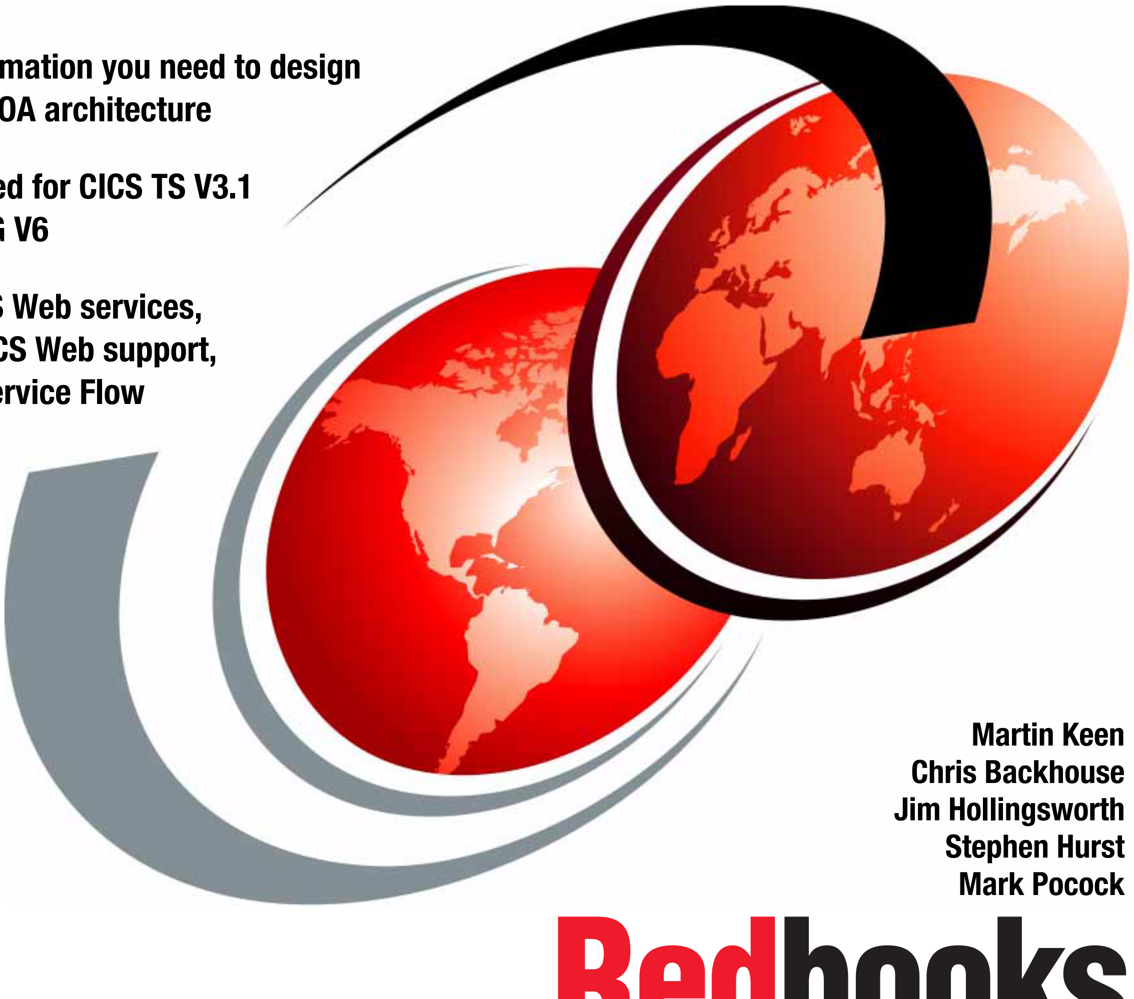
WebSphere. software

IBM

Architecting Access to CICS within an SOA



All the information you need to design your CICS SOA architecture

Fully updated for CICS TS V3.1 and CICS TG V6

Covers CICS Web services, CICS TG, CICS Web support, and CICS Service Flow Feature

Martin Keen
Chris Backhouse
Jim Hollingsworth
Stephen Hurst
Mark Pocock

Redbooks

ibm.com/redbooks

International Technical Support Organization

Architecting Access to CICS within an SOA

October 2006

SG24-5466-05

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

Sixth Edition (October 2006)

This edition applies to Version 3, Release 1, of CICS Transaction Server.

Contents

Notices

Trademarks

Preface

With the emergence of service-oriented architecture (SOA), the options for accessing existing CICS® assets have become more varied than ever. This IBM® Redbook is intended for IT architects who select, plan, and design SOA solutions that make use of CICS assets.

First, we provide an introduction to SOA and the options for transforming CICS assets into SOA solutions. Then we introduce the different architectures and technologies on which a CICS SOA solution can be based.

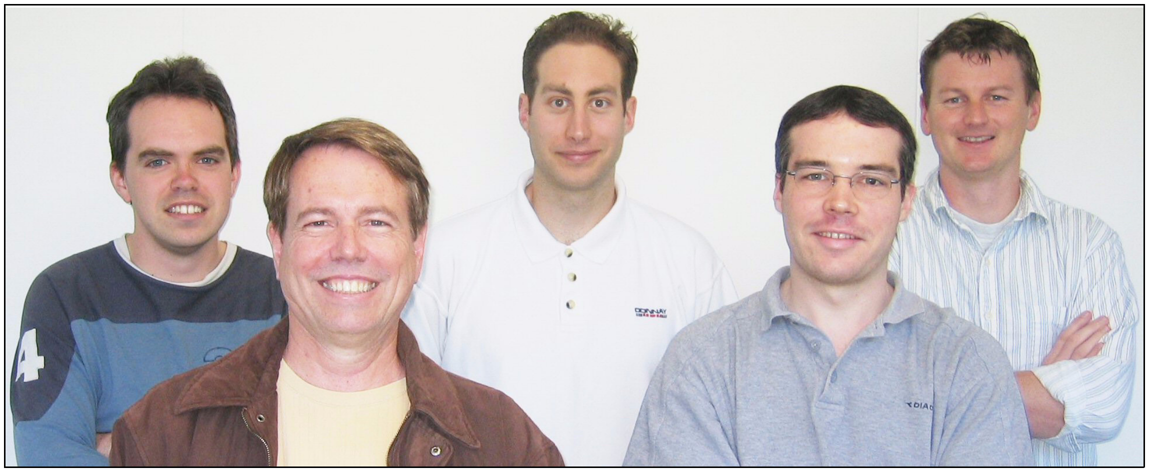
Next we offer a broad overview of the different technologies available for SOA enablement of CICS applications. We also offer a comparison of some of the current strategic technologies that include the CICS Web services support in CICS TS 3.1, CICS Transaction Gateway, CICS Web support, Link3270 bridge, and the CICS Service Flow Feature.

We continue with an in-depth analysis of the issues pertaining to security, transaction management, performance, application development, and data conversion when designing and developing your SOA solutions.

Finally, we document different customer scenarios, where these technologies have been successfully used to integrate existing enterprise information systems into new SOA solutions.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.


The IBM Redbook team (left to right): Chris, Jim, Mark, Steve, and Martin

Martin Keen is a Senior IT Specialist at the ITSO, Raleigh Center. He writes extensively about WebSphere® products, SOA, and Patterns for e-business. He also teaches IBM classes worldwide about WebSphere, SOA, and business process management. Before joining the ITSO, Martin worked on the EMEA WebSphere Lab Services team in Hursley, UK, supporting CICS Transaction Server. Martin holds a bachelor's degree in Computer Studies from Southampton Institute of Higher Education.

Chris Backhouse is a software engineer in the CICS Transaction Server development organization. He has a degree in Computer Engineering from the University of Southampton. Chris' areas of expertise include CICS TS, enterprise connectivity with a focus on Web services, J2EE™, and enterprise application transformation and integration. Before his development role, he worked for four years as a designer on enterprise integration scenario projects.

Jim Hollingsworth is a Senior Software Engineer and the Team Lead for WebSphere Developer for zSeries®-based CICS application deployment management. He is a certified Consulting IT Specialist and a certified CICS e-business Solution Designer. His development accomplishments include the first TCP/IP gateway to CICS, the CA8E SupportPac™ for dynamic attribute support with the 3270 Web Bridge, the CICS Web Interface Generic Converter, the Generic Bridge, and the Link3270 vector processing support for the CICS Service Flow Feature. He has developed patents and pending patents in XML parsing technology and high performance state data token design. Jim has presented CICS topics at both SHARE and CICS Transaction and Messaging conferences.

Stephen Hurst is an IT Consultant in IBM Software Services for the WebSphere pan-IOT organization. He has worked for IBM for 6 years, mostly within the Transaction Processing Development organization based in Hursley, UK. He holds a degree in Artificial Intelligence from the University of Durham. His areas of expertise include enterprise modernization and application transformation, predominantly based around access to CICS.

Mark Pocock is a Software Engineer in the CICS Transaction Server level 3 support team and has nine years of experience supporting CICS. He holds a degree in Mathematics and Computer Science from the University of Kent. His areas of expertise within CICS include Web Support, Web Services, 3270 Bridge, data conversion and Java™ and EJB™ support. This is the second version of this redbook that Mark has been involved in updating.

Thanks to the following people for their contributions to this project:

Phil Wakelin
IBM Hursley, CICS Strategy and Planning

Fraser Bohm
IBM Hursley, CICS Development Software Engineer

Paul Cooper
IBM Hursley, CICS Development Software Engineer

Trevor Clarke
IBM Hursley, CICS Development Software Engineer

John Burgess
IBM Hursley, CICS Development Software Engineer

Anne Murden
IBM Availability Manager

Peter Havercan
IBM Hursley, CICS Development

Earl Rogers

Thanks to the team that wrote the previous release of this redbook: Chris Rayns, Pingze Gao, Robert Herman, Jim Hollingsworth, Mark Pocock, Phil Wakelin, and Nigel Williams.

Thanks to the team that wrote the redbook *Implementing CICS Web Services,* SG24-7206: Nigel Williams, Grant Ward Able, Paolo Chieregatti, Robert Herman, Tommy Joergensen, Luis Aused Lopez and Steve Wall.

Thanks to the team that wrote the redbook *Developing for CICS Web Services,* SG24-7126: Chris Rayns, David Evans, Leigh Compton and Isabel Arnold.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-5466-05, *Architecting Access to CICS within an SOA*
as created or updated on October 31, 2006.

Second Edition, March 2001

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- *Matters of State* chapter

Changed information

- *Introduction to CICS and Web-enabling* chapter merged with *CICS/Web selection Guide* chapter
- *CICS Web Application development*, *Changes to Existing applications* and *Portability* chapters consolidated into *Application Development* chapter
- *Administration* chapter removed
- NetCICS removed as a CICS Web solution
- Customer scenarios updated

Third Edition, July 2001

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- CICS TS V2.1 EJB support
- CICS to TCP/IP Sockets Interface
- Patterns for e-business

Changed information

- Application Development for CWS 3270 Web bridge
- Security considerations for CICS Web support (APAR PQ45098)
- Part 3 chapters updated and consolidated into the *CICS Web decision points* chapter in Part I

Fourth Edition, October 2002

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- CICS TS V2.2, Link3270 bridge
- CICS TS V2.2 ECI over TCP/IP support
- CICS TS V2.3, Link3270 bridge extended support
- Miami-Dade County customer scenario

Changed information

- CICS TS V2.2 EJB tooling and security (Chapter 4, *Security* and Chapter 7, *Application development*)
- CICS TG V5 COMMAREA null stripping (Chapter 6, *Performance and scalability*)

Fifth Edition, February 2005

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- SOAP for CICS feature customer scenario
- CICS TS V2.3, Link3270 bridge extended support
- Chapter 6, “Transactional scope” on page 163

Changed information

- SOAP for CICS feature added to all chapters
- J2EE Connector Architecture added to all chapters
- *Matters of State* chapter removed

Sixth Edition, October 2006

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- CICS TS V3.1, CICS Web services support
- CICS Web services support customer scenario
- CICS Service Flow Feature

Changed information

- Chapter 1, “Introduction to SOA and CICS” on page 3
- CICS TG V6 updates
- SOAP for CICS and CICS EJB support moved to an appendix

Part 1

Introduction

In Part 1, we introduce service-oriented architecture (SOA) and its relationship to CICS TS. We provide an introduction to the different styles that can be used when integrating existing CICS assets into an SOA solution. We then introduce and compare the standard architectures and transport mechanisms on which these solutions can be based. Finally, we provide an overview of the technologies and products which can be used to implement CICS SOA solutions.

