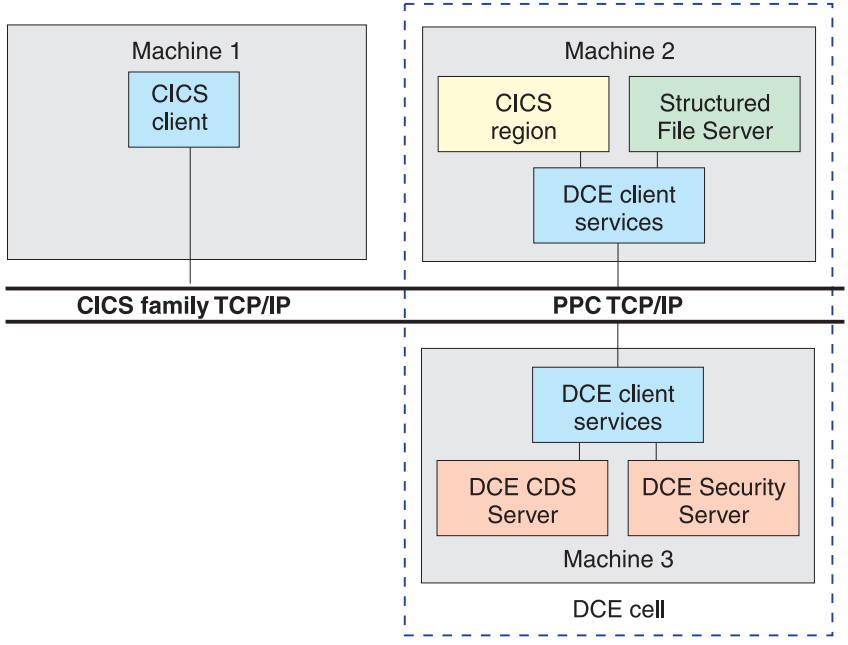


Figure 20. A distributed configuration using CICS in a DCE cell environment

The example configuration shows the following:

- Server location and security services are provided by DCE.
- The SFS server is used for CICS region files and queues, and can be used to store user data.
- The DCE RPCs used to communicate between the CICS region and the SFS server can be authenticated by the DCE Security Server.
- The CICS client provides immediate 3270 terminal access and program access to the CICS region.

Note: The CICS client machine does not have to be part of the DCE cell, unless it also runs a CICS region that uses the DCE cell services.

A simple Encina Monitor cell configuration

A simple Encina Monitor cell configuration, shown in Figure 21 on page 89, contains the following:

- A cell manager, which coordinates the activity of node managers in the cell
- A node manager, which controls the activity of all servers running on the node (machine)
- A Monitor application server, which provides the business logic of an Encina application and acts on data stored in an SFS server

- A Monitor client, which provides the presentation logic of an Encina application

The Monitor cell is part of a DCE cell, which contains another machine on which the DCE CDS and DCE Security Service is installed.

