WebSphere Application Server Enterprise Edition

Introduction to WebSphere Application Server

Version 3.0

SC09-4430-00

WebSphere Application Server Enterprise Edition

Introduction to WebSphere Application Server

Version 3.0

SC09-4430-00

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 77.

First Edition (June 1999)

This edition applies to:

VisualAge Component Development for WebSphere Application Server Version 3.0, Enterprise Edition for AIX, program number 5765–E27
VisualAge Component Development for WebSphere Application Server Version 3.0, Enterprise Edition for Windows NT, program number 5639–I07
WebSphere Application Server Version 3.0, Enterprise Edition for AIX, program number 5765–E28
WebSphere Application Server Version 3.0, Enterprise Edition for Solaris, program number 5765–E29
WebSphere Application Server Version 3.0, Enterprise Edition for Windows NT, program number 5639–I09
WebSphere Application Server Version 3.0, Enterprise Edition Development Runtime for Windows NT, program number 5639–I11
WebSphere Application Server Version 3.0, Enterprise Edition Development Runtime for AIX, program number 5765–E31
WebSphere Application Server Version 3.0, Enterprise Edition Development Runtime for Solaris, program number 5765–E30

and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Consult the latest edition of the applicable system bibliography for current information on these products.

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

Contents

Figures . . . . **v**

Tables . . . . **vii**

About this book . . . . **ix**
Who should read this book ix
Document organization ix
Related information x
Conventions used in this book x
How to send your comments. xii

Chapter 1. Introducing the IBM WebSphere Family **1**
IBM and e-business 1
The WebSphere Family: providing e-business solutions 2
WebSphere Application Server: three editions for different customer needs 2
Other members of the WebSphere Family: enhancing the capabilities of the WebSphere Application Server 3
VisualAge for Java 6
Distributed computing and WebSphere Application Server 7
Three-tiered client/server computing 7
Transactions: ensuring data consistency and permanence in a distributed environment 8
Security: ensuring authorized use only 10
How do I get more information about WebSphere Application Server? 11
Installers and system administrators 11
Application developers and system architects 11

Chapter 2. WebSphere Application Server, Standard and Advanced Editions. **13**
What is the difference between the Standard and Advanced Application Servers? . . . 13
Introduction to Advanced Application Server 14
The Advanced Application Server environment 14
Java applets and servlets 15
JavaServer Pages 16
Web servers 17
EJB servers and enterprise beans . . . 17
The administration model in Advanced Application Server 18
Administration tools 19
The extensible markup language (XML) 20
Services used by the Advanced Application Server 21
Naming service 21
Transaction service 21
Security service 21
Workload management service 22

Chapter 3. WebSphere Application Server Enterprise Edition **23**
Why use Enterprise Application Server? 23
Low-level services used in Enterprise Application Server products 24
Distributed Computing Environment (DCE) 24
Common Object Request Broker Architecture (CORBA) 26
Component Object Model (COM) . . . 27
Other tools available with the Enterprise Application Server 27

Chapter 4. Introduction to Component Broker **29**
The Component Broker application architecture 31
Middle-tier architecture 33
Component architecture 34
Object services 37
Concurrency Control Service 37
Event Service 37
Notification Service 37
Externalization Service 38
Identity Service 38
Life Cycle Service 38
Naming Service 38
Security Service 39
Transaction Service 40
Session Service 40
Query Service 40
Cache Service 41

Workload Management. 41
System management 41
Development tools 43
Object Builder. 43

Chapter 5. Introduction to TXSeries . . . 47
More about TXSeries CICS 47
Basic CICS concepts. 48
The CICS application programming interface. 49
CICS-supplied transactions 50
Relational database support 50
Queue services 50
User exits 51
Intersystem communication 51
CICS SNA support 52
Communicating with users 52
CICS administration. 54
TXSeries Encina 55
Encina Monitor 55
The Recoverable Queueing Service (RQS) 56
The Structured File Server (SFS). . . . 57
The Peer-to-Peer Communications (PPC) Services 57
The DE-Light Gateway 58
The Encina Toolkit 58
Encina++ 59
Encina tools available only on Windows platforms 60
Example configurations 61
A simple CICS configuration 61
A simple CICS configuration within a DCE cell 62
A simple Encina Monitor cell configuration 63

Appendix. The library for WebSphere Application Server Enterprise Edition 65

Notices 77
Trademarks and service marks 79

Index 83

Figures

1. Three-tiered client / server architecture 8
2. The components of the Advanced Application Server environment . . . 15
3. The Component Broker System Manager 42
4. Communication between CICS clients and a CICS region. 53
5. A simple distributed configuration using CICS in a non-DCE cell environment 61
6. A distributed configuration using CICS in a DCE cell environment 62
7. A simple Encina Monitor configuration 63

Tables

1. Conventions used in this book . . . x
2. The library for WebSphere Application Server Enterprise Edition 65

About this book

This document provides a high-level look at the components that make up the WebSphere Application Server product. Although this book is specifically intended for use with 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 the components of the 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. WebSphere Application Server, Standard and Advanced Editions" on page 13 describes the main components and concepts of the Advanced Application Server and the Standard Application Server, with emphasis on the former.
- "Chapter 3. WebSphere Application Server Enterprise Edition" on page 23 introduces the Enterprise Application Server and associated concepts.
- "Chapter 4. Introduction to Component Broker" on page 29 provides more detailed information on Component Broker.
- "Chapter 5. Introduction to TXSeries" on page 47 provides more detailed information on TXSeries™.
- "Appendix. The library for WebSphere Application Server Enterprise Edition" on page 65 provides a complete list of the documentation available with the Enterprise Application Server.

Related information

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

- *Building Business Solutions with the WebSphere Family*
- *CICS and IMS Application Adaptor Quick Beginnings*
- *CICS Application Programming Guide*
- Component Broker *Application Development Tools Guide*
- Component Broker *Planning, Performance, and Installation Guide*
- Component Broker *Programming Guide*
- *Getting Started with Advanced Edition*
- *Getting Started with Component Broker*
- *Getting Started with TXSeries*
- *MQSeries Application Adaptor Quick Beginnings*
- *Oracle Application Adaptor Quick Beginnings*
- TXSeries *Planning and Installation Guide*
- *Writing Encina Applications*
- *Writing Encina Applications on Windows*
- *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 waseedoc@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 World Wide Web (the Web) is still relatively new, but its popularity among both individuals and businesses has grown rapidly. Although individuals use the Web for an array of different purposes, businesses use the Web primarily to provide products, services, and information to their customers, suppliers, 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 (like software companies) were among the first to enter this new frontier, and they often made their products, in the form of information or software, directly available for downloading.

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

While the Web side of many businesses was changing rapidly, non-Web business systems also went through some 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 systems remained largely detached from each other. The IBM e-business initiative and the WebSphere Family change that by enabling businesses to integrate their Web-based systems with their non-Web systems, to produce a single enterprisewide business system.

Further, the WebSphere Family is available in three different editions so that customers can approach the challenge of implementing e-business solutions in several different ways.

The WebSphere Family: providing e-business solutions

The IBM WebSphere Family was designed to help users realize the promise of e-business. The WebSphere Family 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 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, enterprisewide business systems and applications
- Businesses that want to integrate their Web presence with their non-Web 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: 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 enables powerful interactions 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 introduces server capabilities for applications built to the Enterprise JavaBeans™ Specification from Sun Microsystems and provides some support for integrating the Web applications to other non-Web business systems.
- The WebSphere Application Server Enterprise Edition (also called the Enterprise Application Server) builds on the Advanced Application Server and also offers a robust solution to grow e-business applications into enterprise environments. It combines TXSeries™, IBM's world-class transactional application environment, 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.

These three editions are available on two UNIX® platforms (IBM AIX® and Sun® Microsystems Solaris™) and Microsoft® Windows NT®. WebSphere Application Server is also available for the OS/390® and AS/400® platforms; only one edition of WebSphere Application Server is available on these platforms.

Other members of the WebSphere Family: enhancing the capabilities of the WebSphere Application Server

WebSphere Performance Pack

IBM WebSphere Performance Pack (Performance Pack) provides software building blocks to reduce Web server congestion, increase content availability, and improve Web server performance. Performance Pack is used across the WebSphere Application Server editions. It provides sophisticated detection of system utilization and error events across multiple networks and servers. It is extremely robust and scalable, providing caching of content across multiple servers and automating the replication and mirroring of data and applications.

Performance Pack is geared for Internet service providers (ISP), whether they are in the ISP business or in the business of providing access to internal users of corporate information technology (IT). Performance Pack is designed to help you provide Internet access and content hosting and includes the following components:

- IBM's AFS®, an enterprise file system that provides file sharing, replication, load balancing, and security. AFS can be placed anywhere in the network to provide automatic, nondisruptive real-time replication of information across multiple servers. AFS guarantees data consistency and availability, which in turn provide the global stability and administrative efficiency required by large, distributed Web sites.
- IBM SecureWay® Network Dispatcher (Network Dispatcher), load-balancing software with high availability features. Network Dispatcher can dynamically monitor and balance requests to available TCP/IP servers and applications in real time. Load balancing allows heavily accessed Web sites to increase capacity by linking many individual servers to a single logical server.
- IBM Web Traffic Express (Web Traffic Express), a caching proxy server with Platform for Internet Content Selection (PICS) filtering. Web Traffic Express provides highly scalable caching and filtering functions associated with receiving requests and serving URLs. With tunable caching capable of supporting high cache hit rates, the server can reduce bandwidth costs and provide more consistent rapid customer response times.