Introduction to SOA and CICS

This chapter introduces the concept of service-oriented architecture (SOA) and discusses how it applies to System z™. This includes a discussion about the business of SOA and its IT benefits and gives an overview of Web services technologies.

This chapter then describes how you can transform existing CICS assets to play a role in SOA solutions.

The sections in this chapter are:

- Service-oriented architecture on System z
- Transforming CICS assets into SOA solutions
- Interaction between CICS and other core WebSphere SOA products

1.1 Service-oriented architecture on System z

Service-oriented architecture (SOA) is an architecture that organizations and their IT departments are adopting, but what is it? This section gives an overview of SOA, the benefits it introduces, and its positioning on System z.

1.1.1 What is SOA and why adopt it?

This section covers the growing use of SOA and attempts to explain what it is and how organizations can benefit from adopting it.

An introduction to SOA

Rather than being a revolution, SOA is an evolution of best practices and technologies that have gone before. It takes advantage of developments in Internet-based technology and interoperability standards to offer unrivalled business and IT benefits. There have been many definitions for SOA, some clearer than others, but all lead themselves to the concept of loosely coupled business services that are provided in an interoperable and technology agnostic manner.

SOA is an integration architecture approach that is based on the concept of *services*. The business and infrastructure functions that are required to build distributed systems are provided as services that individually or collectively deliver application functions to either user applications or to other services.

Note: SOA is, as its name implies, an *architecture* that allows you to encapsulate business logic and separate it from application logic. It is not a formal specification. To create an SOA *implementation*, you need to use a technology such as Web services or Service Component Architecture (SCA) to make this architecture a reality.

Taking the definition of SOA a little deeper, you can view it from the following *perspectives*:

- A set of *business aligned IT services* that support an organization’s business goals and objectives
- A set of *architectural principles* that address characteristics such as modularity, loose-coupling, and separation of functions
- An *architectural style* that requires a service provider, a service consumer, and a service description
- A set of services that can be combined and choreographed to produce *composite enterprise scale services*

- A *programming model* that comes complete with standards, tools, methods, and technologies, such as Web services

By adopting an SOA approach and implementing it using supporting technologies, companies can build flexible systems that implement changing business processes quickly and can make extensive use of reusable components.

Figure 1-1 shows how services are invoked to support a particular business task or process.

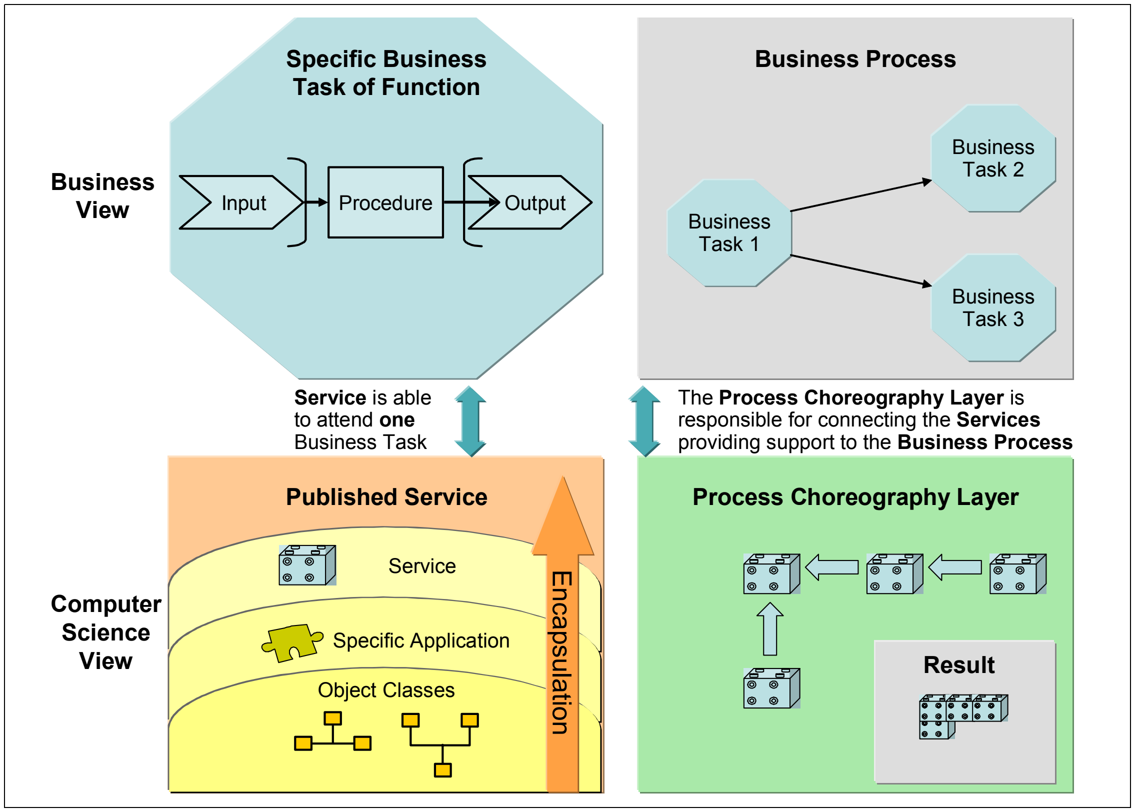

Specific Business Task of Function
Business View
Input
Procedure
Output
Business Process
Business Task 1
Business Task 2
Business Task 3
Service is able to attend one Business Task
The Process Choreography Layer is responsible for connecting the Services providing support to the Business Process
Published Service
Service
Specific Application
Object Classes
Encapsulation
Computer Science View
Process Choreography Layer
Result

Figure 1-1 Mapping services with business tasks or functions

Basic components of an SOA

At the most basic level, an SOA consists of the following three components (Figure 1-2 on page 6):

- Service provider
- Service requester
- Service registry

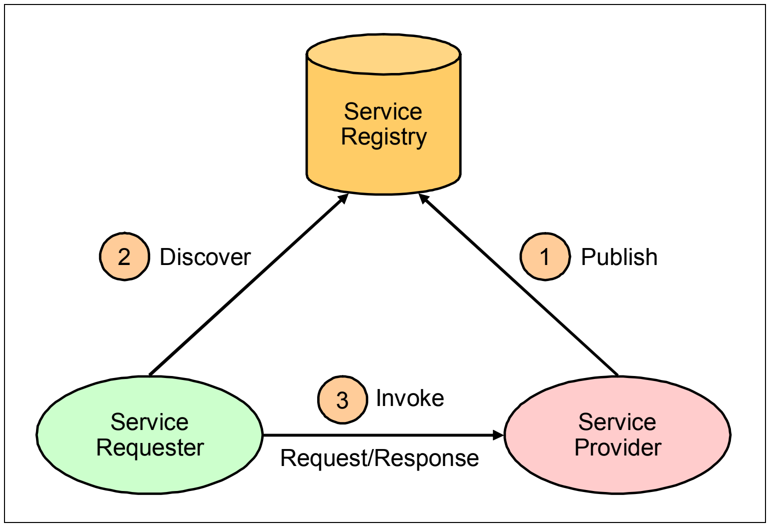


Figure 1-2 SOA components and operations

The *service provider* creates a service and in some cases publishes its interface and access information to a service registry.

Each provider must decide which services to expose, evaluate trade-offs between security and easy availability, determine how to price the services or figure out how to exploit the value of the services if they are free. The provider also has to decide what category the service should be listed, and what sort of trading partner agreements are required to use the service.

The s*ervice registry* is responsible for making the service interface and implementation access information available to service providers.

The implementers of a service registry need to consider the scope in which the registry will be implemented. There are public service registries available over the Internet to an unrestricted audience, as well as private service registries that are only accessible to users within a company-wide intranet.

The *service provider* locates (discovery) entries in the service registry and then binds to the service provider in order to invoke the defined service.

Each component can also act as one of the two other components. For instance, if a service provider needs additional information that it can only acquire from another service, it acts as a service requester.

Defining a service

SOA is an architectural approach to defining integration architectures that are based on the concept of services. A *service* can be described as a function that can be offered or provided to a requester. This function can be an atomic business function or part of a collection of business functions that are wired together to form a process.

There are many additional aspects to a service that must also be considered in the definition of a service within an SOA. The most commonly agreed-on aspects of a service are that:

- Services encapsulate a reusable business function
- Services are defined by explicit, implementation-independent interfaces
- Services are invoked through communication protocols that stress location transparency and interoperability

Ideally, a service should be reusable and should be accessible by more than one requesting application in the architecture. It is, therefore, important to get the service description and reusability correct. For example, a service that offers a calculation such as a home insurance quote could be requested by multiple requesters inside the enterprise and by third parties — as long as the interfaces of the component that offers the service are defined clearly.

Services can be invoked independently by either external or internal service requesters to process simple functions or can be chained together to form more complex functionality to devise new functionality quickly.

Clearly defined interfaces

The interface for SOA should encapsulate only those aspects of process and behavior that are used in the interaction between the service requester and provider. An explicit interface definition, or *contract*, binds a service requester with the provider. The interface should specify only the mutual behavior that is required for the interaction and nothing about the actual implementation of the requester or provider.

This arrangement means that those system aspects where the requester and provider are hosted (their platforms) are independent of the interaction and are free to change. This abstraction allows for flexible improvements to the underlying IT infrastructure.

Communication protocols that stress location transparency

SOA does not specify that the consumer need any specific protocol to have access to a service. A key principle in SOA is that a service is not defined by the communication protocol that it uses but instead, should be defined in a protocol-independent way that allows different protocols to be used to access the same service. Ideally, a service should only be defined once, through a service interface, and should have many implementations with different access protocols. This type of definition helps to increase the reusability of any service definition.

1.1.2 The business and IT benefits of SOA

This section gives a concise view on the business and IT benefits that an organization can gain from adopting an SOA.

Business benefits

Organizations will always seek out innovative means in business and IT to gain competitive advantages. SOA allows the typically heterogeneous IT environment of an enterprise to be agile and responsive to fast changing business conditions.

The following are some of the business advantages to be gained from SOA:

- The concept of components and reuse allows organizations to increase the speed at which they can implement new products and services. By introducing new processes and data, changing existing reusable elements, or recombining them quickly enables technical support and provisioning of new products and services in the marketplace.
- The increased abstraction of business processes from implementation and runtime concerns and constraints mean that there are fewer technical inhibitors that can slow down progress and change.
- The modularity and reuse of components means that services are highly optimized to business needs.
- The ability to *extract more from what is already there* means that organizations are able to introduce new capabilities that bring business advantages. For example, applications that were once siloed can now work together behind the scenes and can help shorten human-based processes and tasks.
- The ability to make available repeatable and reusable services across the enterprise means less duplication of functions and, therefore, reduced instances of duplicated data such as customer details — the ability to improve service quality and retain customers increases with more accurate information.

IT benefits

Because SOA is an approach that specifically aligns IT capabilities to business drivers and needs, the distinction between what is an IT benefit as opposed to a business benefit becomes somewhat blurred. Nevertheless, the IT benefits that an organization can realize by implementing SOA are as follows:

- The adoption of open standards and component-based development brings about long-term reductions in development costs and on-going maintenance.
- The sharing of services and improved consistency reduces duplication of once siloed IT functions and consequently consolidation of hardware and software is made possible, thus reducing costs.
- The revival of core applications through SOA capabilities reduces the need to replace such systems, thereby minimizing risk, disruption, and replacement costs.