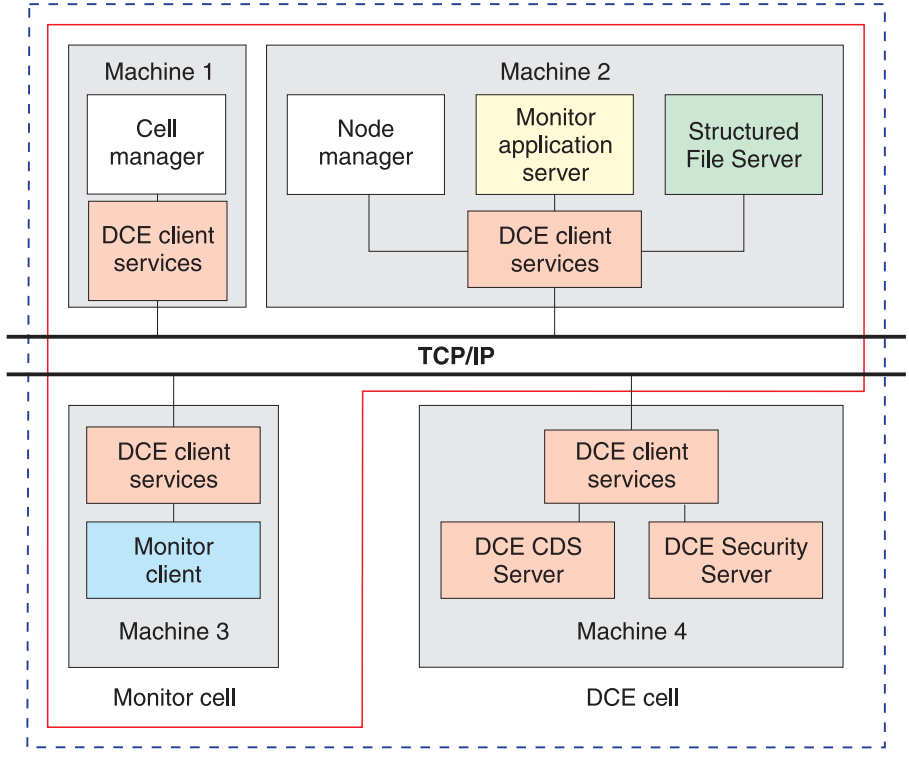


Figure 21. A simple Encina Monitor configuration

Server location and security services are provided by DCE. The DCE RPCs used to communicate between the Encina servers and client can be authenticated by the DCE Security Service.

Component Broker configurations

The following illustrates some example configurations for Component Broker.

Simple configuration

The simple Component Broker topology shown in Figure 22 on page 90 runs all business objects on the same host using a single application server.

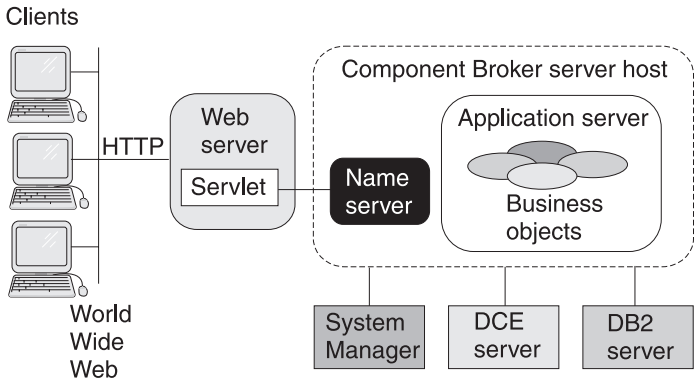


Figure 22. Basic Component Broker topology

In addition to the Component Broker host server, this topology requires three supporting servers. The System Manager runs the Component Broker installation and manages the Component Broker server that hosts the application server. The DCE server provides underlying services. The DB2 server represents a heavily used third-tier resource manager; other resource managers that can be configured on this tier include IMS and CICS.

Basic workload management configuration

The topology shown in Figure 23 on page 91 represents a basic workload management configuration. The business objects still run on one application server, but multiple copies of this server are running on different machines in the network.

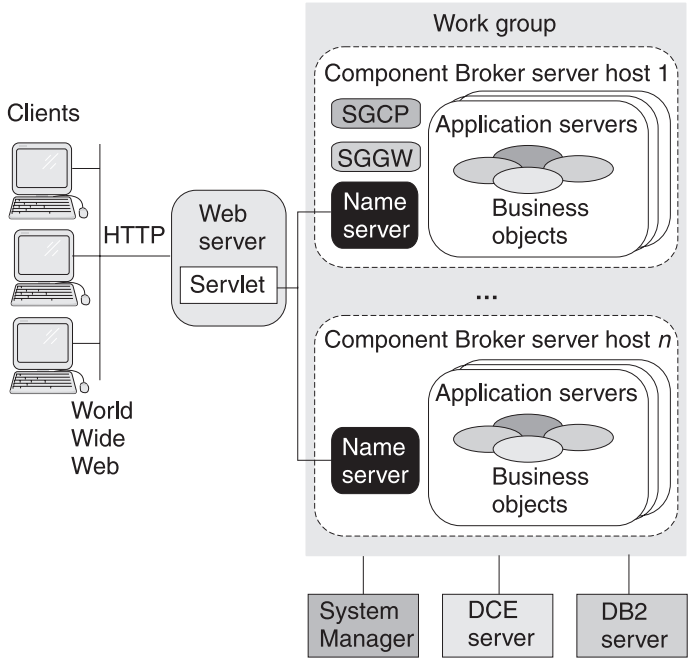


Figure 23. Horizontal workload management topology for Component Broker

Two Component Broker servers are configured into a work group. Multiple application servers run on each Component Broker server.