WebSphere Studio

The 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.

The 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, 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 your 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 on any of your WebSphere Application Servers—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 and 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 sub-sites 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 NetObjects TopPage 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.

OS/390 Component Broker

The OS/390 Component Broker product directly extends the reach of the Enterprise Application Server to the mainframe. This enables Component Broker developers to create distributed applications that can run on the Windows NT, AIX, Solaris, and OS/390 operating systems.

The Component Broker programming model effectively removes the burden of programming at the mainframe level, allowing the developer to concentrate on solving the business problem. Traditional programming tasks such as opening a socket, obtaining a lock, or managing virtual memory pools are not necessary when using OS/390 Component Broker. Nevertheless, OS/390 Component Broker is tightly integrated with the OS/390 operating system.

For more information on Component Broker, see “Chapter 4. Introduction to Component Broker” on page 29.

VisualAge for Java

VisualAge for Java is an integrated development environment that supports the complete cycle of Java program development. Although it is not a component of WebSphere Application Server or the WebSphere Family, 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, Java beans, and enterprise beans built to the Enterprise Java Beans (EJB) Specification. It also enables you to import existing code and export code as required from the underlying file system.

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 sophisticated code management system that makes it easy for you to maintain multiple editions of programs. When you want to capture the state of your code at any point, you can associate a version with different editions. This marks the particular edition as read-only and allows you to give it a name.

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 involved in distributed computing.

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 8. 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 may 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.

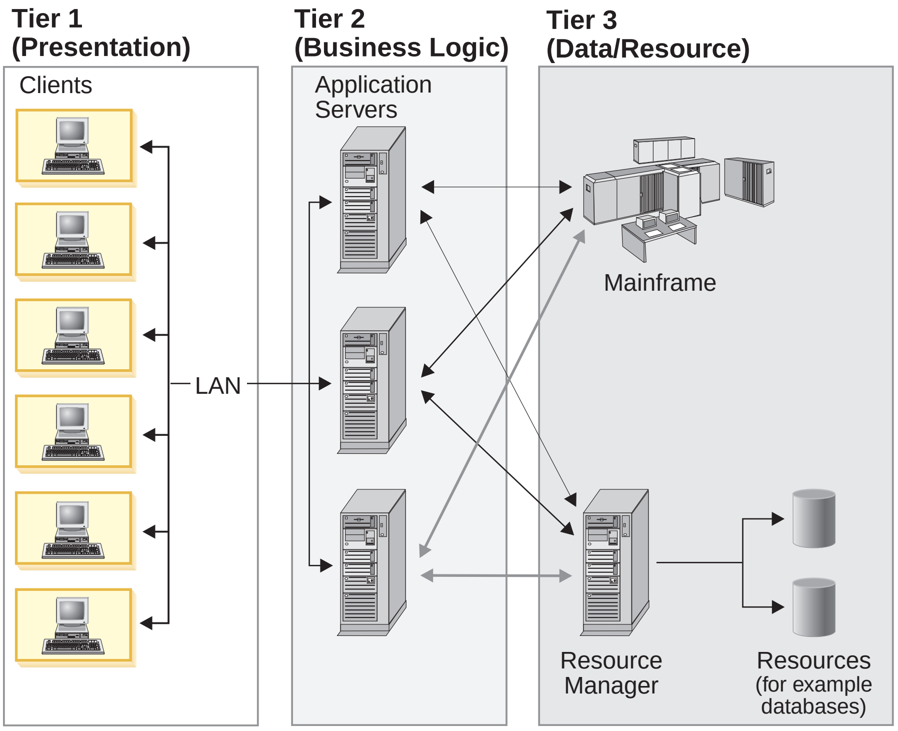

Figure 1. Three-tiered client/server architecture

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 central computer 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 really changed very much. 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; whereas 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 if 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.

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. These documents are available in HTML and PDF formats.

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 Advanced Application Server, read *Getting Started with Advanced Edition*. This document is designed for first-time users of the Advanced Application Server who want to get a simple system up and running quickly.
- To learn about managing the Advanced Application Server, access the Documentation Center and the online help available with the WebSphere Administrative Console.
- To learn the basics of installing Component Broker, read *Getting Started with Component Broker*; to learn the basics of installing and configuring TXSeries, read *Getting Started with TXSeries*. These documents are designed for first-time users of Enterprise Application Server who want to get either Component Broker or TXSeries up and running quickly.
- To learn about installing, configuring, and managing a more complicated 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.
- To learn how to use the adaptors available with Component Broker, start by reading one or more of the following:
  - *CICS and IMS Application Adaptor Quick Beginnings*
  - *Oracle Application Adaptor Quick Beginnings*
  - *MQSeries Application Adaptor Quick Beginnings*

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 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 the WebSphere Family*.
- To learn about developing applications in Component Broker, start by reading the *Application Development Tools Guide* and then read the Component Broker *Programming Guide*.
- 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* (for development on Windows NT, Windows® 95, or Windows 98 systems).

For a complete list of the documentation available with the Enterprise Application Server, see "Appendix. The library for WebSphere Application Server Enterprise Edition" on page 65.

Chapter 2. WebSphere Application Server, Standard and Advanced Editions

The Standard and Advanced Application Servers provide many of the powerful enterprise tools also found in the Enterprise Application Server, allowing you to build powerful e-business solutions.

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

There are several major difference between the Standard Application Server and the Advanced Application Server:

- The Advanced Application Server licenses and supports the development and use of enterprise beans written to the Enterprise JavaBeans™ (EJB) Specification from Sun Microsystems. The Standard Application Server neither supports nor licenses the development of enterprise beans.
- The Advanced Application Server supports the replication of EJB server models that makes it easy to clone EJB servers across multiple nodes, improving availability. The Standard Application Server does not allow replication.
- The Advanced Application Server supports a multiple machine environment for servers and servlets. The Standard Application Server supports only a single-machine environment for servers and servlets. Of course, both editions support access from multiple client machines.
- 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.

Despite these differences, there is complete compatibility between the two editions, which makes upgrading from Standard Application Server to Advanced Application Server a simple task.

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

Introduction to 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.
- 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, and presenting extensible markup language (XML) documents. This functionality is also available in the Standard Application Server.

The Advanced Application Server environment

The Advanced Application Server contains the following components, which can be combined to create a powerful Java-centered three-tiered system that puts heavy emphasis on a customer's Web site. These components are illustrated in Figure 2 on page 15.

- *Browser-based applications*—Allow users to send and receive information from Web sites by using the Hypertext Transfer Protocol (HTTP). There are three general types of browser-based applications: Java applets, Java servlets, and JavaServer Pages™ (JSP). For more information, see "Java applets and servlets" on page 15 and "JavaServer Pages" on page 16.
- *Web servers*—Except for stand-alone Java applets, which are restricted by built-in Java security, browser-based applications require that a Web server be installed on at least one machine in your Advanced Application Server environment. For more information, see "Web servers" on page 17.
- *EJB servers* and *enterprise beans*—The WebSphere EJB server contains one or more enterprise beans, which encapsulate the business logic and data used and shared by EJB applications. The enterprise beans installed in an EJB server do not communicate directly with the server; instead, an *EJB container* provides an interface between the enterprise beans and the EJB server, providing many low-level services such as threading, support for transactions, and management of data storage and retrieval. For more information, see "EJB servers and enterprise beans" on page 17 .
- *Java applications*—Java applications can interact directly with an EJB server by using Java remote method invocation over the Internet Inter-ORB Protocol (RMI/IIOP).

- *Data sources*—There are two types of enterprise beans: session beans, which encapsulate short-lived, client-specific tasks and objects, and entity beans, which encapsulate permanent or *persistent* data. The EJB server stores and retrieves this persistent data in a database.
- *Administration server* and the *administrative interface*—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 administrative interface to the administration server. For more information, see "The administration model in Advanced Application Server" on page 18.

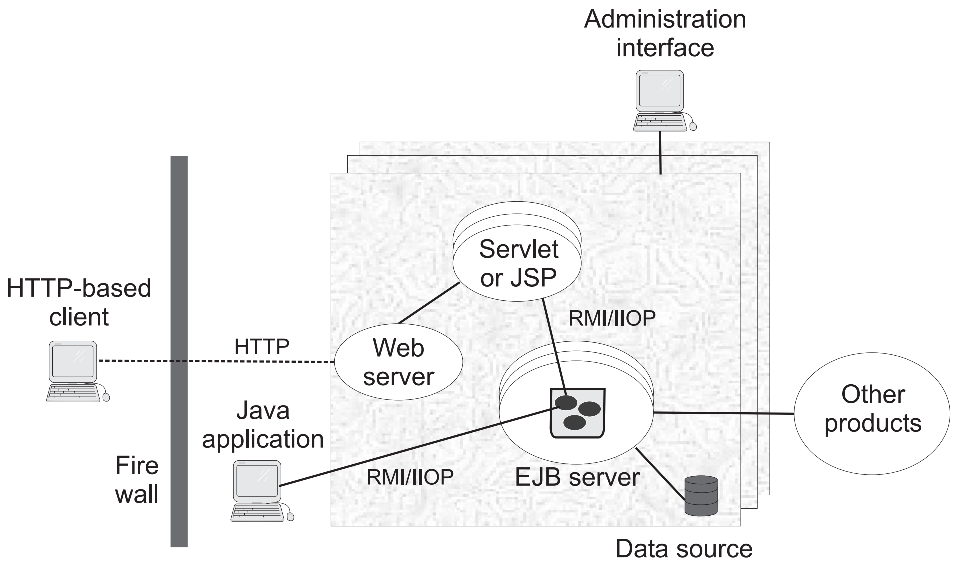

Figure 2. The components of the Advanced Application Server environment

Java applets and servlets

Java *applets* are Java applications that run on a browser and extend the browser's capabilities. Java applets can be designed by using the standard packages found in the Java JDK™ or by using the components of the Java Foundation Classes (JFC). For a Java applet to run inside a browser, the browser must support the classes used within the Java applet; however, most browsers can be updated to support the latest JDK by installing browser plug-ins.