Gartner quotes the following benefits of SOA for IT:

> *SOA will shift the focus from tools and packaged suites to modular offerings from multiple vendors that can be assembled and combined by a systems integrator. By 2008, SOA will provide the basis for 80 percent of development projects. By 2008, simple object database access plus service-oriented business applications (SOBAs) will enable Type A organizations to increase code reuse by more than 100 percent. The distinction between software integrators and vendors will blur because packaged applications will be broken up and delivered as SOBAs. In 2006, more than 60 percent of the $527 billion IT professional services market will be based on the exploitation of Web services standards and technology.*

Gartner also says "*SOA shifts developer focus from software to business functions, thereby transforming installed software from an inhibitor to a facilitator of rapid business change.*"

For more information, see *Positions on the Five Hottest IT Topics and Trends in 2005*, which is available at:

```
http://www.gartner.com
```

1.1.3 Web services

Web services are fast becoming the standard for basic SOA implementation. Web services take advantage of existing open-standard Web technologies, such as XML, Uniform Resource Locator (URL), and Hypertext Transfer Protocol (HTTP), and are themselves a set of standards that facilitates open system-to-system communication.

By adhering to Web services standards applications, which are based invariably upon differentiating platforms and technologies, can cooperate through well defined interfaces. Web services follow the SOA philosophy of loose coupling between service requesters and providers. Figure 1-3 illustrates how loose-coupling is maintained within the Web services interaction model.

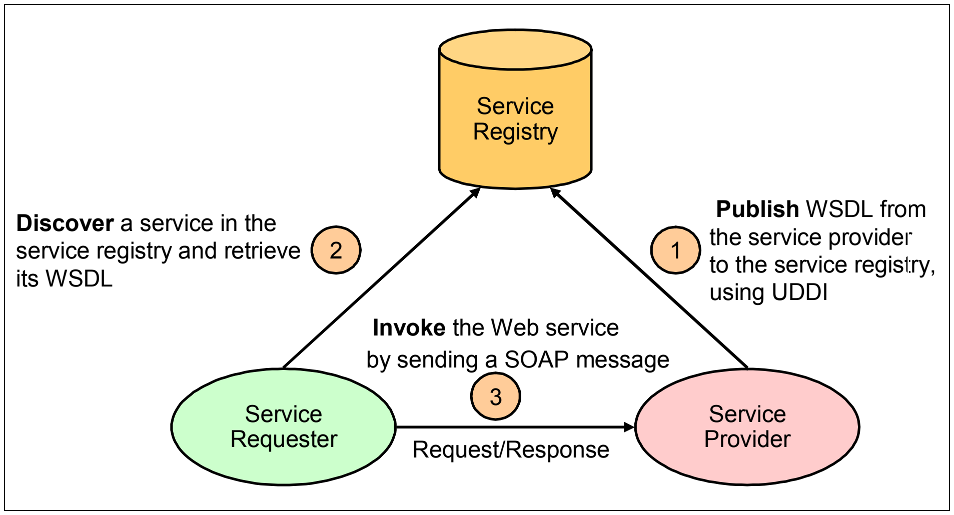

Service Registry
Discover a service in the service registry and retrieve its WSDL
2
1
Publish WSDL from the service provider to the service registry, using UDDI
Invoke the Web service by sending a SOAP message
3
Service Requester
Request/Response
Service Provider

Figure 1-3 Web services invocation model

The interaction shown in Figure 1-3 works as follows:

1. The service provider publishes Web Services Description Language (WSDL) data that defines its interface and location to a service registry, such as a UDDI service registry.
2. The service requester contacts the service registry to obtain a reference to a service provider.
3. The service requester, having obtained the location of the service provider, makes calls on the service provider by sending a SOAP-formatted message.

Basic Web services support provides three simple usage models:

- One-way usage scenario

  A Web services message is sent from a requester to a provider, and no response message is expected.

- Synchronous request/response usage scenario

  A Web services message is sent from a requester to a provider, and a response message is expected.

- Basic callback usage scenario

  A Web service message is sent from a requester to a provider using the 2-way invocation model, but the response is treated only as an acknowledgement that the request has been received. The provider then responds by using a Web service callback to the requester.

Other Web service standards are built upon these basic standards and invocation models to provide higher level functions and qualities of service.

1.1.4 System z and why is it appropriate for SOA

This section gives an overview of System z and the benefits to be gained from implementing SOA capabilities within a z/OS® environment.

System z

IBM eServer™ System z9™ (formerly IBM eServer zSeries) is built on more than 40 years of industry leadership in mainframes. It uses a modular multi-book design that supports one to four books per server. Multiple features such as redundant I/O interconnect (RII) help avoid unplanned interruptions and outages. By increasing secure transaction throughput (SSL), System z9 can improve responsiveness while strengthening security through enhanced encryption and hashing algorithms.

System z contains specialized engines such as z9 Application Assist Processor (zAAP), z9 Integrated Information Processor (zIIP), Integrated Facility for Linux® (IFL), and Internal Coupling Facility (ICF), which can all be used for your advantage. The virtualization and intelligent management features of System z9 109 help reduce management complexity and can facilitate a more efficient use of system resources.

z/OS

IBM System z mainframes are supported by a multitude of operating systems, such as z/OS, z/OS.e, z/VSE™, z/VM®, TPF, and Linux on System z. The flagship operating system of this group is z/OS. z/OS, with its roots in MVS™ and OS/390®, is the flagship mainframe operating system based on the 64-bit z/Architecture™. It is designed to deliver the high qualities of service for enterprise transactions and data, making it appropriate for the larger enterprise.

Some highlights of z/OS V1.7 include the z/OS Workload Manager, which helps balance resources, and Intelligent Resource Director (IRD), which extends Workload Manager and makes it possible to manage resources dynamically across multiple logical partitions. z/OS Parallel Sysplex® technology allows you to balance workloads across multiple servers (up to 32) and is designed to provide near continuous availability.

Why have an SOA framework on z/OS

In a sense, the mainframe environment has always led itself to the concept of SOA because it regards *all* the resources within as providing services. Resources specifically for SOA would be those that provide SOA capabilities such as the Enterprise Service Bus (ESB), Process Management engines, and supporting components such as a base J2EE application server and databases.

To offer the power of System z for SOA, IBM has developed specific z/OS versions of its SOA product suite that is built upon WebSphere Application Server V6 for z/OS. WebSphere Process Server and WebSphere Enterprise Service Bus are z/OS enabled, as are supporting components such as DB2® for z/OS V8. This offers a clean and contained architecture within an z/OS environment, one based upon open and interoperability standards. Additionally products such as CICS Transaction Server have added features to support SOA technologies such as Web services, and can integrate with WebSphere Application Server for z/OS based products.

The advantages of using System z and z/OS for SOA can be seen in three broad categories:

- Quality of service
- Core system transaction capabilities for SOA
- Cost of ownership

Quality of service

A framework that incorporates SOA capabilities exploits well proven System z features such as high scalability, availability, reliability, and security. System z clustering is provided through Parallel Sysplex technology and workload management by zWLM to offer:

- Less than five minutes downtime per year
- 99.999% availability at the application level

System z has built upon four decades of development and collaboration to offer unparalleled security in both its hardware and z/OS operating system. In addition, the introduction of virtualization for z/OS helps to decouple actual physical resources from users and services, bringing an additional layer of protection. For more details about security on System z, refer to:

`http://www.ibm.com/servers/eserver/zseries/security/features.html`

Core system transaction capabilities for SOA

The source of most services that service requesters call upon is most likely core systems such as CICS and IMS™ transactions. These core systems themselves can actually be requesters as well as providers of services. The positioning of these systems within a System z environment means that performance is enhanced because of less network traffic and, in the case of z/OS, the HiperSocket technology is leveraged. To facilitate connections to CICS and IMS for a SOA architecture, the CICS Transaction Gateway and IMS SOAP Gateway Version 9.1 are offered.

Web services can be developed with IBM WebSphere Developer for zSeries tooling to generate Web services artifacts easily.

Cost of ownership

As demand for computer usage increases year-on-year, organizations have tended to introduce new boxes, systems, and applications to their widely heterogeneous and distributed IT environments. Thus, managing these distributed environments can introduce the following hidden costs:

- Increased complexity
- Spiraling resource costs
- Increased downtime costs
- Suboptimized use of resources
- Licensing costs

The Wall Street journal gives an interesting view:

> *Distributed server farms today generate as much as 3,800 watts per square foot, compared to 250 watts per square foot in 1992, with thousands of dollars of cooling capacity needed for each server. Assuming 1,000 distributed servers producing 400 watts each, the electricity bill could hit more than USD 35,000 per month alone. By comparison, a single mainframe z9 generates 312 watts per square foot – one tenth the amount.*

The centralized architecture of the mainframe has always helped avoid such issues, but initial purchase costs and operating costs were high. Recent developments in new technology for System z help to reduce total operating cost (TOC), as follows:

- Virtualization

  Virtualization, which allows a single server or platform to support hundreds of concurrent applications and share data and hardware resources across heterogeneous environments, was invented by mainframes more than 35 years ago. Today, it is highly advantageous for enterprises that are looking for ways to simplify their IT infrastructures and to reduce complexity and costs.

- System z Application Assist Processor (zAAP)

  To help lower costs, IBM has introduced separate processing engines to tackle a collection of mainframe workload types. These engines can free your mainframe CPU for other tasks while lowering related capacity charges. The System z Application Assist Processor (zAAP) engine, released in 2005, reduces costs by processing Java-based application workloads.

- System z Integrated Information Processor (zIIP) engine

  DB2 works in concert with z/OS to tackle workloads that originate on distributed platforms (through DRDA® via TCP/IP) and access DB2 data running on the mainframe. Together, DB2 and zIIP help improve resource optimization for eligible workloads, including those from SAP or other ERP applications, along with CRM and business intelligence initiatives. A zIIP engine can be added for a one-time cost, then it can process up to 40% of such tasks with no additional software or capacity charges.

1.2 Transforming CICS assets into SOA solutions

Existing application assets running in CICS Transaction Server can be utilized in SOA solutions in a number of ways. This section discusses the transformation strategies, and describes how different types of CICS assets can be transformed.

1.2.1 Transformation strategies

The IBM SOA Reference Architecture (shown in Figure 1-4 on page 15) is a technical framework for enterprise transformation that enables software to be delivered as reusable, shareable services. This architecture provides the ability to bridge disparate systems spread across your entire enterprise. Its components are modular, so you can start small and grow your implementation to cover your evolving integration needs, both internally and externally.

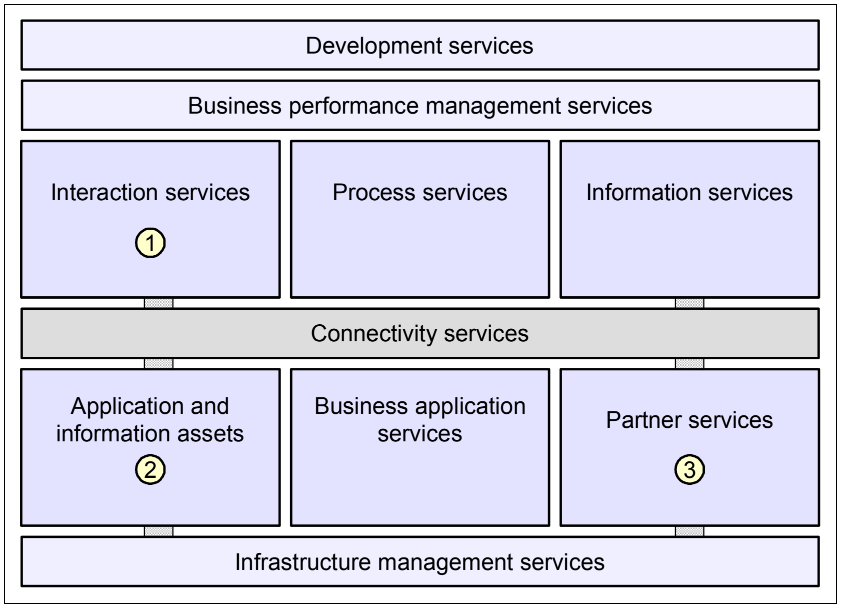

Development services
Business performance management services
Interaction services
1
Process services
Information services
Connectivity services
Application and information assets
2
Business application services
Partner services
3
Infrastructure management services

Figure 1-4 IBM SOA Reference Architecture

We examine three components in Figure 1-4 in closer detail:

- User interface modernization

  This style (denoted by 1 in Figure 1-4) transforms the user experience. It aims to reach new customers while helping to improve productivity and reduce costs. Using this style of transformation can also help reduce training costs and increase overall user satisfaction. This method is the most accessible because it requires the lowest level of investment. You can achieve a rapid return on investment (ROI) through improved user interfaces with a modern interface design, and enhanced productivity with optimized interaction patterns.