Increasing the number of application servers improves the ability of the network to support applications. Client requests can be distributed between the application servers in the workload management group, making the server group appear as a single logical server to the client.

When workload management is enabled by using a controlled server group, two additional servers must be located on one of the hosts in the network:

- Server Group Control Point (SGCP)
- Server Group Gateway (SGGW)

These servers provide underlying support for Component Broker's workload management service.

Appendix. The library for WebSphere Application Server

The following table lists the complete documentation library for WebSphere Application Server. All documents are available online from the WebSphere Application Server library page at www.software.ibm.com/webservers/appserv/library.html.

Table 4. The library for WebSphere Application Server

Form number	Document name	Document description
Common documentation for WebSphere Application Server (for all supported platforms)		
SC09-4581	*Getting Started with WebSphere Application Server*	Provides a common familywide overview of all editions of WebSphere Application Server and the contents of each edition. Formerly titled *Introduction to WebSphere Application Server*.
SC09-4431	*Writing Enterprise Beans in WebSphere*	Provides an introduction to programming with Enterprise JavaBeans™ components in the Advanced and Enterprise Editions of WebSphere Application Server.
SC09-4432	*Building Business Solutions with WebSphere*	Provides programming examples and scenarios that demonstrate application development and recommended programming practices across the WebSphere Application Server family. It also discusses other products in the WebSphere family.
GC09-3951	*Using the Performance Monitoring Infrastructure Client Package*	Describes how to use the Performance Monitoring Infrastructure (PMI) interface to write WebSphere Application Server clients that gather, process, and display performance information from WebSphere servers.
GC09-3952	*Using the JRas Message Logging and Trace Facility*	Describes how to use the WebSphere JRas interface to incorporate message and logging facilities into WebSphere Application Server applications.
GC09-4550	*Using the WorkArea Facility*	Describes how to use the WorkArea facility to set, retrieve, and manage scoped contexts in WebSphere Application Server applications.
WebSphere Application Server Standard and Advanced Editions documentation (for all supported platforms)		
	InfoCenter	Provides information on installing, configuring, maintaining, and programming with the Standard and Advanced Editions of WebSphere Application Server.
Common Component Broker documentation (for all supported platforms)		

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
SC09-4442	*Programming Guide*	Provides information on developing Component Broker applications in all supported programming languages on all supported platforms. It describes the programming model, including business objects and data objects, and includes information about various basic programming issues.
SC09-4443	*Advanced Programming Guide*	Describes Component Broker's implementation of the Common Object Request Broker Architecture (CORBA) Object Service; the Component Broker Object Request Broker (ORB); Cache, Notification, Query, Session, and other Services; interlanguage object model (IOM); and workload management.
SC09-4444	*MQSeries Application Adaptor Development Guide*	Provides information about developing Component Broker applications that use the MQSeries Application Adaptor.
SC09-4572	*Procedural Application Adaptor Development Guide*	Provides information about developing Component Broker applications that use the Procedural Applicaton Adaptor.
SC09-4573	*Database Application Adaptors Development Guide*	Provides information about developing Component Broker applications that use the DB2, Oracle, and Informix Application Adaptors.
SC09-4445	*System Administration Guide*	Provides information on administering Component Broker and Component Broker applications on all supported platforms. Also provides general information about using the System Manager interface.
SC09-4588	*Programming Reference*	Documents the complete Component Broker application programming interfaces available for all supported programming languages.
SC09-4448	*Application Development Tools Guide*	Provides information about using the Component Broker Toolkit components, with a focus on common development scenarios such as inheritance and team development.
SC09-4449	*Problem Determination Guide*	Provides information to help administrators and programmers identify and solve problems with Component Broker and Component Broker applications. It includes information on installation problems, run-time errors, debugging of applications, and analysis of log messages.
SC09-4450	*Glossary*	Lists and defines terms commonly used in Component Broker documentation.