Java *servlets* run on a Java-enabled Web server and extend the server's capabilities. Servlets are Java programs that use the Java Servlet API and the associated classes and methods. In addition to the Java Servlet API, servlets can use Java class packages that extend and add to the API. Java applets can be designed to interact with Java servlets, though this is not required.

Servlets extend Web server capabilities by creating a framework for providing request and response services over the Web. When a client sends a request to the server, the server can send the request information to a servlet and have the servlet construct the response that the server sends back to the client.

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.

Servlets perform a wide range of functions; for example, a servlet can:

- Create and return an entire HTML Web page containing dynamic content based on the nature of the client request.
- Create a portion of an HTML Web page (an HTML fragment) that can be embedded in an existing HTML page.
- Communicate with other server resources, including databases and Java-based applications.
- Handle connections with multiple clients, accepting input from and broadcasting results to the multiple clients. For example, a servlet can be a multiplayer game server.
- Open a new connection from the server to an applet on the browser and keep the connection open, allowing many data transfers on the single connection. The applet can also initiate a connection between the client browser and the server, allowing the client and server to easily and efficiently carry on a conversation. The communication can be through a custom protocol or through a standard such as IIOP.
- Filter data by MIME type for special processing, such as image conversion and server-side includes (SSI).
- Provide customized processing to any of the server’s standard routines. For example, a servlet can modify how a user is authenticated.

JavaServer Pages

The Advanced Application Server supports a powerful, new 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, Java servlet code (or other Java code) is embedded directly into the HTML page.

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

Web servers

The Web server provides the communications link between browser-based applications and the other components of Advanced Application Server. The Advanced Application Server contains a Java-based servlet engine that is independent of both your Web server and its underlying operating system.

Advanced 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 http://www.ibm.com/software/webservers/appserv.

EJB servers and 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. For more information on these services, see “Services used by the Advanced Application Server” on page 21.

There are two types of enterprise beans:

- An *entity* bean encapsulates permanent data, which is stored in a data source like a database or a file system, and associated methods to manipulate that data. 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.

  For example, the information about a bank account can be encapsulated in an entity bean instance. An account enterprise bean might contain an account ID, an account type (checking or savings), and a balance.
- A *session* bean encapsulates one or more business tasks and nonpermanent data associated with a particular client. Unlike the data in an entity bean, the data in a session bean is not stored in a permanent data source and no harm is caused if this data is lost. Nevertheless, a session bean can update

data in an underlying database, usually by accessing an entity bean. For this reason, a session bean can be transaction aware. When created, instances of a session bean are identical, though some session beans can store semipermanent data that makes them unique at certain points in their life cycle. A session bean is always associated with a single client.

For example, the task associated with transferring funds between two bank accounts can be encapsulated in a session bean. Such a transfer enterprise bean might find two instances of an account enterprise bean (by using the account IDs), and then subtract a specified amount from one account and add the same amount to the other account.

Before an enterprise bean can be installed in an EJB server, the enterprise beans must be deployed. During deployment, several EJB server-specific classes are generated. The *deployment descriptor* contains attribute and environment settings that define how the EJB server invokes enterprise bean functionality. Every enterprise bean (both session and entity) must have a deployment descriptor that contains settings used by the EJB server; these attributes can often be set for the entire enterprise bean or for the individual methods in the bean.

The Advanced Application Server provides tools for creating deployment descriptors and deploying enterprise beans.

The administration model in Advanced Application Server

WebSphere Application Server provides central administration of EJB servers and other resources. 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 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.

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 were replicated. This enables the administrator to duplicate server functionality across multiple nodes, improving availability and efficiency.

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 a framework for defining document markup languages. In simple terms, a document 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.

While HTML is the most widely used markup language for Web-based documents, as the popularity of HTML increased, the limitations of the language became more apparent. Those limitations include restricting the user to a relatively small set of tags. HTML authors cannot create their own HTML tags, because commercially available Web browsers have no knowledge of tags that are not part of the HTML standards that the browsers support.

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 and Hypertext Markup Language (HTML) are derived from the more complex Standard Generalized Markup Language (SGML). SGML's complexity and high cost of implementation spurred the interest in developing alternatives.

The *XML Document Structure Services* contained in the Advanced Application Server enables users to develop servlets and applications that implement server-side XML document 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.

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 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

The workload management service improves the scalability of the EJB server environment by grouping multiple EJB servers into EJB server groups. Clients then access these EJB server groups as if they were a single server, and the workload management service ensures that the workload is evenly distributed across the EJB servers in the EJB server groups. An EJB server can belong to only one EJB server group. The creation of EJB server groups is an administrative task that is handled from within the WebSphere Administrative Console.

Chapter 3. 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 Enterprise Application Server products run and additional products packaged with 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.

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 next two chapters examine the components of Enterprise Application Server and explains what each of these components do best. The remainder 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 Enterprise Application Server.

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 26.
- Microsoft Corporation's Component Object Model (COM). For more information, see "Component Object Model (COM)" on page 27.

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 4. Introduction to Component Broker" on page 29.

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 dynamically 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

This section discusses the standard parts of an ORB used in one or more Enterprise Application Server products.

CORBA IDL

One of the most important parts of the CORBA specification is the interface definition language (IDL). This object-oriented language enables programmers to create interfaces between components written in different programming languages, so that objects can be used by remote clients as if the objects were local to the client. This interaction between language-disparate components provides great flexibility to developers working in environments with many platforms and development tools.

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.

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.

Naming and binding

The CORBAservices specification details the manner in which a naming service must be implemented; however, the specification allows for differences between ORBs. Whether a particular ORB contains a naming service is left to the discretion of the ORB implementor. Both Orbix and the Component Broker ORB contain a naming service.

ORBs that do not implement a naming service must implement some way of enabling a client to bind to a server. For example, the Orbix _bind function can be used instead of the Orbix Naming Service.

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 more easily.

One of the most important aspects of COM is that a COM component can be used to create an application in any language. Once instantiated in a client, a COM object acts as a proxy for the client, marshaling and unmarshaling data to contact a server and returning data and status information back to the client.

Both TXSeries and Component Broker provide COM functionality.

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, which 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 CMP entity beans in both EJB implementations.
- IBM HTTP Server—The IBM HTTP Server is a powerful Web server that is based on the popular Apache Server. In addition to providing the full range of Web Server features, it provides enhanced SSL for secure transactions. In addition, the Advanced Application Server 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 (IDE) environment that contains many features for building an e-business solution. 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 6.
- 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, OS/2, and Windows NT.

Chapter 4. Introduction to Component Broker

Component Broker is an enterprise solution for distributed computing, providing a scalable, manageable environment for developing and deploying distributed component-based solutions. It is one of the most complete and integrated implementations 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.

Primarily, Component Broker is 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 a development environment called VisualAge Component Development for WebSphere Application Server V3.0 Enterprise Edition (CB Tools). The run-time package provides a server in which business object components run and are managed through a set of management tools.

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 multi-threaded server which provides easy access to a wide variety of services and capabilities.

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

Combinations of these object services are called *qualities-of-service* and are acquired according to the containers in which the objects run. Developers build the business logic, and the administrators make decisions about the qualities-of-service to be provided for any given installation of a business object.

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.

Many business object abstractions depend on existing resources. Component Broker supports the separation of the business logic from the state data in the base programming model. This separation is managed and controlled by the Component Broker run time. Component Broker provides access to 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 one of two ways. First, Component Broker provides a complete set of tools and support for building CORBA-based applications. Second, Component Broker tools enable the deployment of enterprise beans that were created by using other tools, such as VisualAge for Java. These two development methods are explained briefly in the following paragraphs.

A CORBA-based application is developed by using the Object Builder tools that come with Component Broker. Designs for systems can be imported from the Rational Rose® visual-modelling tool. After these designs are imported into Object Builder, the template for the implementation is available for use. Developers merely fill in the business logic in prescribed places within the 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 large-scale team development. Object Builder facilitates transformation of a large-system object architecture into separate pieces that can be worked on by many teams. Object Builder then facilitates linking these models back together and building an integrated solution.

Object Builder also serves as the deployment tool for enterprise beans that run in the Component Broker run time. Object Builder facilitates the mapping of entity beans with container-managed persistence (CMP) to databases and existing applications. Distributed object applications that leverage both enterprise beans and CORBA-based objects are easily constructed, tested, and deployed using the Object Builder tool and the Component Broker run time.

Component Broker is also an EJB server environment. It supports deploying and running components based on the Enterprise JavaBeans 1.0 Specification. It provides the qualities-of-service prescribed by the EJB specification and allows developers to build enterprise application business objects using the EJB programming model. Support for CMP allows mapping to a wide variety of resource managers. 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.