- Application integration

  This style (denoted by 2) transforms application connectivity. It aims to extend existing applications beyond their original designs to support integrated business processes, helping to reduce errors and development costs. You can turn existing applications into reusable services that can be accessed by a new set of users or reused to create new front-end business functions. The underlying principle — that you can reuse existing applications with little or no change — offers a lower-risk approach than a replacement strategy, which involves rewriting applications.

- Service orientation

  This style (denoted by 3) transforms the application architecture to provide greater responsiveness to business partners and customers. It involves some re-engineering of the original application. Undoubtedly, this method requires higher investment of resources and time, but gives you the capability to create components from existing applications, which are more flexible and configurable for use in new applications. This reuse of business logic is called componentization and might result in significant cost savings when compared with developing new application code.

1.2.2 Which CICS assets can be transformed?

Over the past 35 years, developers have created two major types of CICS applications, or assets:

- CICS COMMAREA programs
- CICS terminal-oriented programs

CICS COMMAREA programs receive requests and send responses through an area of memory called the COMMunications AREA (COMMAREA). CICS programs can be written in COBOL, PL/I, C, C++, Assembler or Java. In general, CICS COMMAREA programs are similar to subroutines in that they are unaware of how they were invoked. They are often stateless, with CICS - on behalf of the program - automatically managing the transactional scope and security context, which are typically inherited from the caller and a transaction definition.

CICS terminal-oriented programs are sometimes known as *3270 programs* because they are designed to be invoked directly from an IBM 3270 Display Station or similar buffered terminal device. Invocation usually corresponds to a single interaction in an end-user dialog, starting with receipt of a message from the terminal and ending with transmission of a reply message to the same device. Input data from the terminal device is carried in a datastream, which the application acquires through a **RECEIVE** command. After processing, an output datastream is transmitted back to the terminal device through a **SEND** command. Terminal-oriented programs must be capable of analyzing device-specific input data streams and building output data streams to be transmitted to the terminal.

CICS also provides a service known as *Basic Mapping Support (BMS)*, which simplifies application programming for terminals such as the IBM 3270 Display Station. This enables the programmer to define a static layout for each screen to be displayed, with identified fields for dynamic content acquired through a **RECEIVE MAP** command. This in turn causes BMS to analyze the datastream and to return record-formatted data to the application. Similarly, the application presents output data in record format using a `SEND MAP` command, which causes BMS to build an output datastream for the terminal. BMS is widely used because it frees the application programmer from needing knowledge of device specifics and enables applications to be device-independent to some degree.

A *pseudo-conversational* model is normally associated with terminal-oriented transactions. A pseudo-conversational sequence of transactions contains a series of transactions that look to the user like a single conversational transaction involving several screens of input. However, each transaction in the sequence is in fact a single transaction which handles one input, sends back the response, and terminates.

1.2.3 Access to COMMAREA programs

Best practice in CICS application design for a number of years has been to separate the key elements of the application, in particular:

- Client adapt or presentation logic
- Integration logic
- Business logic
- Data access logic

Figure 1-5 shows a transaction made up of these four separate components. A COMMAREA interface (this includes channels with CICS TS 3.1) is used to pass data between the components.

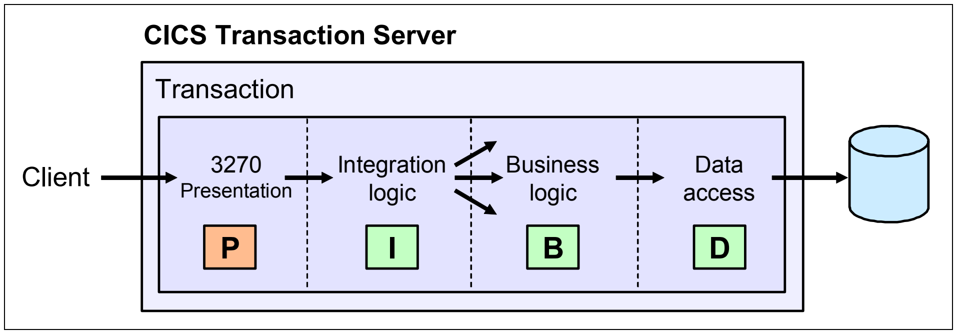


Figure 1-5 Separating key application elements

This separation provides a framework that enables reuse of business logic and data access logic programs as subroutines within a larger application, as well as reuse with alternative implementations of presentation logic (for example a Web service, Web browser, or 3270 device).

CICS COMMAREA programs can be relatively easily enabled for access from a variety of different client applications running on a wide range of platforms. Typical clients include:

- Web service requester
- Java servlet or Enterprise JavaBean (EJB) running in a Java 2 Platform Enterprise Edition (J2EE) application server
- An application running in a Microsoft® .NET environment
- Web browser
- Messaging middleware

In most cases, connections from a client will use a combination of:

- Internal adapters
- External connectors
- Standard *Internet Protocol (IP)* based protocols

An adapter is simply a program that accepts a request and converts the data from an external format to the internal format used by the CICS business logic program. Figure 1-6 shows how a terminal-oriented program and Web service requester can access the same CICS applications. For example, an adapter may convert a SOAP message to a COMMAREA format. The transport mechanism used to invoke the adapter may by synchronous or asynchronous.

An internal adapter is run-time code, possibly generated by a tool that converts from one request format to another, such as converting SOAP over HTTP to a COMMAREA. You can implement the adapter in any language supported by CICS and make it independent of the specific protocol used.

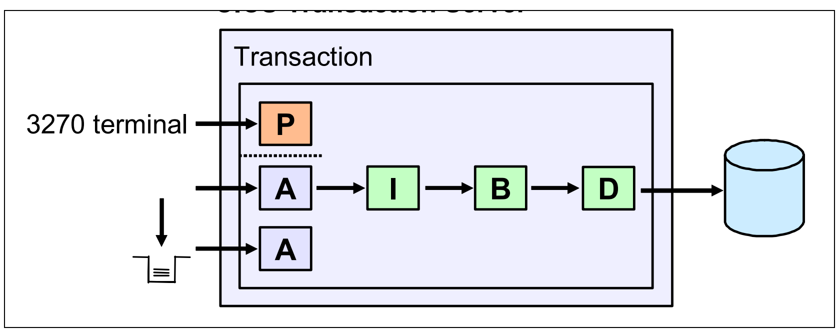


Figure 1-6 Access options provided by CICS facilitate effective reuse of existing business logic

An external connector provides a remote call interface and implements a private protocol to invoke an application running under CICS Transaction Server. You must also use an external adapter to convert data from its external format to the COMMAREA format used by your programs in CICS Transaction Server. The most well-known example of an external connector is IBM CICS Transaction Gateway, which implements the Common Connector Interface (CCI) specified by the J2EE Connector Architecture (JCA), and is used with adapters implemented as Java beans.

Along with these techniques, you can choose to create a standard IP-based adapter that uses a specific transport, such as IBM WebSphere MQ, HTTP and TCP/IP sockets. This approach might be the only available option that supports some types of clients, and it permits greater flexibility in the functionality that can be implemented. However, this flexibility must be balanced against additional development effort, and a loss of generality and reuse, because you can use the adapter only with a specific transport protocol.

Your preferred architectural approach is a key decision because of its effect on the costs of developing the solution and its long-term ROI. However, business factors such as existing development processes and the availability of skills might be as significant as technical factors influencing this decision. It is important to recognize that there is no single right answer suitable for all solutions.

Note: For more information about transforming COMMAREA-oriented CICS programs, see Chapter 3, “CICS access technologies” on page 39.

1.2.4 Access to terminal-oriented programs

Many programs remain that do not have such a clear separation of concerns as COMMAREA programs, combining presentation logic (denoted as P in Figure 1-7 on page 20) and business logic (B) into a single program for which there is only a 3270 interface.

IBM CICS TS V3.1 provides a Link3270 bridge function that neatly addresses this problem. The client uses the Link3270 bridge to run 3270 transactions by linking to the program DFHL3270 and passing a COMMAREA that includes the transaction identifier and the data to be passed to the application. The response contains the 3270 screen data reply. If the target application used BMS, this information is presented in the form of an application data structure (ADS), which is another name for the symbolic map that is generated by the BMS macros used to define the mapping of the 3270 screen. No changes are required for the existing application code, and knowledge of 3270 data streams is usually not needed. As a result, the Link3270 bridge provides a programmatic interface for an important class of terminal-oriented programs, enabling them to be reused without resorting to less-efficient and more-fragile screen scraping.

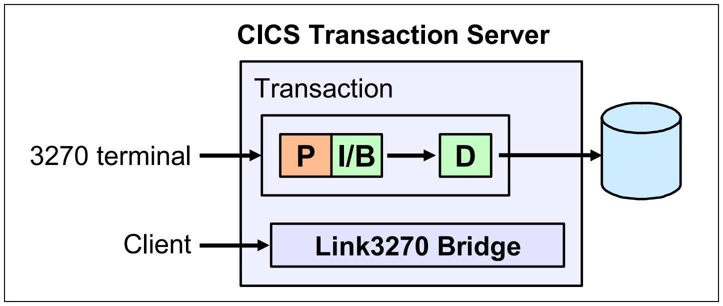


Figure 1-7 Access options provided by CICS facilitate effective reuse of existing terminal-oriented programs

Historically, many 3270 transactions were written as pseudo-conversations, consisting of a number of terminal-oriented programs that run in a defined sequence. Each program in a pseudo-conversation displays data to a user and then terminates, leaving only a small amount of state data to be picked up by the next program in the sequence, which is initiated by the next input data received from the user's terminal. The Link3270 bridge is able to fully reuse these pseudo-conversational transactions.

CICS programs are typically grouped into application suites, or components, for performing a common set of business actions. Identifying the CICS programs that provide flexible public interfaces and understanding these interfaces is the first key step in reuse. The next is to decide the best access options to support your solution.

Note: For more information about transforming terminal-oriented CICS transactions, see Chapter 4, "CICS application access technologies" on page 77.

1.3 Interaction between CICS and other core WebSphere SOA products

CICS Transaction Server provides features to build complete SOA solutions. Service requesters can use these features to gain access to CICS assets. However, CICS TS does not have to be used in isolation to build SOA solutions.

The following products are also key to building SOA implementations, and can be used to interact with CICS TS assets:

- WebSphere Application Server

  Hosts J2EE (Java 2 Enterprise Edition) enterprise applications such as EJB enterprise beans and Web services. Enterprise applications deployed to WebSphere Application Server can interact CICS assets using a variety of techniques described within this redbook.

  For more information about WebSphere Application Server, refer to:

  `http://www.ibm.com/software/webservers/appserv/was/`

- WebSphere Enterprise Service Bus

  Mediates SOAP, JMS, and IIOP messages as they travel between service requesters and service providers. The mediations can provide content-based routing (to pick the service provider that should be used), protocol and message transformation, and so forth. SOA interactions to or from CICS Transaction Server can be mediated in WebSphere Enterprise Service Bus.

  For more information about WebSphere Enterprise Service Bus, refer to:

  `http://www.ibm.com/software/integration/wsesb/`

- WebSphere Process Server

  Manages business processes which incorporate calls to automated activities (such as Web services) and manual activities (such as those completed by a human). Calls to CICS assets can be incorporated into business processes running in WebSphere Process Server.

  For more information about WebSphere Process Server, refer to:

  `http://www.ibm.com/software/integration/wps/`

Note: Each of the three products listed above can all run on both z/OS or distributed platforms.