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
GC09-4490	*Component Broker Release Notes*	Provides platform- and release-specific information about Component Broker, including late-breaking functional changes; known restrictions; and, where possible, suitable workarounds for restrictions.
	Distributed Debugger for Workstations	Contains information about the Distributed Debugger program on workstations.
	Distributed Debugger for OS/390	Contains information about the Distributed Debugger program on OS/390.
	Object Level Trace	Contains information about Object Level Trace.
Component Broker for AIX documentation		
SC09-4437	*Planning, Performance, and Installation Guide for AIX*	Provides complete instructions for installing, configuring, and upgrading to the latest version of Component Broker on AIX.
Component Broker for Solaris documentation		
SC09-4438	*Planning, Performance, and Installation Guide for Solaris*	Provides complete instructions for installing and configuring the latest version of Component Broker on Solaris.
Component Broker for Windows systems documentation		
SC09-4436	*Planning, Performance, and Installation Guide for Windows Systems*	Provides complete instructions for installing, configuring, and upgrading to the latest version of Component Broker on Windows systems.
Component Broker for OS/390 documentation		
GA22-7325	*OS/390 Component Broker Installation and Customization*	Describes the planning and installation considerations for Component Broker on the OS/390 platform.
GA22-7326	*OS/390 Component Broker Programming: Assembling Applications*	Provides information for assembling applications using Component Broker on the OS/390 platform.
GA22-7331	*WebSphere Application Server for OS/390 Concepts and Planning*	Describes planning considerations for WebSphere Application Server on the OS/390 platform.
GA22-7328	*OS/390 Component Broker Operations and Administration Guide*	Provides information for operating and administering Component Broker on the OS/390 platform.
GA22-7329	*OS/390 Component Broker Messages and Diagnosis*	Provides diagnosis information and describes the messages associated with Component Broker on the OS/390 platform.
SC33-6587	*OS/390 Component Broker System Management User Interface*	Describes the System Management user interface provided with Component Broker on the OS/390 platform.

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
GA22-7478	*OS/390 System Management Scripting Application Programming Interface*	Describes the System Management scripting interface provided with Component Broker on the OS/390 platform.
GA22-7460	*WebSphere Application Server for OS/390 Program Directory*	Describes the WebSphere Application Server for OS/390 program directory.
GA22-7462	*WebSphere Application Server for OS/390 Licensed Program Specifications*	Describes the WebSphere Application Server for OS/390 licensed program specifications.
Common TXSeries (CICS and Encina) documentation (for all supported platforms)		
SC09-4582	*Concepts and Planning*	Introduces the TXSeries product, providing high-level descriptions of transaction processing, CICS, and Encina.
GC09-4491	*TXSeries Release Notes*	Provides platform- and release-specific information about TXSeries, including descriptions of new features that are more thorough than those in the TXSeries README file, information for features or changes learned too late for incorporation into the product documentation, descriptions of defects fixed since the last release of the product, and information about known restrictions associated with TXSeries and, where possible, suitable workarounds.
TXSeries for AIX documentation		
SC09-4452	*Planning and Installation Guide for AIX*	Provides complete instructions for installing, configuring, and upgrading to the latest version of TXSeries (CICS and Encina) on AIX.
TXSeries for Solaris documentation		
SC09-4453	*Planning and Installation Guide for Solaris*	Provides complete instructions for installing, configuring, and upgrading to the latest version of TXSeries (CICS and Encina) on Solaris.
TXSeries for Windows systems documentation		
SC09-4451	*Planning and Installation Guide for Windows Systems*	Provides complete instructions for installing, configuring, and upgrading to the latest version of TXSeries (CICS and Encina) on Windows systems.
TXSeries for HP-UX documentation		
SC09-4583	*Planning and Installation Guide for HP-UX*	Provides complete instructions for installing, configuring, and upgrading to the latest version of TXSeries (CICS and Encina) on HP-UX.
CICS documentation (for all supported platforms unless otherwise noted)		