This overview provides an introduction to the following Component Broker features:
- “The Component Broker application architecture”
- “Middle-tier architecture” on page 33
- “Component architecture” on page 34
- “Object services” on page 37
- “System management” on page 41
- “Development tools” on page 43

The Component Broker application architecture

Component Broker applications are designed as three-tiered applications, as described in “Three-tiered client/server computing” on page 7. The content of each tier is summarized below:
- *First tier*—A programming model that allows client applications to be implemented in C++, Java, or Visual Basic, and allows clients to access components on the server through a CORBA-compliant ORB.
- *Middle tier*—A programming model with full tools support, and a run-time environment, in which the components (CORBA-based business objects or enterprise beans) are deployed.

- *Third tier*—Application adaptors on the middle tier allow components to access data in various resource managers, including DB2, Oracle, CICS, and IMS.

Three-tiered applications are reliable, extensible, and scalable. The development of such 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.

Component Broker provides components that 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).

In combination with existing resource managers, Component Broker functions as an object server by providing 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. Component Broker is platform-independent and allows you to design, develop, and deploy distributed object-oriented server applications for mission-critical solutions.

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 on a properly prepared object. This means that the object may have to be activated in memory, its state data may need to be refreshed, or it may need other services attached to the request before execution. Some examples of additional services provided to managed objects follow:
- Persistence, transactions, and security
- Workload management and availability management over multiple servers
- Object-oriented access to existing databases and applications

Component Broker enables operational reuse. Operational reuse focuses on reengineering existing software so it can be used as building blocks for new applications. The Component Broker infrastructure enables the construction of these new building blocks. These building blocks achieve operational reuse and present an abstraction layer that can in turn be reused to build many new business applications.

Component Broker clients can be C++ programs, Java applets, or Visual Basic programs that a user interacts with directly, or they can consist of Web servers or application servers that have their own clients. A typical client application consists of view objects, which provide the end-user interface interactions and a mapping to server components, and client objects, which implement business logic specific to the client and provide integration with other desktop applications, such as spreadsheets and document processors. Anything that can send an IIOP request is a potential client.

The server can have components on one machine or on many different machines. This layer includes programs written by developers as well as Component Broker components that make up the infrastructure.

The third tier can be running on many different physical hosts, and can be using application logic that itself provides additional physical tiers.

In the first tier, clients access Component Broker server objects through proxy objects.

The Component Broker client programming model also supports access to components in a fashion that matches the client language and programming model. For example, Component Broker provides support for component proxies with ActiveX/COM objects to facilitate their use in Visual Basic applications.

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

Middle-tier architecture

The middle tier of the three-tier architecture consists of components implemented as CORBA-based business objects and enterprise beans. The architecture for the middle tier of a Component Broker application has three layers, with three corresponding kinds of components: persistent components, composite components, and application components.

Persistent components are abstractions that map to existing data or procedural programs. While these components can have dependencies on other persistent components (one-to-one and one-to-many relationships), they are generally fine grained enough to be highly reusable.

Composite components represent new abstractions that are easily usable and understandable by client programmers and by application component programmers. These compositions often represent an aggregation of persistent components. Methods of the composite component typically delegate or bridge down to methods on the persistent components, which the component aggregates. These mappings can be one-to-one (mapping directly to a method of an aggregated component) or one-to-many (executing methods on each of the aggregated components). While composite components are not as reusable as persistent components, they can be more valuable when they are reused, because they represent larger portions of the application.

Application components focus on business logic and usage of other components. This is the layer that is accessed by client applications. Application components implement processes or tasks, as defined by object-oriented analysis and design. They implement any business logic that is not properly modeled as a method on other, individual components. Application components also provide the mechanism for moving application logic from the client to the server. Generally, application components represent the parts of the application architecture that are specific to an application, such as certain business processes or tasks. By separating out these elements, which are less suitable for reuse, you leave the rest of the application (composite components and persistent components) as reusable as possible.

When you extend an existing Component Broker application, or create a new application that uses the same data, you create new application components to provide application-specific business logic. However, you can reuse the persistent components and potentially the composite components. The persistent components need to change only if the underlying data store changes. The composite components need to change only if the definition of the aggregation changes. Even when change is necessary, the underlying component architecture allows the data store to change without affecting reuse of a component’s behavior or interface. This is described in the following section on component architecture.

Component architecture

Each component actually consists of a set of objects, which work together and 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. The client accesses the component to perform business functions. All Component Broker server objects are derived from the MOFW, which provides default code that allows the set of objects to work together and behave as a single entity. Components are factored into multiple objects to make them more maintainable. For example, a component’s mapping to a data store is kept in a separate object, so if the mapping changes, or a different data store is chosen, only the separate object needs to be updated.

Application components and composite components are made up of three basic objects, along with some helper objects for locating and creating the component. Persistent components add a fourth basic object, which provides the code to access resource managers or data stores on the third tier.

The main component objects are:
- Business objects
- Managed objects
- Data objects
- Key and copy helpers
- Persistent objects

Business objects

Business objects define the interface of the component, in terms of attributes and methods. 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 either C++ or Java. 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 using Component Broker tools, the framework is extended for you. Attributes considered as 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 having to:
- Know which of many data stores to use to make its state persistent
- Know how to access the data store
- Manage 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

Keys and copy helpers

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 is an optional object that provides an efficient way for the client application to create new instances of the component on the server. 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.

Persistent objects

A persistent object is a C++ object that provides 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 DB schema)
- Those used for accessing procedural data (mapping to a CICS and IMS bean, or Procedural Adaptor bean (PA bean)).

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. Once created, the components live in a container, which acts as an application adaptor for the component, providing management services to the component through its managed object.

When a call is made to a persistent component's get method (to retrieve the value of an attribute) the component objects work together as follows:

1. The managed object accepts the call, and calls its associated container for object services before passing the call on to the business object.
2. The business object accepts the call, and either returns the value of the attribute based on a cached copy of the data or delegates the call to the data object.
3. The data object accepts the call, and either returns the value of the attribute based on a cached copy of the data or delegates the call to the persistent object.

4. The persistent object retrieves the value from a database or other persistent data store, and returns it.
5. The value is returned up the component tree until it reaches the managed object, which calls the container again for object services before returning the value to the caller.

From the point of view of the caller, only the business object interface (which acts as the interface to the whole component) is visible. Even the managed object is hidden.

Object services

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

Concurrency Control Service

The Concurrency Control Service is intended primarily 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 creates a form of communication between objects that are unfamiliar with each other. This defines a channel between these objects which defines their roles. These channels allow multiple objects to communicate asynchronously. There are two defined roles: supplier objects and consumer objects. Suppliers produce events, while 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 or unregister their interest in certain events by creating an interface between objects that are unfamiliar with each other. These events occur within an object that is specified to be of interest to one or more objects. The Notification Service decouples communications between objects by defining two roles for objects: supplier objects and consumer objects. Suppliers produce events, while consumers process events.

Component Broker's Notification Service contains event channels that act as supplier and consumer objects. These event channels allow multiple suppliers to communicate with multiple consumers asynchronously and without confusing the many low-level details within the objects.

Externalization Service

The Externalization Service provides a mechanism by which objects are able to save and restore their state 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

Component Broker derives an object identity from relative information that positions the object within its container, server, host, and domain. This information can be used within the Component Broker Managed Object Framework to uniquely identify each object from any other object in the distributed system.

If your client program has two object references, you cannot simply compare the pointers to those references to determine whether they are referring to the same object. It is possible for two distinct references to refer to the same object, even though the reference objects themselves are two distinct objects. The references have different memory locations; therefore, you cannot reliably use the pointer values to determine whether the two objects are the same object. The Identity Service addresses this problem by providing unique identity information.

Life Cycle Service

A Life Cycle Service provides operations for creating, copying, moving, and deleting objects in a distributed environment. Because of the distributed nature of the environment, clients need to perform life cycle operations on objects in various locations. The Life Cycle Service in Component Broker provides a level of abstraction between the client program creating an object and the determination of the location where that new object will exist.

Naming Service

The Component Broker Naming Service allows you to create naming hierarchies so you can 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 and can also be used in a more limited role and have a less sophisticated implementation.

Security Service

Component Broker supports a variety of security services. Component Broker 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 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.

Component Broker defines the following qualities of protection (QOP), which can be configured according to business needs:

- Establish Trust in Client—Ensures that the requesting principal (either at a client process, or when a server invokes requests to other servers) is authenticated on any requests sent to a server.
- Establish Trust in Server—Ensures that the targeted server is authenticated on any requests sent to the server.
- Replay Detection—Verifies that the same message is not sent twice. For example, this is useful to prevent someone from forcing a bank withdrawal request to occur twice by copying the first one.
- Out-of-Sequence Detection—Verifies that two requests are not received out of order. For example, this is useful to prevent someone from forcing a bank withdrawal request from being processed before the deposit request; forcing the account to be overdrawn.
- Integrity—Ensures that the message content is not changed. For example, this is useful to prevent someone from changing the withdrawal amount.

Integrity adds some overhead to method requests, because each message is signed at the client and verified at the server.

- Confidentiality—Ensures that the message content can not be read. For example, this is useful to prevent someone from either seeing the withdrawal amount or even that this is a withdrawal request. Confidentiality adds a significant overhead to method requests, because each message is encrypted at the client and decrypted at the server.

Transaction Service

The Transaction Service enables programmers to implement transactions by using standard object-oriented interfaces in a distributed environment. Component Broker uses the Transaction Service to ensure that each application has correctly grouped the updates in the transaction so that the data is always updated consistently. If the 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 8.