2

Architectural choices

This chapter is an introduction to the different architectures on which CICS SOA solutions can be based. These include the Web services architecture and J2EE Connector Architecture (JCA).

When deciding which architecture is best for your service application, consider these questions:

- What are the security requirements of the application?
- Is there a requirement for CICS transactional updates to recoverable resources to be synchronized with the updates of an external server?
- What are the performance objectives and what workload management capabilities are required?
- What application interface will be used?
- Will a synchronous or asynchronous invocation be used?
- Will the client and server components be loosely coupled or tightly coupled?
- Do the CICS applications already exist or will they be created from scratch?

2.1 Which architecture should be used to access CICS?

To choose the most suitable architectural approach for a particular application, you need to be able to articulate the functional and non-functional requirements of the application, and map these onto the capabilities of the CICS SOA access options. Because your application is likely to be delivered across several service requesters, you may need to use a range of CICS access options.

The application requirements will typically encompass:

- Security
- Transactional scope
- Performance and workload management
- Application interface
- Synchronous or asynchronous invocation
- Client/server coupling.

Important: The choice of architectural approach is a key decision because it might affect the costs of developing service applications and their long-term value. Business factors, such as the availability of skills, might be as significant as technical factors influencing this decision. It is important to recognize is that there is no single right answer, just as there is no right programming language for all applications.

Security

The first security requirement to consider is how end users and middle tier servers will be authenticated. Simple user ID and password authentication is still widely used, although SSL client certificates, Kerberos tickets and other schemes are becoming popular. Whichever technique is adopted, the user's credentials must eventually be mapped to an external security manager (ESM) user ID in order to support the authorization and accounting requirements that normally apply to CICS applications.

The security characteristics of different CICS SOA access solutions are described in detail in Chapter 5, “Security” on page 101.

Transactional scope

This requirement refers to the capability of a given access option to support *local* transactions (one-phase commit), enabling a number of updates performed by CICS applications to be processed as a single unit of work; or *global* transactions (two-phase commit), enabling an external server to coordinate updates performed by CICS with updates to local resources held by that server.

The Web services and JCA architectures both support global transactions. CICS provides support for the WS-Atomic Transaction specification, allowing us to tie together a client transaction and the invoked CICS transaction.

The transactional characteristics of different CICS SOA access solutions are described in detail in Chapter 6, “Transactional scope” on page 163.

Performance and workload management

Response time and CPU cost per transaction are important aspects of performance in a production system. CICS seeks to minimize these and is highly optimized for traditional styles of access, such as 3270 terminal access over a System Network Architecture (SNA) network. Most SOA solutions require additional elements such as connectors, adapters and encrypted data flows. These are less optimized, and impose an overhead on the execution of the target business program.

Workload Management is the process of spreading multiple requests for work over the resources that can do the work. It optimizes the distribution of processing tasks therefore improving performance, scalability, and reliability of an application. It also provides failover when servers or systems are not available.

The performance and workload management characteristics of different CICS SOA access solutions are described in detail in Chapter 7, “Performance and scalability” on page 201.

Application interface

The access option needs to support the CICS program interface either directly or via an adapter without imposing additional restrictions. The interface encompasses the data elements, the code page, and the size of the message. The application interface of some access options are currently restricted by architectural limits, for example, solutions which use the JCA are limited to a 32 KB COMMAREA.

Note: While the COMMAREA can hold up to 32 KB of certain data applications, such as CICS TG, it cannot utilize the full 32 KB available since ECI control blocks must also be contained within the 32 KB.

Application development considerations, including the type of application interface, for the different CICS SOA access solutions are described in detail in Chapter 8, “Application development” on page 245 and data conversion considerations are described in detail in Chapter 9, “Data conversion” on page 273.

Synchronous or asynchronous invocation

The majority of access options support synchronous invocation, meaning that a client request receives a single reply from CICS and the client waits for the reply. In the alternative approach, known as asynchronous invocation, the client request is not responded to immediately and the client may continue processing before receiving the response. A polling or event-based mechanism is required so that the response can be obtained at a later time. In some cases, an immediate response is sent from the server to the client confirming receipt of the original request, indicating therefore that the application response will be delayed.

In general, asynchronous access solutions are more robust. Planned or unplanned outages, software upgrades and other operational events have less impact on a client's ability to send requests. Synchronous access solutions, on the other hand, normally offer better interoperability, for example, closer coordination of transactional updates.

Client/server coupling

Some access options are described as *tightly coupled*, while others are described as *loosely coupled*. Tight coupling implies that the client and server share many assumptions and dependencies. Loose coupling is not a precise concept but refers to several possibilities:

- Self-describing messages, that is, the format of the data messages exchanged between the client and server programs is described separately from the programs themselves
- Implementation independence, that is, the client may use dissimilar technology from the server program, thus offering operating system, middleware, programming language independence
- Server location independence, that is, the client program does not need to be changed when the location of the server program changes

CICS has a range of access options that support the diverse requirements of the service requesters that we have just discussed. These CICS SOA access options can be based on the following standard architectures:

- Web services architecture
- J2EE Connector Architecture (JCA)

We discuss these standard architectures in 2.2, "Standard architectures" on page 27.

Standard transports are suitable for use by applications that require greater control of the protocol and do not need the development tools provided by the standard architectures. The standard transports are:

- WebSphere MQ
- HTTP
- TCP/IP Sockets (the TCP/IP socket interface for CICS)

We discuss these standard transports in 2.3, “Standard transports” on page 33.

2.2 Standard architectures

This section introduces the major architectures which can be used to build CICS SOA solutions. Solutions based on these architectures benefit from the comprehensive set of development tooling which is provided to help in the generation of applications.

Standard architectures such as the JCA provide built-in support for qualities of service such as management of security and transactions. These qualities are slowly being introduced into the Web services architecture. For example, the WS-Security specification, which provides for message level security, and the WS-Atomic transaction, which provides two-phase commit transactional functionality, are now standard.

2.2.1 Web services

Web services are an implementation of a service-oriented architecture. A *service* is an application component which has a well-defined published interface that allows other application components to invoke operations on the service without any knowledge of how the service is implemented.

The technologies that can be used to implement a Web services solution are relatively new but they have received wide acceptance as the strategic way of building distributed IT solutions which integrate heterogeneous applications over the Internet.

The Web service specifications are completely independent of programming language, operating system, and hardware in order to promote loose coupling between the service requester (or consumer) and service provider. The technology is based on open standards such as:

- eXtensible Markup Language (XML)
- SOAP - a standard protocol for exchanging XML messages
- Web Services Description Language (WSDL), which defines an XML grammar for describing Web services
- Universal Description, Discovery and Integration (UDDI), a registry mechanism that can be used to look up Web service descriptions

Using open standards provides broad interoperability among different vendor solutions. These principles mean that companies can implement Web services without having any knowledge of the service requesters, and vice-versa. This facilitates just-in-time integration and allows businesses to establish new partnerships easily and dynamically.

Figure 2-1 shows how a SOAP message consists of an envelope containing zero or more headers and exactly one body. Application designers determine the contents of the headers; the SOAP specification itself does not define what headers should be used. For example, application designers might define a header that contains authentication credentials or information for transaction management. The body is where the main end-to-end information (the payload) conveyed in a SOAP message must be carried. This information might be parameters for calling a service (for a service request) or the result of calling the service (for a service response).

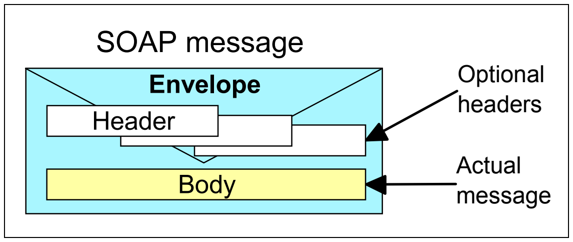


Figure 2-1 SOAP message

The major advantages of SOAP are as follows.

- It provides a standard for exchanging data in XML format, for example, the parameters used in a program call (for the inbound message) and the data resulting from the call (for the outbound message).
- It is protocol, platform, operating system and programming language independent.

- It is flexible and extensible.
- It enables the use of Web services standards such as WS-Security.

SOAP supports the remote procedure call (RPC) style of Web service, as well as the document message style. Although it is transport protocol independent, HTTP is the most widely used protocol today for transporting SOAP messages.

A Web service is fully defined in a Web Services Description Language (WSDL) file. Usually, tools are used to import the WSDL file and generate a proxy for the service requester to use to construct and send the SOAP message.

Application programs running in CICS TS V3.1 can participate in a heterogeneous Web services environment as service requesters, service providers, or both, using either an HTTP transport or a WebSphere MQ transport. Figure 2-2 shows an outline of the Web services support in CICS

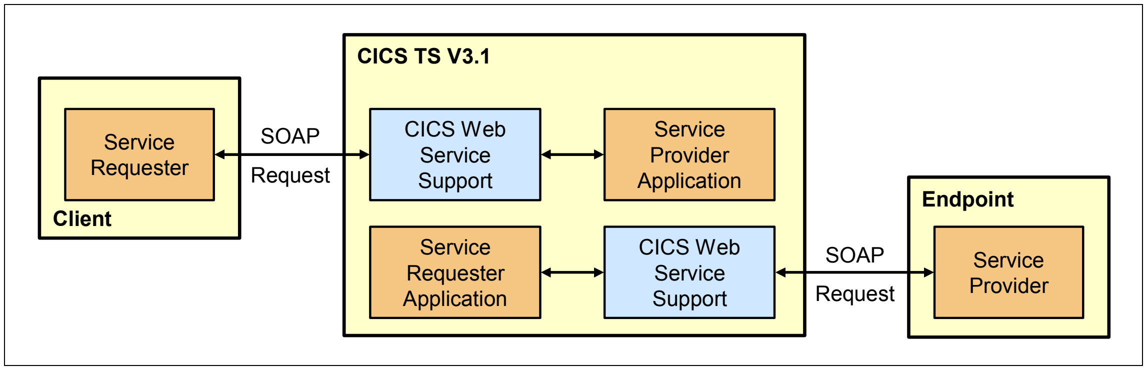


Figure 2-2 Web Services support in CICS V3.1

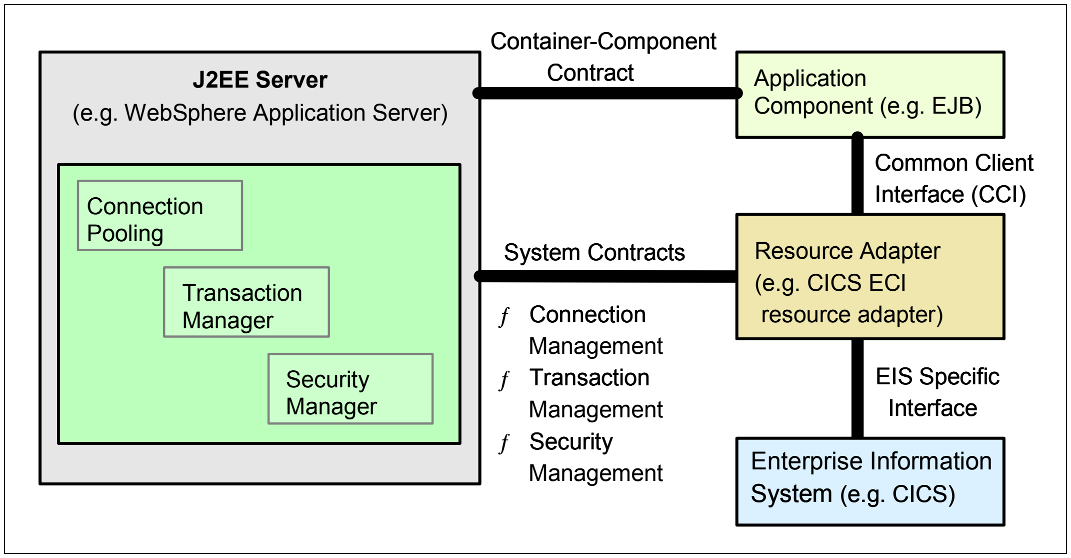
Refer to 3.1.1, "CICS Web services support in CICS TS V3.1" on page 40 for more information about using CICS Web services.