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
SC09-4587	*CICS Administration Guide for Open Systems*	Provides guide information for administering CICS and CICS applications on UNIX platforms. It includes information on the Tivoli interface for CICS. It also includes the CICS glossary.
SC09-4456	*CICS Administration Guide for Windows Systems*	Provides guide information for administering CICS and CICS applications on Windows systems. It includes information on the Tivoli interface for CICS. It also includes the CICS glossary.
SC09-4459	*CICS Administration Reference*	Provides complete reference information for commands used to administer CICS on all supported platforms.
SC09-4460	*CICS Application Programming Guide*	Provides information for developing CICS-based applications in all supported programming languages on all supported platforms.
SC09-4461	*CICS Application Programming Reference*	Provides reference information for the CICS application programming interfaces for all supported languages on all supported platforms.
SC09-4462	*CICS Intercommunication Guide*	Describes how to implement communications between a CICS region and other systems (for example, another CICS region on a UNIX or Windows machine or another application on a system such as a mainframe).
SC09-4589	*CICS Messages and Codes*	Lists and describes all messages and codes that can be issued by a TXSeries CICS system.
SC09-4465	*CICS Problem Determination Guide*	Helps administrators identify and diagnose problems with a CICS system or application. It describes symptoms of problems and their possible causes.
SC09-4466	*Using CICS Workload Management*	Describes the CICS Workload Management utility.
SC09-4585	*CICS IIOP and Java Programming Guide*	Describes the CICS Internet Inter-ORB Programming (IIOP) interface, which enables CICS applications to communicate with Common Object Request Broker Architecture (CORBA) and Java Object Request Brokers (ORBs). It also describes how to use the Java application programming interface for CICS.
SC09-4468	*CICS Front-End Programming Interface for Windows NT*	Provides information about developing applications that use the CICS front-end programming interface (FEPI) for Windows NT.
SC09-4469	*Using IBM Communications Server for AIX with CICS*	Provides information for using CICS with the Systems Network Architecture (SNA) package provided by IBM Communications Server for AIX.

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
SC09-4470	*Using IBM Communications Server for Windows Systems with CICS*	Provides information for using CICS with the Systems Network Architecture (SNA) package provided by IBM Communications Server for Windows systems.
SC09-4471	*Using Microsoft SNA Server with CICS*	Provides information for using CICS with the Systems Network Architecture (SNA) package provided by Microsoft for Windows systems.
SC09-4472	*Using SNAP-IX for Solaris with CICS*	Provides information for using CICS with the SNAP-IX Systems Network Architecture (SNA) package provided by Data Connection Limited (DCL) for Solaris.
SC09-4586	*Using HP-UX SNAplus2 with CICS*	Provides information for using CICS with the SNAplus2 Systems Network Architecture (SNA) package provided by Hewlett-Packard.
Encina administrative guide documentation (for all supported platforms)		
SC09-4473	*Encina Administration Guide Volume 1: Basic Administration*	Provides basic information about administering Encina and Encina applications on all supported platforms. It describes the use of the Enconsole administrative interface and includes the complete Encina glossary. It also includes information on the Tivoli interface for Encina.
SC09-4474	*Encina Administration Guide Volume 2: Server Administration*	Describes administration of Structured File Server (SFS), Recoverable Queueing Service (RQS), Peer-to-Peer Communications (PPC) Gateway, and DCE Encina Lightweight Client (DE-Light) Gateway servers.
SC09-4475	*Encina Administration Guide Volume 3: Advanced Administration*	Describes advanced administration topics such as the Encina object hierarchy and the command-line scripting interface, **enccp**, that can be used with Encina. It also includes an appendix that describes the environment variables used by all Encina components.
Encina programming guide documentation (for all supported platforms)		
SC09-4476	*Encina COBOL Programming Guide*	Describes the COBOL application programming interface available with Encina.
SC09-4477	*Encina Monitor Programming Guide*	Describes how to program with the Encina Monitor application programming interface.
SC09-4478	*Encina Object-Oriented Programming Guide*	Describes how to program with the Encina object-oriented (Encina++) application programming interfaces used to develop applications in either a Distributed Computing Environment (DCE) or Common Object Request Broker Architecture (CORBA) environment.