Session Service

The Session Service provides detailed information for applications in a distributed object environment to control the extent of a session and the application profile and arbitrary 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 is started and the point when the session is ended. 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 enables you to find objects in a Component Broker collection based on a set of conditions described with an object-oriented structure query language (OOSQL). The OOSQL enables you to describe complex search criteria. It is a extension of SQL with features for handling object collections, object attributes, and methods in query statements. The Query Service can return a list of object references or it can return a list of object attribute values. The Query Service takes advantage of search capabilities and indexes in the underlying database to make searching for objects 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

The Workload Management capability allows the Component Broker run time to dynamically allocate an application server to process a request. As more clients use an application, the amount of work increases and the load on the servers increases. The purpose of workload distribution is to minimize response time for client requests and maximize server dispatch by reducing routing between objects in the distributed network. The key to workload distribution 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 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.

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. You can configure your business applications and the clients and servers they need 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 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

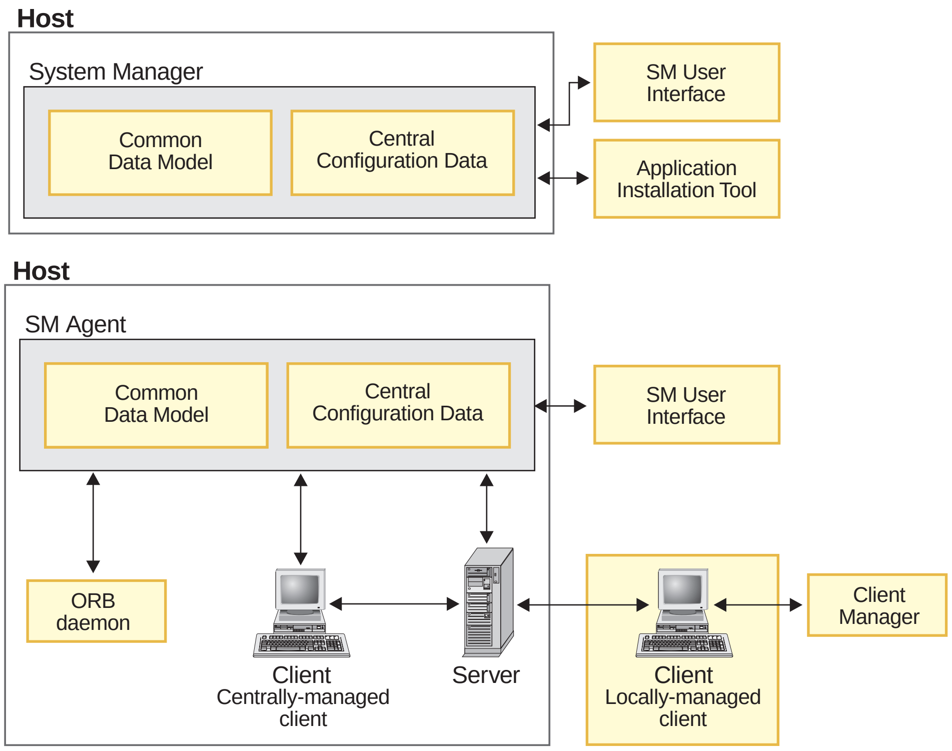

Figure 3. The Component Broker System Manager

The System Manager controls the applications, servers, and clients within the Component Broker network. It provides the logic required to manage configurations of Component Broker servers, clients, applications, and their resources. All servers, clients, applications, and their resources throughout the Component Broker network 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.

Development tools

The CB Tools can be used with VisualAge for Java and VisualAge C++ to generate multi-tier applications required by the Component Broker Server run time and systems management environments.

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 start by designing in Rational Rose and then import the design into Object Builder, and there 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. See the *WebSphere Application Server Enterprise Edition Component Broker Application Development Tools Guide* for more information.

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 which needs to be regenerated because of changes to the model since the last code generation are rebuilt.

You can set constraints to ensure that the components you develop are deployable on your target platforms. You can set object-specific platform constraints on:
- Data object implementations
- Managed objects
- Managed object configurations
- Containers
- Dynamic link library (DLL) files

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 a OS/390-specific version of the object.

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

The model for your application is constructed from components. Many of the development tasks in Object Builder revolve around defining 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. There are mechanisms in place for dealing with these differences. 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 backends 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 5. 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. This server combines the technologies of IBM's market-leading Customer Information Control System (CICS) and IBM's Encina transaction processing products. It enables customers to create a distributed, client/server environment with all the reliability, availability, and data integrity required for today's online transaction processing (OLTP).

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, and Sybase® relational databases.

More about 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, Solaris, and Windows NT, but other versions of CICS run on OS/390, AS/400, OS/2®, VMS, and other platforms.

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

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 TXSeries *Concepts and Facilities*.

Basic CICS concepts

An instance of a CICS system is called a CICS *region*. On UNIX or NT systems, this region consists of a number of processes. However, the region is configured and administered as a unit and controls a common set of resources. It is common to run multiple CICS regions on the same system. These regions can be independent of one another—for example, one for accounting, one for inventory management, and so on—or they can be closely tied together. CICS provides a number of facilities for interregion (also called intersystem) communication. (See “Intersystem communication” on page 51 for more information.)

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

Each user interaction with a CICS region involves one or more transactions. In CICS terms, a *transaction* is a basic operation that is offered to the user. For example, a banking application could include a query transaction, a debit transaction, a funds-transfer transaction, and so on. 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. The users typically select transactions by specifying the four-character name for the transaction. CICS then schedules the transaction to run. The transaction is then run in a process called an *application server*. If the transaction is implemented by several programs, those programs can run in the same or separate processes. The CICS region monitors the progress of the program, serving its requests for data, communications, and other resources. When the transaction completes, the CICS region commits any data changes, terminates the program, and frees resources for use by other transactions.

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

CICS uses a file manager—either the Encina Structured File Server (SFS) or DB2—to store transient data and Virtual Storage Access Method (VSAM) files. For more information, see “Transient data queue services” on page 51.

The CICS application programming interface

CICS provides a rich application programming interface (API) to allow 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:

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

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

On some platforms, CICS API commands are also implemented as C++ and Java methods.

CICS-supplied transactions

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

- The CEMT transaction allows you to examine and alter CICS system resource definitions for a running region.
- The CECI transaction allows you to check the syntax of and run EXEC CICS commands.
- The CEBR transaction allows you to browse temporary storage and transient data queues.
- The CMLV transaction browses the message log.
- The CSTD transaction displays statistics information.

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 using a full two-phase commit process if needed.

CICS provides support for the following relational database management systems:

- DB2 UDB
- Oracle
- 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. Both of these queue types are discussed and compared in the following sections.

Transient data queue services

CICS transient data queue services provide a generalized queueing facility. Data can be queued (stored) for subsequent internal or external processing. You can read, write, and delete data in a transient queue.

Temporary storage queue services

CICS temporary storage queue services provide the application programmer with the ability to store data in temporary storage queues. These queues can be located either in main storage, or in auxiliary storage on a direct-access storage device. Data stored in a temporary storage queue is known as temporary data. Temporary storage queues are efficient, but they must be created and processed entirely within CICS. You can read, write, update, and delete data in a temporary storage queue.

Temporary storage queues can be defined dynamically by an application. They do not have to be defined to CICS before the application uses them.

User exits

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

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
- 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* allows an application program to access files, transient data queues, and temporary storage queues belonging to another CICS system.
- *Transaction routing* allows you to execute a transaction on a remote system. The transaction is able to display information on your terminal as if it were running on your local system.
- *Asynchronous processing* extends the EXEC CICS START command to allow an application to initiate a transaction to run on another CICS system. As with standard EXEC CICS START calls, the transaction requested in the START command runs independently of the application issuing the START command.
- *Distributed transaction processing (DTP)* uses additional EXEC CICS commands that allow two applications running on different systems to pass information between themselves. These EXEC CICS commands map to the LU 6.2 mapped conversation verbs defined in the SNA Architecture.

  DTP is the only CICS intercommunication facility that can be used to communicate with non-CICS applications. The non-CICS applications must 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 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.

Users can communicate with CICS through the following:

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

If you have other devices for communicating with users, these are connected to IBM CICS clients. For example, an automatic teller machine (ATM) can be connected to a client to provide its user interface to CICS. Similarly, a printer attached to a client can be used for output from CICS.

Figure 4 illustrates the various communications methods used by CICS clients.