2.2.2 J2EE Connector Architecture (JCA)

The J2EE Connector Architecture (JCA) defines a standard for connecting from the J2EE platform to heterogeneous Enterprise Information Systems (EIS). CICS is an example of an EIS.

The JCA enables an EIS vendor to provide a standard *resource adapter* which is a middle-tier connector between a Java application and an EIS, which permits the Java application to connect to the EIS.

JCA V1.5 defines a number of components that make up this architecture, shown in Figure 2-3 on page 30.



Container-Component Contract
J2EE Server
(e.g. WebSphere Application Server)
Connection Pooling
Transaction Manager
Security Manager
Application Component (e.g. EJB)
Common Client Interface (CCI)
System Contracts
ƒ Connection Management
ƒ Transaction Management
ƒ Security Management
Resource Adapter (e.g. CICS ECI resource adapter)
EIS Specific Interface
Enterprise Information System (e.g. CICS)

Figure 2-3 J2EE Connector Architecture component structure

Figure 2-3 shows the JCA being used in a managed environment, that is, the application is running in WebSphere Application Server. In this case, management of connections, transactions and security are managed by the application server. The JCA can also be used in an non-managed environment, in which case the application must manage connections, transactions and security itself.

Note: We strongly recommend you use a managed environment over a non-managed environment. The application development costs for a non-managed environment are significant and the quality of service generally not as good as that provided by a managed environment such as WebSphere Application Server.

The Common Client Interface (CCI) defines a common application programming model for interacting with resource adapters and is independent of any specific EIS. Of course, this does not mean a developer can write exactly the same code to access one EIS (for example, CICS) as he writes to access another EIS (for example, an IMS system). However, the generic CCI classes are the same in that they are independent of the EIS, whereas specific EIS classes cater to the differences. For example, the parameters used to call a CICS program are different from those used to invoke an IMS transaction, but the programming model is the same - independent of the EIS. As a result, you can increase developer productivity when developing applications to communicate with multiple EISs. The CCI programming interface is similar to other J2EE interfaces, such as the JDBC™ (Java Database Connectivity) interface or JMS (Java Message Service) interface.

Before the existence of the JCA, IBM recognized a need for a common way to connect to EIS systems and introduced the Common Connector Framework (CCF), JCA is an effective replacement now for the CCA. The JCA provides similar function to the CCF, but it is an open specification that can be implemented by any EIS vendor. In addition, the JCA provides a stronger emphasis on system contracts and qualities of service, allowing closer integration between the J2EE application server and the EIS.

Resource adapters

The CICS Transaction Gateway (CICS TG) provides two ECI (External Call Interface) resource adapters.

cicseci.rar	The CICS ECI resource adapter is provided with both CICS TG for multiplatforms and CICS TG for z/OS. It supports the LocalTransaction interface.
cicseciXA.rar	The CICS ECI XA resource adapter that is provided with CICS TG for z/OS only as of V6.1. It provides full support for global transactions by implementing the XAResource interface.

The ECI resource adapter is the simplest to use and the most commonly used CICS TG resource adapter. Support is provided both for synchronous and asynchronous calls. However, asynchronous calls using the CICS ECI resource adapter have their limitations, for example, you cannot make several concurrent calls and then wait for the response. You must take the response of each previous call, before making another call.

The JCA resource adapters provided by the CICS TG are effective replacements for the CICS ECI Java classes. Support for the ECI resource adapters is included in the Rational® Software Development Platform series of products, whereas tooling support for direct use of the ECI Java classes is not.

System contracts

The JCA defines a standard set of system-level contracts between a J2EE application server and a resource adapter. The standard contracts include:

- A *connection-management* contract that provides a consistent application programming model for connection acquisition and enables a J2EE application server to pool connections to a back-end EIS. This leads to a scalable and efficient environment that can support a large number of components requiring access to an EIS system.

- A *transaction-management* contract that defines the scope of transactional integration between the J2EE application server and an EIS that supports transactional access. This contract allows a J2EE application server to use a transaction manager to manage transactions across multiple resource managers (known as global transactions). This contract also supports the LocalTransaction interface, which refers to transactions that are managed internal to a resource manager without the involvement of an external transaction manager.
- A *security-management* contract that enables secure access to an EIS. This contract provides support for a secure application environment, which reduces security threats to the EIS and protects valuable information resources managed by the EIS. Both container-managed sign-on (in which the J2EE application server is responsible for flowing security context to the EIS) and component-managed sign-on (in which the application is responsible for flowing security context to the EIS) are supported.

  When used with WebSphere Application Server for z/OS, the CICS ECI resource adapter enables automatic propagation of security credentials from the application server to CICS. This functionality is known as *thread identity support*.

These system contracts are transparent to the application developers, which means they do not have to implement these services themselves. In a managed environment it is these system contracts that make the JCA such a powerful solution for integrating existing CICS applications with new J2EE applications running in an application server such as WebSphere Application Server.

Figure 2-4 shows how the CICS TG enables SOA access to a CICS business logic program.

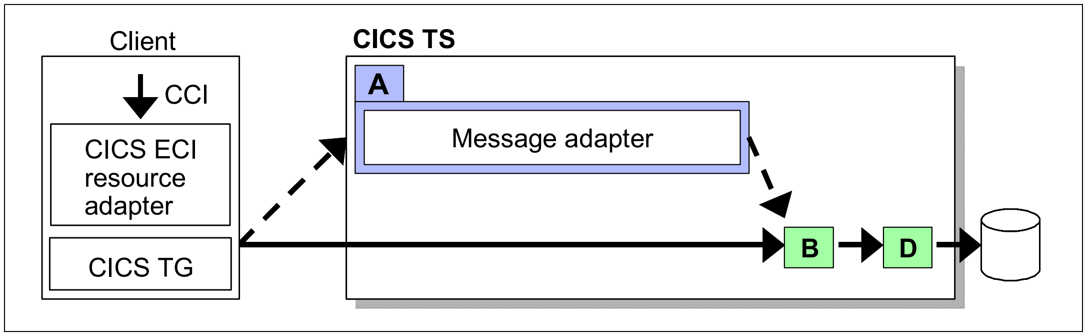


Figure 2-4 SOA access to CICS using the JCA

A J2EE application uses the CCI programming interface to invoke the CICS ECI resource adapter. The CICS TG ECI classes are packaged with the ECI resource adapter and are used to pass the application request to the CICS TG.

The J2EE application can invoke the CICS business logic program (B) directly if no message transformation is required. In this case, Rational Application Developer can be used to create a Java bean to represent a COMMAREA formatted as COBOL types, with Java methods for getting and setting fields.

A message adapter in CICS is required only if the message is to be transformed, for example the request is in XML and the CICS business logic program requires a COBOL record format. The length of the message is subject to the normal CICS COMMAREA message length limitation of 32 KB.

The CICS TG is the preferred implementation for JCA connectors to access all CICS servers from WebSphere Application Server, for applications that require a high performing, secure and scalable access option with tight integration to existing CICS applications. The CICS TG benefits from ease of installation, flexible configuration options, and requires minimal changes to CICS and in most cases no changes to existing CICS applications. In addition, the CICS TG supports a range of non Java clients, including C, C++, COBOL and COM.

The JCA is considered a medium coupling architecture, compared to the EJB architecture (high coupling) and the Web services architecture (low coupling). The JCA can be used with a diverse range of supported environments and different deployment options; these are described in detail in 3.1.2, “CICS Transaction Gateway” on page 48.

2.3 Standard transports

Standard transports are suitable for use by applications that require greater control of the protocol and do not need the development tools which are generally provided by the standard architectures. These applications will also normally assume more responsibility for security, transactions and recovery.

2.3.1 WebSphere MQ

WebSphere MQ allows you to easily exchange information across different platforms, integrating existing business applications in the process. WebSphere MQ assures reliable delivery of messages, dynamically distributes workload across available resources, and helps to make programs portable.

WebSphere MQ provides Java Message Service (JMS) APIs and native WebSphere MQ APIs for use by service requesters on a wide variety of platforms, with many options for routing and encrypting messages prior to arriving on WebSphere MQ for z/OS.

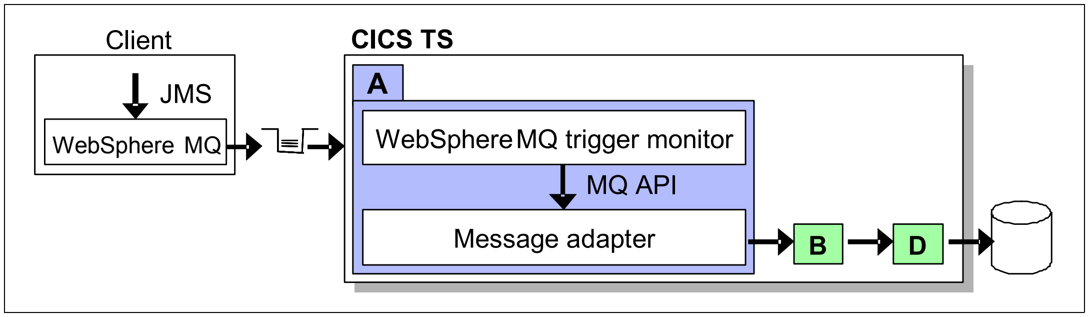


Figure 2-5 SOA access to CICS using WebSphere MQ

Figure 2-5 shows the WebSphere MQ trigger monitor program provided by CICS, which can be used to automatically start an appropriate message adapter program when messages arrive. The message adapter uses WebSphere MQ native APIs to receive the message, transform it if required and call the business logic program. A reply message can be sent using the reply-to queue defined in the message. For efficiency, the message adapter program will usually continue to process messages on the inbound queue until it is empty.

The WebSphere MQ DPL bridge for CICS provides an alternative option (see Figure 2-6). This generic adapter passes a message from a named input queue to a business logic program through the COMMAREA. This is ideal in the situation where the service requester can format the message into a form acceptable by the business logic program.

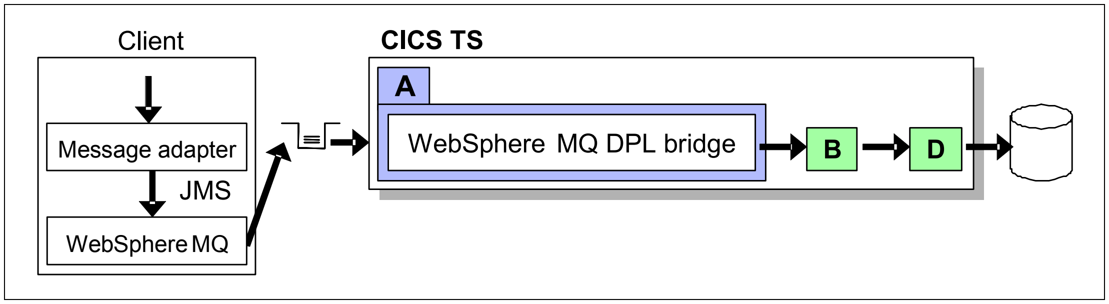


Figure 2-6 SOA access to CICS using the WebSphere MQ DPL bridge

When using the WebSphere MQ DPL bridge, the client application writes a structured message to the queue. This message must contain information in a predefined format that the monitoring transaction can use to decide how to handle the message. Several formats are possible, each starting with a block of data called an MQMD header. This field contains control information used by the monitoring transaction like the message format type, along with optional information, such as a reply-queue identifier and a user ID.

For more information about the use of WebSphere MQ with CICS, refer to 3.2.1, “WebSphere MQ” on page 64.

2.3.2 HTTP

Figure 2-7 shows how the HyperText Transmission Protocol (HTTP) can be used directly with CICS TS. CICS Web support provides an HTTP listener and a message adapter program can be written using standard CICS APIs.