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
SC09-4479	*Encina RQS++ and SFS++ Programming Guide*	Describes how to program with the Encina object-oriented application programming interfaces for the Structured File Server (SFS) and Recoverable Queueing Service (RQS) components.
SC09-4480	*Encina DE-Light Programming Guide*	Describes how to program to the DCE Encina Lightweight Client (DE-Light) application programming interfaces. These interfaces provide C- and Java-based means of communicating with a DE-Light Gateway server to access Distributed Computing Environment (DCE) and Encina applications from low-end machines incapable of supporting these resource-intensive systems.
SC09-4481	*Encina PPC Services Programming Guide*	Describes how to program to the Encina Peer-to-Peer Communications (PPC) Executive application programming interface.
SC09-4482	*Encina RQS Programming Guide*	Describes how to program to the Encina Recoverable Queueing Service (RQS) application programming interface.
SC09-4483	*Encina SFS Programming Guide*	Describes how to program to the Encina Structured File Server (SFS) application programming interface.
SC09-4484	*Encina Toolkit Programming Guide*	Introduces and describes how to program to the various Encina Client (Executive) and Server Core application programming interfaces.
SC09-4485	*Encina Transactional Programming Guide*	Describes how to program with the the Encina Transactional-C (Tran-C) application programming interface.
SC09-4486	*Writing Encina Applications*	Provides an introduction to creating an Encina application. The document takes a tutorial-like approach to the development of an application with various Encina application programming interfaces. It includes general information on Encina terminology and application development.
SC09-4487	*Writing Encina Applications on Windows Systems*	Provides information on using the TXSeries application development toolkit (ADK) to develop Encina applications on Windows systems. The ADK also provides support for the Microsoft COM interface, which this document describes.
SC09-4488	*Encina Messages and Codes*	Lists and describes all messages and status codes that can be issued by Encina. The information is designed to help administrators and developers understand and correct problems with Encina or Encina applications.
Encina administrative reference documentation (for all supported platforms)		

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
GC09-4492	*Introductory Administrative Page*	Introduces the Encina administrative interfaces. It provides high-level descriptions of the various interfaces, their uses, and their syntax conventions.
GC09-4493	*The drpcadmin Command Pages*	Describes the **drpcadmin** suite of administrative commands. These commands are used to administer DCE Encina Lightweight Client (DE-Light) Gateway servers.
GC09-4494	*The emadmin Command Pages*	Describes the **emadmin** suite of administrative commands. These commands were previously used to manipulate the object hierarchy provided with Encina. Their functionality has been superseded by the **enccp** interface, and they will become obsolete with this or a future release of Encina.
GC09-4495	*The enccp Introductory Page*	Introduces the Encina control program (**enccp**) command-line and scripting interface, introducing the commands in the **enccp** interface, describing the common syntax of the commands, and generally augmenting the information provided on the other **enccp** reference pages.
GC09-4496	*The enccp Example Pages*	Provides examples of the commands in the **enccp** interface.
GC09-4497	*The enccp Object Pages*	Describes the objects that make up the Encina Monitor object hierarchy. This hierarchy replaces the previous set of objects manipulated by the **emadmin** suite of commands.
GC09-4498	*The enccp Operation Pages*	Describes the operations (commands) available in the **enccp** interface. The commands are used to manipulate the Encina Monitor object hierarchy.
GC09-4499	*Miscellaneous Administrative Reference Pages*	Describes the miscellaneous (nonsuite) commands available with Encina, including the **ecm** (Encina cell manager) startup command; the **enm** (Encina node manager) startup command; the **rqs**, **sfs**, **ppcgwy**, and **drpcgwy** startup commands; the Encina restart scripts; and the Encina command-line tracing utilities.
GC09-4500	*The otsadmin Command Pages*	Describes the **otsadmin** suite of administrative commands. These commands are used to provide Toolkit-like administration of servers based on the Encina Object Transaction Service (OTS) programming interfaces.
GC09-4501	*The ppcadmin Command Pages*	Describes the **ppcadmin** suite of administrative commands. These commands are used to administer a Peer-to-Peer Communications (PPC) Gateway server and its interaction with the Systems Network Architecture (SNA).