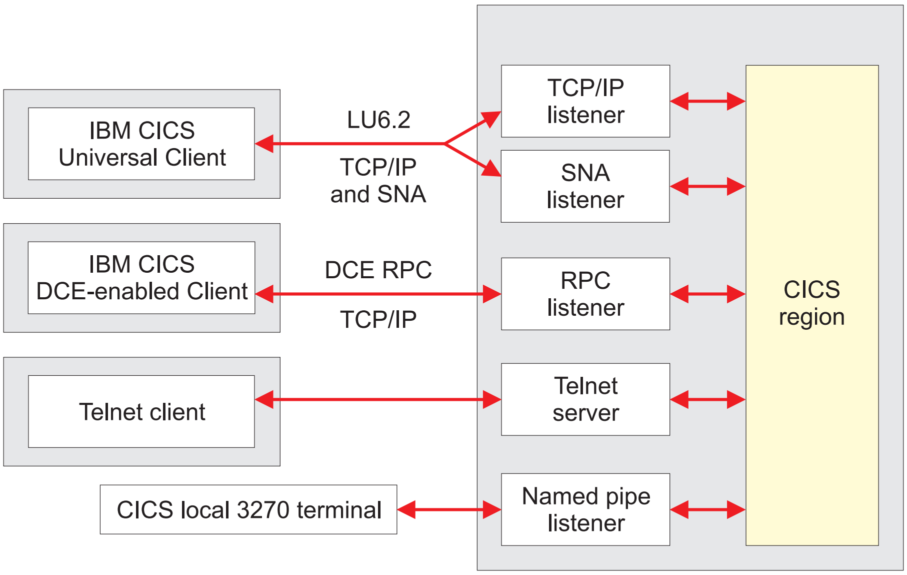

Figure 4. Communication between CICS clients and a CICS region

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 using the CICS Transaction Gateway, 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 client interfaces

CICS clients provide the following user interfaces:

- The External Call Interface (ECI), which enables a non-CICS application program running on the client machine to call a CICS program synchronously or asynchronously, as a subroutine. Client-based applications use simple ECI calls to pass blocks of data to CICS regions, without needing any special communications code.
- The External Presentation Interface (EPI), which enables an application program running in the client to invoke a CICS transaction on the server. The transaction is executed as though it is started from a 3270 terminal. The transaction returns a 3270 data stream to the client, which can present it in a GUI. This enables technologies such as graphical or multimedia interfaces to be used with traditional 3270 CICS applications without changing the CICS applications.
- 3270 terminal and printer emulation, which enables the client workstation to function as a 3270 terminal or printer. Existing CICS applications can be ported to the TXSeries environment (for example, from mainframe CICS regions) without change, as long as users have client machines that run the 3270 emulator (often called a CICS terminal or **cicsterm**). Those 3270 terminal sessions can be used to request the start of CICS transactions and to receive 3270 data-stream output from applications running on CICS regions.

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

CICS regions support requests from IBM CICS clients by means of TCP/IP and SNA *listener* processes running on the CICS regions. The listener process communicates with a corresponding process on the client machine and imitates the flows between regions. When a CICS client connects to a region, the server automatically installs a communications definition that identifies the CICS client and other communications attributes for it. For an EPI or **cicsterm** request, the server also installs a terminal definition that it uses to identify the 3270 characteristics of the CICS client.

CICS administration

Systems administration for CICS consists of configuring the CICS environment so that CICS regions can be started, monitoring running regions, shutting regions down, and recovering from problems. Administering CICS involves procedures that affect other components such as the 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 the administrative procedures. You can also use other tools, such as CICS commands and transactions.

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

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 these 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.

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 uses the following constructs to manage and manipulate data:

- A *queue* is a linear data structure that can be used to pass information from one application to another. Applications enqueue (add) elements to the tail of a queue and dequeue (remove) elements from the head of a queue in a first-in first-out (FIFO) manner (although RQS also provides other ways of accessing elements). Applications use queues to store data in the form of elements.
- An *element* is record-oriented data specific to an application. The fields of an element store related pieces of the information. For example, a billing element can have fields for storing the customer name, customer account number, and current account balance. Each element must have a type, which is specified when the element is added to a queue.
- An *element type* also defines an element’s keys. An element key is a sequence of one or more element fields to be used as a basis for retrieving the element. An element type is not required to have any associated

element keys. Element types are independent of queues; a single queue can contain elements of several different element types.

- A *queue set* serves as a dequeueing structure that regulates how elements are dequeued from its member queues. An application simply dequeues from the queue set, and the individual queue is chosen by the selection process defined for the set. Each queue in a queue set is assigned to a priority class, which ranks the queue (or a group of queues) in importance relative to other queues in the same queue set—priority 1 is higher than priority 3 in the same queue set.

RQS is described in the *Encina RQS Programming Guide*.

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 the 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 is described in the *Encina SFS Programming Guide*.

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. PPC Services supports 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.

The Encina PPC Services offer the following features:

- Integration and migration between mainframe and Encina environments. Encina transaction processing systems can interoperate with systems using

SNA. PPC Services enable bidirectional communications, so that both applications and data can be shared between mainframes and Encina, with either side initiating communications.

- LU 6.2 connectivity to the Encina Monitor. This enables an application running on an SNA network to allocate a conversation through the PPC gateway to an application in the Encina Monitor.
- Flexible administrative tools that can be used from any platform.
- Concurrency. The PPC Services provide thread-safe DPL, CPI-C, and CPI-RR routines for execution in the Open Software Foundation (OSF) DCE.

PPC is described in more detail in the *Encina PPC Services Programming Guide*.

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 the Distributed Computing Environment (DCE) and Encina to personal computers and other systems that are not running as DCE clients.

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

DE-Light is composed of the following components:

- Java API—Used to develop Java clients for standalone Java applications. DE-Light Java clients communicate with gateways via TCP/IP and Hypertext Transfer Protocol (HTTP).
- C API—Used to develop clients for the Microsoft Windows NT and Windows 95 environments. DE-Light C clients use TCP/IP to communicate with gateways at known endpoints.
- Gateway server—Enables communications between DE-Light clients and DCE or Encina.

DE-Light is described in more detail in the *Encina DE-Light Programming Guide*.

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.

Of the components that make up Encina++, some can be used only with Encina++/DCE or only with Encina++/CORBA, and some can be used by both.

The following Encina++ components can be used in any type of Encina++ application:

- The *Encina++* interface defines C++ classes and member functions that enable the creation and management of client/server applications and provide support for the underlying environment.
- The *Transactional-C++* (Tran-C++) interface defines C++ constructs and macros as well as classes and member functions for distributed transaction processing. This interface provides an object-oriented alternative to the Encina Transactional-C interface.
- The *Object Management Group Object Transaction Service* (OMG OTS) interface also defines C++ classes and member functions for distributed transactional processing. This interface implements the OMG *Object Transaction Service* specification as documented in *OMG document 94.8.4*.

For information on these components, see the *Encina Object-Oriented Programming Guide*.

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

- The RQS C++ interface (RQS++) defines C++ classes and functions for enqueueing and dequeueing data transactionally at an RQS server.
- The SFS C++ interface (SFS++) defines C++ classes and functions for manipulating data stored in record-oriented files at an SFS server while maintaining transactional integrity.

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

Encina++ provides two programming interfaces that are supported only for CORBA (including mixed applications). One defines a locking mechanism for CORBA-based servers, and the other enables you to develop Java transactional clients that can communicate with Encina++ servers:

- The Object Concurrency Control Service (OCCS) interface defines C++ classes and functions that enable multiple clients to coordinate access to shared resources. This interface implements the OMG *Concurrency Control Service Proposal*.
- The Java OTS Client interface defines Java classes and functions that enable Java client applications to begin and control distributed transactions. This interface implements the OMG *Object Transaction Service* specification.

For information on these components, see the *Encina Object-Oriented 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.

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. Depending on the type of server being created, this wizard invokes the DCE and CORBA **idl** and the Encina **tidl** compilers to create the required source and header files.
- 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. It invokes the Encina **comtidl** compiler, the Microsoft **midl** compiler and the DCE **idl** compiler to produce a COM project. After the COM project is used to compile the Encina COM dynamic link library (DLL) file, the 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.

Example 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 5. This configuration is recommended for first-time CICS installations because it is the easiest to install, test, and maintain.

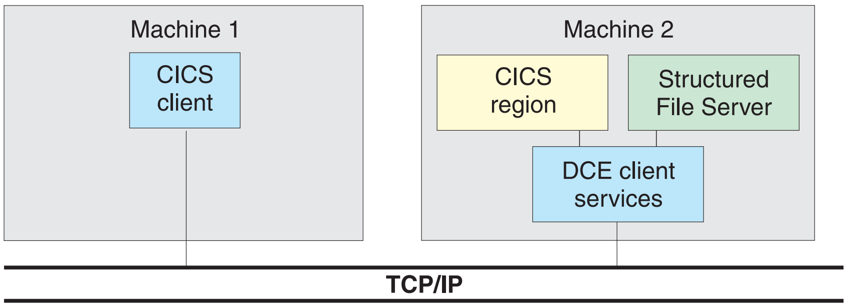

Figure 5. A simple distributed configuration using CICS in a non-DCE cell environment

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 and program access to the CICS region.

A simple CICS configuration within a DCE cell

A simple distributed CICS environment within a DCE cell consists of a client on one machine and a CICS region running on another machine, as shown in Figure 6. This configuration requires the DCE CDS and DCE Security Service to be installed on machines in the DCE cell.

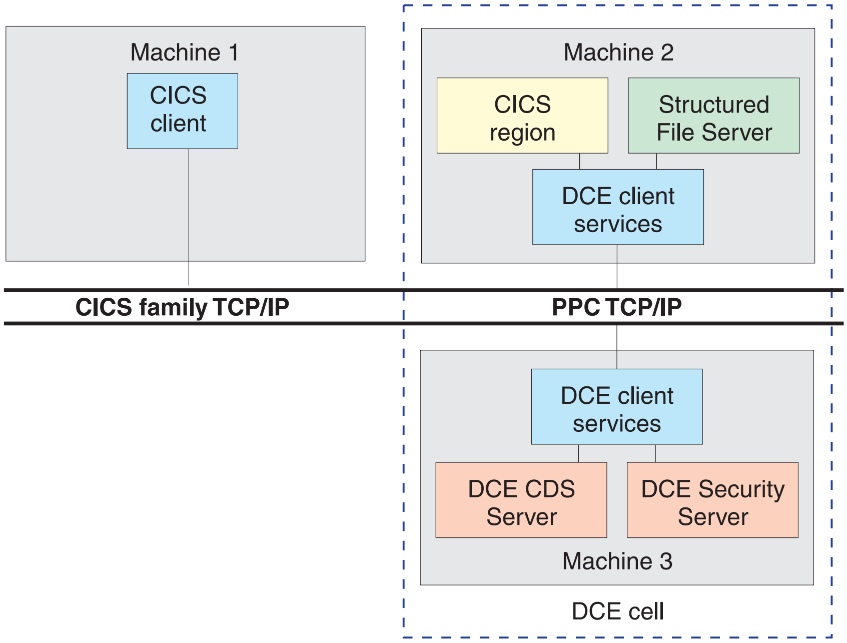

Figure 6. A distributed configuration using CICS in a DCE cell environment

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 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 7, 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.