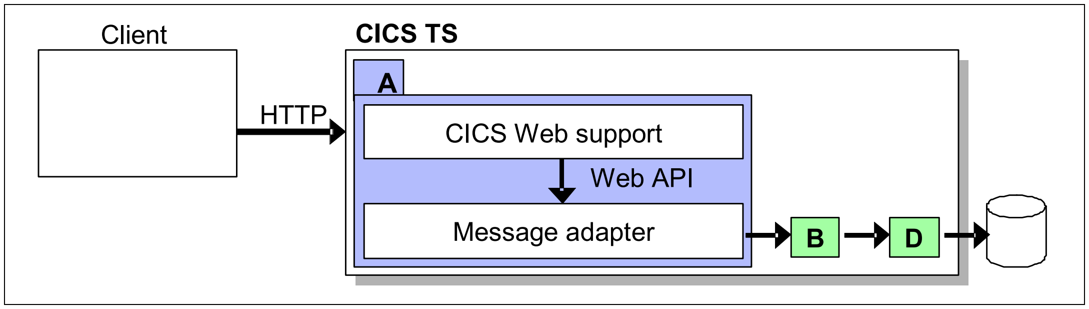


Figure 2-7 SOA access to CICS using CICS Web support

CICS supports HTTP Basic Authentication for user ID identification or the more secure SSL encryption and authentication with client and server certificates. CICS Web support sets up the transaction and security environment and calls the message adapter. The message adapter uses the CICS WEB APIs to extract the HTTP user data, which is typically formatted as an HTML form. The message adapter has access to the HTTP and TCP/IP headers if required. The message adapter transforms this information into a COMMAREA and calls the business logic program.

Note: CICS TS currently supports the HTTP 1.0 and HTTP 1.1 specifications.

If the service requester is a Web browser, the response message will typically be formatted as HTML. If the service requester is an application, the response message will normally be formatted as XML. The message adapter can use the CICS DOCUMENT APIs to easily merge static HTML or XML with dynamic data from the business logic program. The response is returned to the client for display or processing.

HTTP is synchronous and stateless. However, if state management is required, CICS provides a utility for storing state data indexed by a state management token that the HTTP client can return on subsequent calls in order to retrieve the state.

CICS Web support also allows a CICS application to initiate an HTTP request and to receive the response from an HTTP server program, thus providing bi-directional support for the HTTP protocol.

For more information about CICS Web support, refer to 3.1.4, "CICS Web support" on page 60.

2.3.3 TCP/IP Sockets

The TCP/IP Socket Interface for CICS (sometimes referred to as "CICS sockets") is provided by z/OS Communications Server and supports peer-to-peer applications in which both ends of the connection are programmable (see Figure 2-8). CICS sockets provide a variant of the Berkeley Software Distribution 4.3 Sockets interface, which is a low-level API with no built-in support for transactions or security.

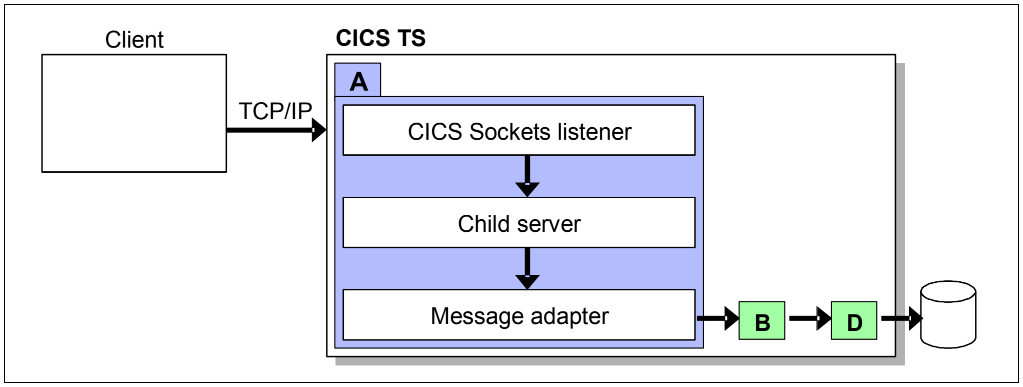


Figure 2-8 SOA access to CICS using CICS sockets

CICS sockets provide a concurrent listener, or you can write your own listener to meet your needs. The listener and child server use the CICS Sockets APIs to receive and send data, and perform general communications control functions. The programs can be written in COBOL, PL/I, assembler language or C. Client adapters can be written to create new outbound connections.

For more information about CICS sockets, refer to 3.2.5, "CICS sockets" on page 73.

2.4 Conclusion

CICS provides a range of access options to support modern connectivity architectures, such as Web services and J2EE, and other standard transport mechanisms. With the right external connectors and internal adapters, you can maximize the reuse of your existing mission-critical CICS assets. The following table compares the connection architectures and standard transport mechanisms discussed in this chapter.

Table 2-1 Common architectures and standard transport mechanism

	Capabilities	**Security to IBM System z**	**Transactional scope**	**Interface**	**Coupling**
Standard architecture					
Web services	► Inbound and outbound ► Synchronous (HTTP) ► Asynchronous (WebSphere MQ)	► Web services ► SSL ► User ID and password	► Web services ► Sync on return	► COMMAREA ► CONTAINER	Low
JCA	► Inbound only ► Synchronous ► Asynchronous ► 32 KB maximum message size	► SSL ► User ID and password ► Thread identity	► Local ► Global ► Sync on return	► COMMAREA	Medium

	Capabilities	**Security to IBM System z**	**Transactional scope**	**Interface**	**Coupling**
Standard transport					
WebSphere MQ	► Inbound and outbound ► Asynchronous ► Assured delivery	► SSL ► User ID and password	Sync on return	► COMMAREA ► WebSphere MQ API	Medium
HTTP	► Inbound and outbound ► Synchronous	► SSL ► User ID and password	Sync on return	CICS Web API	Medium
TCP/IP Sockets	► Inbound and outbound ► Synchronous ► Asynchronous	► SSL ► User ID and password	Sync on return	CICS sockets API	High

Both CICS Transaction Server and WebSphere Application Server are strategic middleware products that interoperate well using technologies, such as Web services, to support end-to-end on demand systems. They exploit and complement z/OS qualities of service, such as high availability and scalability at low cost per transaction, with a high level of security. In combination, WebSphere Application Server and CICS support almost any mission-critical SOA solution.

3

CICS access technologies

In this chapter, we introduce the CICS access technologies. We provide product information as well as an overview of the components and topologies.

The three CICS access technologies we discuss here and again in later chapters are:

- CICS Web service support
- CICS Transaction Gateway (CICS TG)
- CICS Web support

In addition to the above technologies, we also introduce other technologies which might be appropriate for some business scenarios. These are not discussed in Part 2 of this book, but we give references to existing Redbooks and other publications. The other technologies are:

- WebSphere MQ
- CICS Enterprise JavaBeans™ (EJB) support
- SOAP for CICS feature
- Host Access Transformation Services
- TCP/IP Socket Interface for CICS

3.1 CICS access technologies

In the following sections, we provide a short introduction to the strategic CICS access technologies discussed in this book.

- CICS Web service support
- CICS Transaction Gateway (CICS TG)
- CICS Web support

3.1.1 CICS Web services support in CICS TS V3.1

Application programs running in CICS TS V3.1 can participate in a heterogeneous Web services environment as service requesters, service providers, or both, using either an HTTP transport or a WebSphere MQ transport. Figure 3-1 shows an outline of the Web services support in CICS.

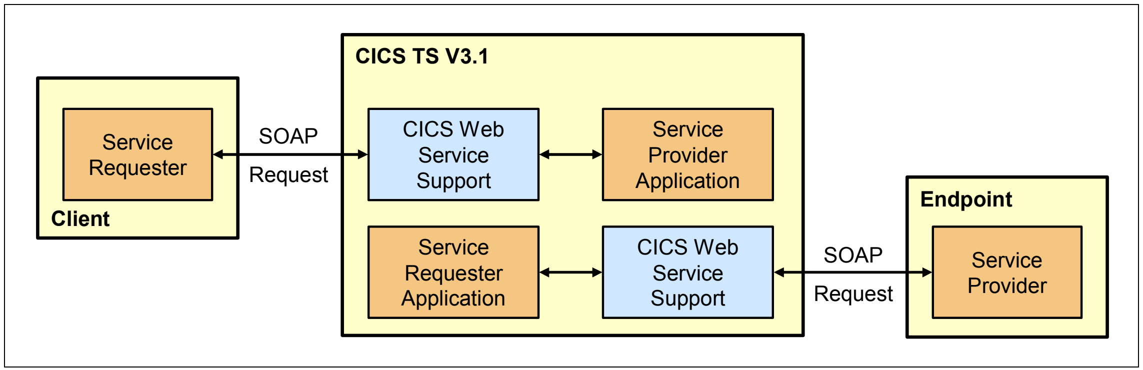


Figure 3-1 Web Services support in CICS TS V3.1

The components of a CICS Web services solution are:

- A PIPELINE process

  This controls processing of SOAP messages (SOAP is the message format used to communicate between a service requester and a service provider).

- Tooling to create Web services artifacts

  CICS provides a set of batch tools called the Web Services Assistant used to create the WSDL and WSBIND files required for a CICS Web services solution. Alternatively, these can be created using the WebSphere Developer for System z integrated development environment.

- CICS resources for management

  RDO (CICS Resource Definition Online) database for managing CICS Web services solutions

- A new EXEC CICS API for Web services

Functions of the CICS Web services support in CICS TS V3.1

This section provides more detail about the functions provided by CICS Web services support:

- Web Services Assistant utility for Web services development.

  The Web Services Assistant utility contains two programs, DFHWS2LS and DFHLS2WS. DFHWS2LS helps you map an existing WSDL document into a high-level programming language data structure, while DFHLS2WS creates a new WSDL document from an existing language structure. The Web Services Assistant supports the following programming languages:

  - COBOL
  - PL/I
  - C
  - C++

- Two different approaches to deploying your CICS applications in a Web services environment.

  - When using the Web Services Assistant a file called the WSBIND file is generated. When your application runs, CICS uses the WSBIND file to transform your application data into a SOAP message on output and to transform the SOAP message to application data on input.

    This option is also available when using the tooling available in the XML Services for the Enterprise (XSE) support in WebSphere Developer for System z.

  - You can take complete control of the processing of your data.

    You can write your own code to map between your application data and the message that flows between the service requester and provider. For example, if you want to use non-SOAP messages within the Web service infrastructure, you can write your own code to transform between the message format and the format used by your application.

- Pipeline configuration file to determine which message handlers should be invoked.

  A pipeline can be configured as a service requester pipeline or a service provider pipeline but not both.

  Whether you use the Web Services Assistant or take complete control of the processing yourself, you can write your own message handlers to perform additional processing on your request and response messages, or you can use CICS-supplied message handlers.

- CICS supplied message handlers to help you process SOAP messages.

  The pipelines that CICS uses to process Web service requests and responses are generic, in that there are few restrictions on what processing can be performed in each message handler. However, many Web service applications use SOAP messages, and any processing of those messages should comply with the SOAP specification. Therefore, CICS provides special SOAP message handler programs that can help you to configure your pipeline as a SOAP node.

  – A service requester pipeline is the initial SOAP sender for the request, and the ultimate SOAP receiver for the response.

  – A service provider pipeline is the ultimate SOAP receiver for the request, and the initial SOAP sender for the response.

  You cannot configure a CICS pipeline to function as an intermediary node in a SOAP message path.

  The CICS-provided SOAP message handlers can be configured to invoke one or more user-written header processing programs and to enforce the presence of particular headers in the SOAP message.

- Many and different pipelines within a single CICS region.

  You can configure a pipeline to support SOAP 1.1 or SOAP 1.2. Within your CICS system, you can have some pipelines that support SOAP 1.1 and others that support SOAP 1.2.

- New resource definitions to help configure support for Web services:

  – PIPELINE
  – URIMAP
  – WEBSERVICE

  If you used the SOAP for CICS feature, you may be able to use CICS resource definitions to replace the logic you provided in your pipeline programs to distinguish one application from another. For example, in a service provider, you may be able to replace code that distinguishes between applications based on a URI, with a suitable set of URIMAP resources.