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
GC09-4502	*The rqsadmin Command Pages*	Describes the **rqsadmin** suite of administrative commands. These commands are used to administer a Recoverable Queueing Service (RQS) server and the data it contains.
GC09-4503	*The sfsadmin Command Pages*	Describes the **sfsadmin** suite of administrative commands. These commands are used to administer a Structured File Server (SFS) server and the data it contains.
GC09-4504	*The tkadmin Command Pages*	Describes the **tkadmin** suite of administrative commands. These commands are used to administer Encina Toolkit servers. These commands enable administration of data and log volumes, server tracing, transactional activities, and additional aspects of Toolkit servers.
Encina programming reference documentation (for all supported platforms)		
GC09-4505	*Introductory C Programming Page*	Introduces the C programming language application programming interfaces available with Encina. It also describes some high-level issues associated with programming to the C interfaces.
GC09-4506	*Abort Facility Programming Pages*	Describes the C-language-based Abort Facility application programming interface available with Encina. This interface is part of the Toolkit Executive.
GC09-4507	*DE-Light C Programming Pages*	Describes the C-language-based application programming interface available with the DCE Encina Lightweight Client (DE-Light).
GC09-4508	*Distributed Transaction Service (TRAN) Programming Pages*	Describes the C-language-based Distributed Transaction Service (TRAN) application programming interface available with Encina. This interface is part of the Toolkit Executive.
GC09-4509	*EMA Programming Pages*	Describes the C-language-based Encina Monitor Administrative (EMA) application programming interface available with Encina. This interface was previously used to manipulate the object hierarchy provided with Encina. It will become obsolete with this or a future release of Encina.
GC09-4510	*Lock Service (LOCK) Programming Pages*	Describes the C-language-based Lock Service (LOCK) application programming interface available with Encina. This interface is part of the Toolkit Server Core.
GC09-4511	*Log Service (LOG) Programming Pages*	Describes the C-language-based Log Service (LOG) application programming interface available with Encina. This interface is part of the Toolkit Server Core.

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
GC09-4512	*Miscellaneous C Programming Pages*	Describes miscellaneous C-language-based functions available with Encina, including the transactional interface definition language (**tidl**) compiler and miscellaneous conversion functions.
GC09-4513	*Monitor Programming Pages*	Describes the C-language-based Monitor application programming interface available with Encina.
GC09-4514	*PPC Executive Programming Pages*	Describes the C-language-based Peer-to-Peer Communications (PPC) Executive application programming interface available with Encina.
GC09-4515	*Recovery Service (REC) Programming Pages*	Describes the C-language-based Recovery Service (REC) application programming interface available with Encina. This interface is part of the Toolkit Server Core.
GC09-4516	*Restart Service Programming Pages*	Describes the C-language-based Restart Service application programming interface available with Encina. This interface is part of the Toolkit Server Core.
GC09-4517	*RQS Programming Pages*	Describes the C-language-based Recoverable Queueing Service (RQS) application programming interface available with Encina.
GC09-4518	*SFS Programming Pages*	Describes the C-language-based Structured File Server (SFS) application programming interface available with Encina.
GC09-4519	*T-ISAM Programming Pages*	Describes the C-language-based Transactional Indexed Sequential Access Method (T-ISAM) application programming interface available for use with the Structured File Server (SFS) component of Encina.
GC09-4520	*ThreadTid Programming Pages*	Describes the C-language-based Thread-to-Tid Mapping Service (ThreadTid) application programming interface available with Encina. This interface is part of the Toolkit Executive.
GC09-4521	*TM-XA Programming Pages*	Describes the C-language-based Transaction Manager-XA Service (TM-XA) application programming interface available with Encina. This interface is part of the Toolkit Server Core.
GC09-4522	*Trace Facility Programming Pages*	Describes the C-language-based Trace Facility application programming interface available with Encina.
GC09-4523	*Transactional-C Programming Pages*	Describes the C-language-based Transactional-C (Tran-C) application programming interface available with Encina. This interface provides transactional extensions and constructs for the C programming language.