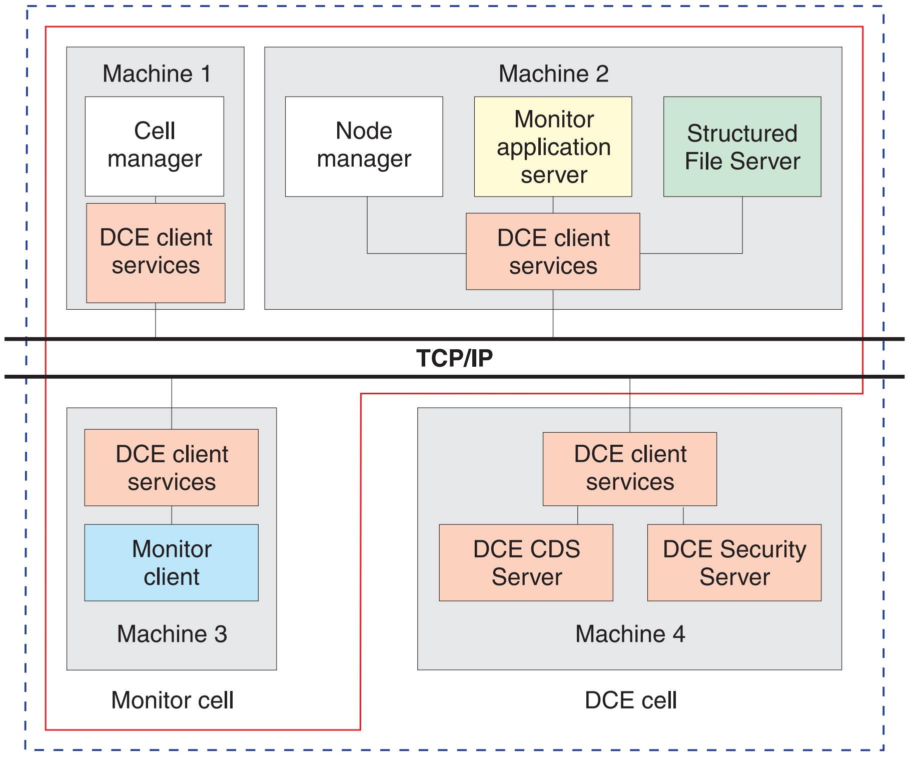

Figure 7. A simple Encina Monitor configuration

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.

Appendix. The library for WebSphere Application Server Enterprise Edition

The following table lists the complete documentation library for Enterprise Application Server:

Table 2. The library for WebSphere Application Server Enterprise Edition

Order number	**Book name**	**Book description**
Common documentation for WebSphere Application Server Enterprise Edition (for all supported platforms)		
SC09-4430	*Introduction to WebSphere Application Server*	Provides a common familywide overview of all editions of WebSphere Application Server and the contents of each edition.
SC09-4431	*Writing Enterprise Beans in WebSphere*	Provides an introduction to programming with Enterprise JavaBeans in the Advanced and Enterprise Editions of WebSphere Application Server.
SC09-4432	*Building Business Solutions with the WebSphere Family*	Provides programming examples and scenarios that demonstrate application development and recommended programming practices across the WebSphere Application Server family.
Common Component Broker documentation (for all supported platforms)		
SC09-4439	*CICS and IMS Application Adaptor Quick Beginnings*	Provides information on installing and verifying the application adaptor that provides communications between Component Broker applications and CICS or IMS.
SC09-4440	*Oracle Application Adaptor Quick Beginnings*	Provides information on installing and verifying the application adaptor that provides communications between Component Broker applications and an Oracle database.
SC09-4441	*MQSeries Application Adaptor Quick Beginnings*	Provides information on installing and verifying the application adaptor that provides communications between Component Broker applications and MQSeries.
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, data objects, and includes information about various basic programming issues.

Table 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	Book name	Book description
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 Concepts and Development Guide*	Introduces the MQSeries Procedural Application Adaptor and provides information about developing applications that use both Component Broker and MQSeries.
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.
SC09-4446	*Programming Reference Volume 1*	With Volume 2, documents the complete Component Broker application programming interfaces (APIs) available for all supported programming languages. In online formats (HTML and PDF), the *Programming Reference* is provided as a single volume.
SC09-4447	*Programming Reference Volume 2*	With Volume 1, documents the complete Component Broker application programming interfaces (APIs) available for all supported programming languages. In online formats (HTML and PDF), the *Programming Reference* is provided as a single volume.
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 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	Book name	Book description
GC09-4490	*Component Broker Release Notes*	Provides platform- and release-specific information about Component Broker, including descriptions of new features that are more thorough than those in the Component Broker README file, information for features or changes learned too late for incorporation into the product documentation, and information about known restrictions associated with Component Broker and, where possible, suitable workarounds.
Component Broker for AIX documentation		
SC09-4434	*Getting Started with Component Broker*	Provides simple, straightforward scenarios for setting up Component Broker on AIX.
SC09-4437	*Planning, Performance, and Installation Guide*	Provides complete instructions for installing, configuring, and upgrading to the latest version of Component Broker on AIX.
Component Broker for Solaris documentation		
SC09-4435	*Getting Started with Component Broker*	Provides simple, straightforward scenarios for setting up Component Broker on Solaris.
SC09-4438	*Planning, Performance, and Installation Guide*	Provides complete instructions for installing and configuring the latest version of Component Broker on Solaris.
Component Broker for Windows NT documentation		
SC09-4433	*Getting Started with Component Broker*	Provides simple, straightforward scenarios for setting up Component Broker on Windows NT.
SC09-4436	*Planning, Performance, and Installation Guide*	Provides complete instructions for installing, configuring, and upgrading to the latest version of Component Broker on Windows NT.
Component Broker for OS/390 documentation		
GA22-7324	*OS/390 Component Broker Introduction*	Describes the concepts and facilities of Component Broker and the value it has on the OS/390 platform.
GA22-7331	*OS/390 Component Broker Planning and Installation*	Describes the planning and installation considerations for Component Broker on the OS/390 platform.
GA22-7328	*OS/390 Component Broker System Administration*	Describes system administration tasks and operations tasks, as provided in the system administration user interface.
GA22-7326	*OS/390 Component Broker Programming: Assembling Applications*	Provides information for assembling applications using Component Broker on the OS/390 platform.

Table 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	Book name	Book description
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.
Common TXSeries (CICS and Encina) documentation (for all supported platforms)		
SC09-4455	*Concepts and Facilities*	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. It also includes information about IBM and third-party products supported for use with TXSeries.
TXSeries for AIX documentation		
SC09-4544	*Getting Started with TXSeries*	Provides simple, straightforward scenarios for setting up CICS and Encina on AIX.
SC09-4452	*Planning and Installation Guide*	Provides complete instructions for installing, configuring, and upgrading to the latest version of TXSeries (CICS, Encina, and the DE-Light Gateway server and clients) on AIX.
TXSeries for Solaris documentation		
SC09-4545	*Getting Started with TXSeries*	Provides simple, straightforward scenarios for setting up CICS and Encina on Solaris.
SC09-4453	*Planning and Installation Guide*	Provides complete instructions for installing, configuring, and upgrading to the latest version of TXSeries (CICS, Encina, and the DE-Light Gateway server and clients) on Solaris.
TXSeries for Windows NT documentation (for Windows NT and Windows 95/98)		
SC09-4543	*Getting Started with TXSeries*	Provides simple, straightforward scenarios for setting up CICS and Encina on Windows NT.
SC09-4451	*Planning and Installation Guide*	Provides complete instructions for installing, configuring, and upgrading to the latest version of TXSeries (CICS, Encina, and the DE-Light Gateway server and clients) on Windows NT.
CICS documentation (for all supported platforms unless otherwise noted)		

Table 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	Book name	Book description
SC09-4457	*CICS Administration Guide*	Provides guide information for administering CICS and CICS applications on AIX. It includes information on the new Tivoli interface for CICS. It also includes the single CICS glossary.
SC09-4458	*CICS Administration Guide*	Provides guide information for administering CICS and CICS applications on Solaris. It includes information on the new Tivoli interface for CICS. It also includes the single CICS glossary.
SC09-4456	*CICS Administration Guide*	Provides guide information for administering CICS and CICS applications on Windows NT. It includes information on the new Tivoli interface for CICS. It also includes the single 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 (APIs) 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 NT machine or another application on a system such as a mainframe).
SC09-4463	*CICS Messages and Codes Volume 1*	With Volume 2, lists and describes all messages and codes that can be issued by a CICS system. In online formats (HTML and PDF), *CICS Messages and Codes* is provided as a single volume.
SC09-4464	*CICS Messages and Codes Volume 2*	With Volume 1, lists and describes all messages and codes that can be issued by a CICS system. In online formats (HTML and PDF), *CICS Messages and Codes* is provided as a single volume.
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 new CICS Workload Management utility.