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
GC09-4524	*TranLog Programming Pages*	Describes the C-language-based Transactional State Log (TranLog) application programming interface available with Encina. This interface is part of the Toolkit Server Core.
GC09-4525	*TRDCE Programming Pages*	Describes the C-language-based Transarc Encina Distributed Computing Environment (TRDCE) utilities available with Encina. This application programming interface is part of the Toolkit Executive.
GC09-4526	*TRPC Programming Pages*	Describes the C-language-based Transactional Remote Procedure Call (TRPC) application programming interface available with Encina.
GC09-4527	*TX Interface Programming Pages*	Describes the C-language-based Encina X/Open TX application programming interface available with Encina.
GC09-4528	*Volume Service (VOL) Programming Pages*	Describes the C-language-based Volume Service (VOL) application programming interface provided with Encina. This interface is part of the Toolkit Server Core.
GC09-4529	*Introductory Object-Oriented Programming Page*	Introduces the different object-oriented (C++ and Java) application programming interfaces available with Encina. Also describes some high-level issues associated with programming to Encina object-oriented interfaces.
GC09-4530	*DE-Light Java Programming Pages*	Describes the Java-based application programming interface available with the DCE Encina Lightweight Client (DE-Light).
GC09-4531	*Encina++ Programming Pages*	Describes the C++-language-based Encina C++ (Encina++) application programming interface.
GC09-4532	*Miscellaneous Object-Oriented Programming Pages*	Describes the data definition language (**ddl**) compiler and miscellaneous C++ and Java classes included with Encina.
GC09-4533	*OCCS Programming Pages*	Describes the C++-language-based Object Concurrency Control Service (OCCS) application programming interface available with Encina.
GC09-4534	*OTS Administrative Programming Pages*	Describes the C++-language-based administrative classes and functions in the Object Transaction Service (OTS) application programming interface available with Encina.
GC09-4535	*OTS C++ Programming Pages*	Describes the C++-language-based classes and functions in the Object Transaction Service (OTS) application programming interface available with Encina.
GC09-4536	*OTS Java Programming Pages*	Describes the Java-based classes and functions in the Object Transaction Service (OTS) application programming interface available with Encina.

Table 4. The library for WebSphere Application Server (continued)

Form number	Document name	Document description
GC09-4537	*OTS Synchronization Class Programming Pages*	Describes the C++-language-based synchronization class and function in the Object Transaction Service (OTS) application programming interface available with Encina.
GC09-4538	*RQS++ Programming Pages*	Describes the C++-language-based Recoverable Queueing Service (RQS) application programming interface available with Encina.
GC09-4539	*SFS++ Programming Pages*	Describes the C++-language-based Structured File Server (SFS) application programming interface available with Encina.
GC09-4540	*Transactional-C++ Programming Pages*	Describes the C++-language-based classes, functions, and constructs in the Transactional-C++ (Tran-C++) application programming interface available with Encina. This interface offers transactional extensions to the C++ programming language.
GC09-4541	*COBOL Programming Pages*	Describes the calls (functions) included with the Encina COBOL programming interface.

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:

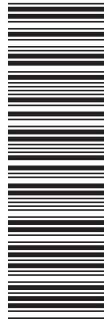
Copyright 1995, 1996 AT&T/NCR
Copyright 1995, 1996 BNR Europe Ltd.
Copyright 1991, 1992, 1995, 1996 by Digital Equipment Corporation
Copyright 1996 Gradient Technologies, Inc.
Copyright 1995, 1996 Groupe Bull
Copyright 1995, 1996 Expersoft Corporation
Copyright 1996 FUJITSU LIMITED
Copyright 1996 Genesis 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.

Index

Part Number: CT6PYNA

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.



(1P) P/N: CT6PYNA

SC09-4581-00

Spine information:

WebSphere

Getting Started with WebSphere Application Server

Version 4.0

SC09-4581-00