Table 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	**Book name**	**Book description**
SC09-4467	*CICS IIOP ORB Programming Guide*	Describes the new CICS Internet Inter-ORB Programming (IIOP) interface for AIX, which enables object-oriented CICS applications to communicate with Common Object Request Broker Architecture (CORBA) Object Request Brokers (ORBs).
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 version 5.0.
SC09-4470	*Using IBM Communications Server for Windows NT with CICS*	Provides information for using CICS with the Systems Network Architecture (SNA) package provided by IBM Communications Server for Windows NT, version 6.0.
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 NT.
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.
SC33–1946	*OO Programming in C++ for CICS Servers*	Describes how to program in the C++ language with the CICS Foundation Classes (CFCs). This documentation is delivered as part of a technology release made available only from a directory on the TXSeries software CD-ROMs.
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 platforms. It describes the use of the Enconsole administrative interface and includes the complete Encina glossary. It also includes information on the new 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)		

Table 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	Book name	Book description
SC09-4476	*Encina COBOL Programming Guide*	Describes the COBOL application programming interface (API) available with Encina.
SC09-4477	*Encina Monitor Programming Guide*	Describes how to program with the Encina Monitor application programming interface (API).
SC09-4478	*Encina Object-Oriented Programming Guide*	Describes how to program with the Encina object-oriented (Encina++) application programming interfaces (APIs) used to develop applications in either a Distributed Computing Environment (DCE) or Common Object Request Broker Architecture (CORBA) environment.
SC09-4479	*Encina RQS++ and SFS++ Programming Guide*	Describes how to program with the Encina object-oriented application programming interfaces (APIs) 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 (APIs). 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 (API).
SC09-4482	*Encina RQS Programming Guide*	Describes how to program to the Encina Recoverable Queueing Service (RQS) application programming interface (API).
SC09-4483	*Encina SFS Programming Guide*	Describes how to program to the Encina Structured File Server (SFS) application programming interface (API).
SC09-4484	*Encina Toolkit Programming Guide*	Introduces and describes how to program to the various Encina Client (Executive) and Server Core application programming interfaces (APIs).
SC09-4485	*Encina Transactional Programming Guide*	Describes how to program with the the Encina Transactional-C (Tran-C) application programming interface (API).
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 (APIs). It includes general information on Encina terminology and application development.

Table 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	Book name	Book description
SC09-4487	*Writing Encina Applications on Windows*	Provides information on using the TXSeries application development toolkit (ADK) to develop Encina applications on Windows NT. 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)		
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.

Table 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	Book name	Book description
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).
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 (APIs) 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 (API) 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 (API) available with the DCE Encina Lightweight Client (DE-Light).

Table 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	Book name	Book description
GC09-4508	*Distributed Transaction Service (TRAN) Programming Pages*	Describes the C-language-based Distributed Transaction Service (TRAN) application programming interface (API) 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 (API) 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 (API) 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 (API) available with Encina. This interface is part of the Toolkit Server Core.
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 (API) available with Encina.
GC09-4514	*PPC Executive Programming Pages*	Describes the C-language-based Peer-to-Peer Communications (PPC) Executive application programming interface (API) available with Encina.
GC09-4515	*Recovery Service (REC) Programming Pages*	Describes the C-language-based Recovery Service (REC) application programming interface (API) 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 (API) 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 (API) available with Encina.
GC09-4518	*SFS Programming Pages*	Describes the C-language-based Structured File Server (SFS) application programming interface (API) available with Encina.
GC09-4519	*T-ISAM Programming Pages*	Describes the C-language-based Transactional Indexed Sequential Access Method (T-ISAM) application programming interface (API) available for use with the Structured File Server (SFS) component of Encina.

Table 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	Book name	Book description
GC09-4520	*ThreadTid Programming Pages*	Describes the C-language-based Thread-to-Tid Mapping Service (ThreadTid) application programming interface (API) 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 (API) 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 (API) available with Encina.
GC09-4523	*Transactional-C Programming Pages*	Describes the C-language-based Transactional-C (Tran-C) application programming interface (API) available with Encina. This interface provides transactional extensions and constructs for the C programming language.
GC09-4524	*TranLog Programming Pages*	Describes the C-language-based Transactional State Log (TranLog) application programming interface (API) 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 (API) is part of the Toolkit Executive.
GC09-4526	*TRPC Programming Pages*	Describes the C-language-based Transactional Remote Procedure Call (TRPC) application programming interface (API) available with Encina.
GC09-4527	*TX Interface Programming Pages*	Describes the C-language-based Encina X/Open TX application programming interface (API) available with Encina.
GC09-4528	*Volume Service (VOL) Programming Pages*	Describes the C-language-based Volume Service (VOL) application programming interface (API) 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 (APIs) 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 (API) 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 (API).

Table 2. The library for WebSphere Application Server Enterprise Edition (continued)

Order number	Book name	Book description
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 (API) 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 (API) 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 (API) 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 (API) available with Encina.
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 (API) available with Encina.
GC09-4538	*RQS++ Programming Pages*	Describes the C++-language-based Recoverable Queueing Service (RQS) application programming interface (API) available with Encina.
GC09-4539	*SFS++ Programming Pages*	Describes the C++-language-based Structured File Server (SFS) application programming interface (API) 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 (API) 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

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:

AFS	IMS
AIX	MQSeries
AS/400	MVS/ESA
CICS	OS/2
CICS OS/2	OS/390
CICS/400	OS/400
CICS/6000	PowerPC
CICS/ESA	RISC System/6000
CICS/MVS	RS/6000
CICS/VSE	S/390
CICSPlex	Transarc
DB2	TXSeries
DCE Encina Lightweight Client	VSE/ESA
DFS	VTAM
Encina	VisualAge
IBM	WebSphere

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks of Microsoft Corporation in the United States and/or other countries.

Oracle and Oracle8 are registered trademarks of the Oracle Corporation in the United States and/or other countries.

Index

A

ACID properties 8
Advanced Application Server 2, 13, 14
AFS 3
Apache Web Server 17, 27
applets 15
authentication 10
authorization 10

C

C++ 27, 29, 43
Cell Directory Service 25
CGI 15
CICS
  administration 54
  API 49
  CICS-supplied transactions 50
  client interfaces 54
  intersystem communication 51
  regions 48
  relational database support 50
  SNA support 52
  supported platforms 47
  Transaction Gateway 53
  transactions 48
  user exits 51
COM 27, 33
Component Broker 29
  architecture 31
  business objects 35
  Cache Service 41
  Concurrency Control Service 37
  copy helpers 36
  data objects 35
  development tools 43
  enterprise beans 31
  Event Service 37
  Externalization Service 38
  Identity Service 38
  keys 36
  Life Cycle Service 38
  Managed Object Framework 34
  managed objects 35
  Naming Service 38
  Notification Service 37
  Object Builder 30, 43
  object services 37
  OS/390 5
Component Broker 29 *(continued)*
  persistent objects 36
  Query Service 40
  resources 30
  Security Service 39
  Session Service 40
  System Management 41
  Transaction Service 40
  Workload Management 41
CORBA 26, 27, 29
  CORBAservices 27
  IDL 26, 35
CPI-C 57
CPI-RR 57

D

DB2 27
DCE 24
DE-Light Gateway 58
distributed computing 7
Distributed Time Service (DTS) 26
distributed transactions 9
documentation 11, 65

E

e-business 1, 2
EJB servers 14, 17, 31
Encina
  COM Wizard 60
  DE-Light Gateway 58
  Monitor 55
  PPC 57
  RQS 56
  Server Wizard 60
  SFS 57
  SNA support 57
  supported platforms 55
  Toolkit 58
  tracing tools 61
Encina++ 59
Enterprise Application Server 2, 23
enterprise beans 17, 29, 31
entity beans 17

H

HTML 15

I

IBM HTTP Server 17, 27
IDL (CORBA) 26, 35

J

Java 15, 27, 29, 31, 60
JFC 15
JNDI 21
JSP files 14, 16

L

LUWs 48

M

Managed Object Framework 34
MQSeries 27

N

naming
  Advanced Application Server 21
  Component Broker 38

O

Object Builder 30, 43
ORBs 26, 27, 30
OS/390 Component Broker 5

P

platforms
  Component Broker 5, 44
  TXSeries CICS 47
  TXSeries Encina 55
  WebSphere Application Server 3

R

remote calls 27
RPCs 24
RQS 56
RQS++ 59

S

SecureWay Network Dispatcher 3
security 10
  Advanced Application Server 21
  Component Broker 39
  Enterprise Application Server 25
servlets 14, 15
session beans 17
SFS 57
SFS++ 59
SNA 52, 57
Standard Application Server 2, 13

T

three-tiered architecture 7
Tran-C 58

transactions 8
- Advanced Application Server 21
- CICS 48
- Component Broker 40
- distributed 9

TXSeries 47
- example configurations 61

V

VisualAge C++ 27, 43

VisualAge for Java 6, 27, 30, 43

W

Web servers 17

WebSphere Administrative Console 14, 18, 19

WebSphere Application Server 2
- distributed computing 7
- documentation 11
- supported platforms 3
- three-tiered architecture 7
- transactions 8

WebSphere Family 2, 3, 4, 5

WebSphere Performance Pack 3

WebSphere Studio 4

workload management
- Advanced Application Server 22
- Component Broker 41

X

XML 14, 20

Program Number: 5765-E27
5639-I07
5765-E28
5765-E29
5639-I09
5639-I11
5765-E31
5765-E30

SC09-4430-00

Spine information:

IBM

WebSphere

Introduction to WebSphere Application Server

Version 3.0

SC09-4430-00