- Provides new EXEC CICS application programming interface (API) commands:

  – SOAPFAULT ADD | CREATE | DELETE
  – INQUIRE WEBSERVICE
  – INVOKE WEBSERVICE

- Conforms to open standards including:

  – SOAP 1.1 and 1.2
  – HTTP 1.1
  – WSDL 1.1

- Web service interoperability

  It ensures maximum interoperability with other Web services implementations by conforming with the Web Services Interoperability Organization (WS-I) Basic Profile 1.0. This profile is a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications, which, taken together, promote interoperability between different implementations of Web services.
- It supports the WS-Atomic Transaction specification.
- It supports the WS-Security specification.

Figure 3-2 shows an overview of how the component of the CICS Web services support fits together.

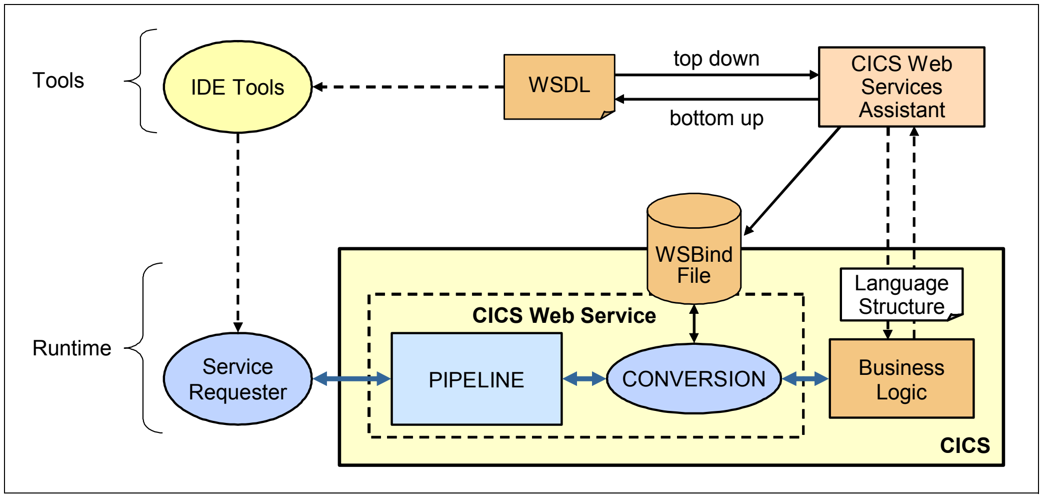


Figure 3-2 Overview of CICS Web services support

CICS as a service provider

When CICS is a service provider, it receives a service request, which is passed through a pipeline to a target application program. The response from the application is returned to the service requester through the same pipeline. In this section we discuss how to prepare for running a CICS application as a service provider, and how CICS processes the incoming service request.

Preparing to run a CICS application as a service provider

Suppose that we have an existing CICS application that we want to expose as a Web service that uses the HTTP transport. Suppose also that we want to use the

Web Services Assistant rather than taking control of the processing ourselves. We go through the following high level steps in Table 3-1.

Table 3-1 Expose CICS artefact as a Web service

Step	Action	Role
1.	Generate WSBIND and WSDL files	Application Developer
2.	Create a TCPIPSERVICE resource definition	System Programmer
3.	Create a PIPELINE resource definition and a PIPELINE configuration file	System Programmer
4.	Install TCPIPSERVICE and PIPELINE definitions - WEBSERVICE and URIMAP definitions created automatically by CICS	System Programmer
5.	Publish the WSDL	System Programmer

Figure 3-3 on page 45 shows how the various artefacts relate to each other and how the dynamic install of resources occurs.

- The batch tools create the WSBIND and WSDL files these are placed in the 'Pick up' directory.
- A PIPELINE is configured using an XML config file (or one of the samples provided with CICS) and a pointer to the pickup directory.
- On installing the PIPELINE a WEBSERVICE resource is created and installed for every WSBind file found in the pickup directory.
- For every WEBSERVICE resource that is installed a corresponding URIMAP resource is created and installed referencing the WEBSERVICE and PIPELINE.

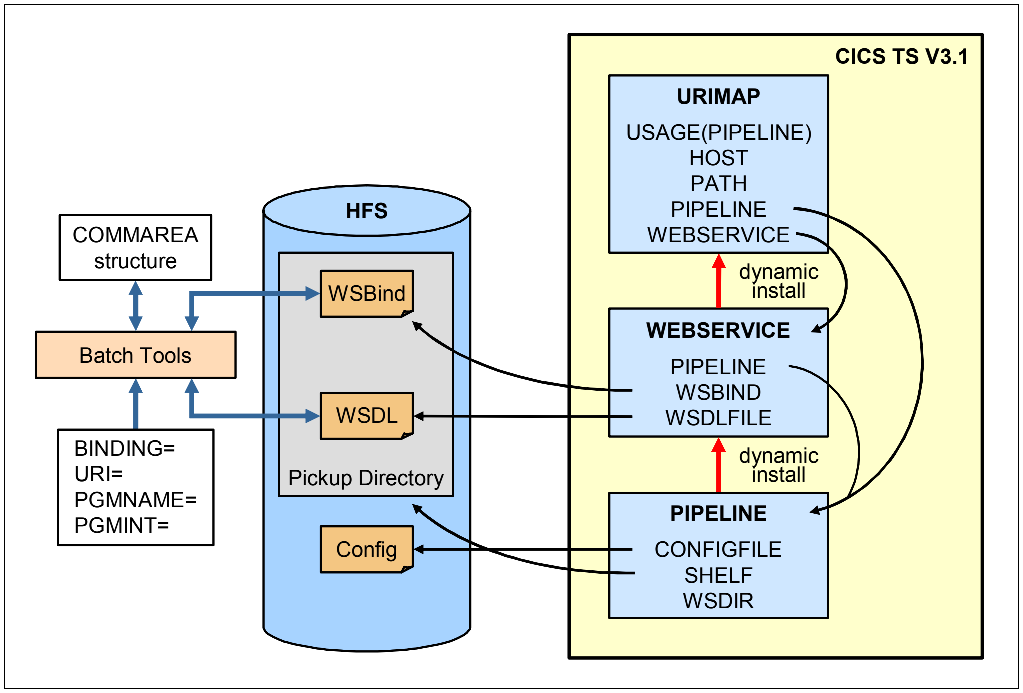


Figure 3-3 CICS Web services artefacts

Processing the inbound service request

Figure 3-4 on page 46 shows the processing that occurs when a service requester sends a SOAP message over HTTP to a service provider application running in a CICS TS V3.1 region.

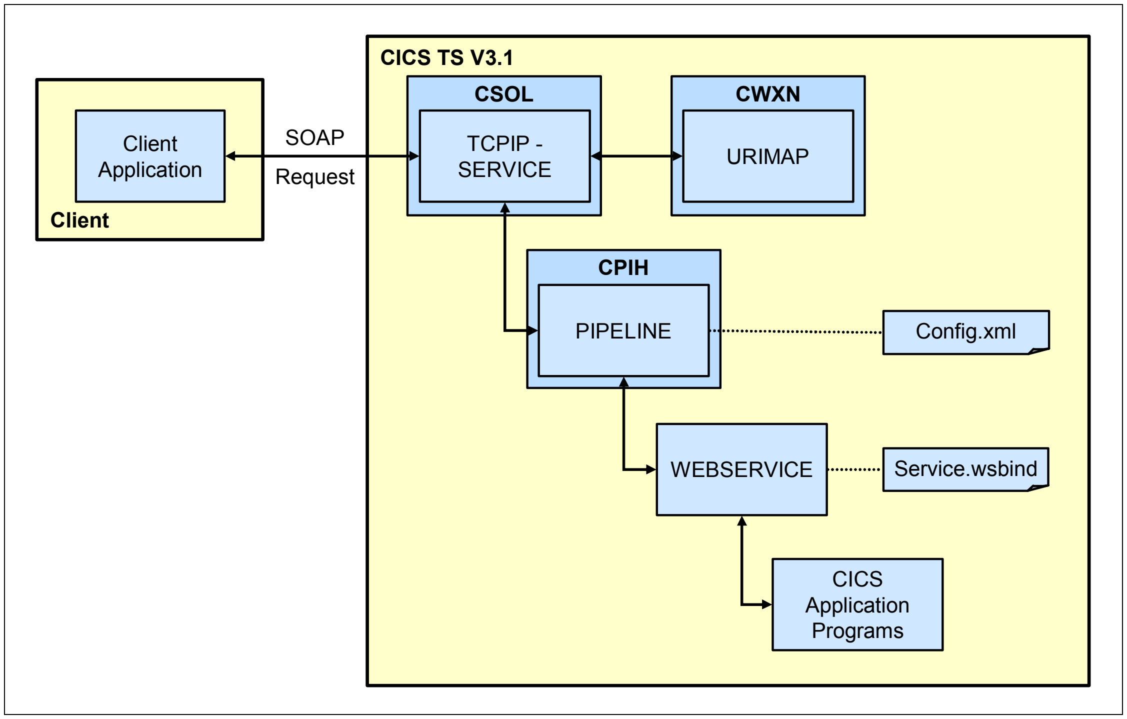


Figure 3-4 Web service runtime service provider processing

The CICS-supplied sockets listener transaction (CSOL) monitors the port specified in the TCPIPSERVICE resource definition for incoming HTTP requests. When the SOAP message arrives, CSOL attaches the transaction specified in the TRANSACTION attribute of the TCPIPSERVICE definition; normally, this will be the CICS-supplied Web attach transaction CWXN.

CWXN finds the URI in the HTTP request and then scans the URIMAP resource definitions for a URIMAP that has its USAGE attribute set to PIPELINE and its PATH attribute set to the URI found in the HTTP request. If CWXN finds such a URIMAP, its uses the PIPELINE and WEBSERVICE attributes of the URIMAP definition to get the name of the PIPELINE and WEBSERVICE definitions that it will use to process the incoming request. CWXN also uses the TRANSACTION attribute of the URIMAP definition to determine the name of the transaction that it should attach to process the pipeline; normally, this will be the CPIH transaction.

CPIH starts the pipeline processing. It uses the PIPELINE definition to find the name of the pipeline configuration file. CPIH uses the pipeline configuration file to determine which message handler programs and SOAP header processing programs to invoke.

A message handler in the pipeline (typically, a CICS-supplied SOAP message handler) removes the SOAP envelope from the inbound request and passes the SOAP body to the data mapper function.

CICS uses the DFHWS-WEBSERVICE container to pass the name of the required WEBSERVICE definition to the data mapper. The data mapper uses the WEBSERVICE definition to locate the main storage control blocks that it needs to map the inbound service request (XML) to a COMMAREA or a container.

The data mapper links to the target service provider application program, providing input in the format that it expects. The application program is not aware that it is being executed as a Web service. The program performs its normal processing, then returns an output COMMAREA or container to the data mapper.

The output data from the CICS application program cannot just be sent back to the pipeline code. The data mapper must first convert the output from the COMMAREA or container format into a SOAP body.

CICS as a service requester

When CICS is a service requester, an application program sends a request, which is passed through a pipeline to a target service provider. The response from the service provider is returned to the application program through the same pipeline. In this section we discuss how to prepare for running a CICS application as a service requester, and how CICS processes the outbound service request.

Preparing to run a CICS application as a service requester

Suppose we want to write a new CICS application that will invoke a Web service. Suppose also that we want to use the Web Services Assistant rather than taking control of the processing ourselves. We go through the following steps listed in Table 3-2.

Table 3-2 Expose CICS asset as a Web service

Step	Action	Role
1.	Generate WSBIND and language structures	Application Developer
2.	Create a PIPELINE resource definition and a PIPELINE configuration file	System Programmer
3.	Install the PIPELINE definiton - WEBSERVICE definition created automatically by CICS	System Programmer
4.	Use the language structure from step 1 to write the client application program	Application Programmer

Processing the outbound service request

Figure 3-5 shows the processing that occurs when a service requester running in a CICS TS V3.1 region sends a SOAP message to a service provider.

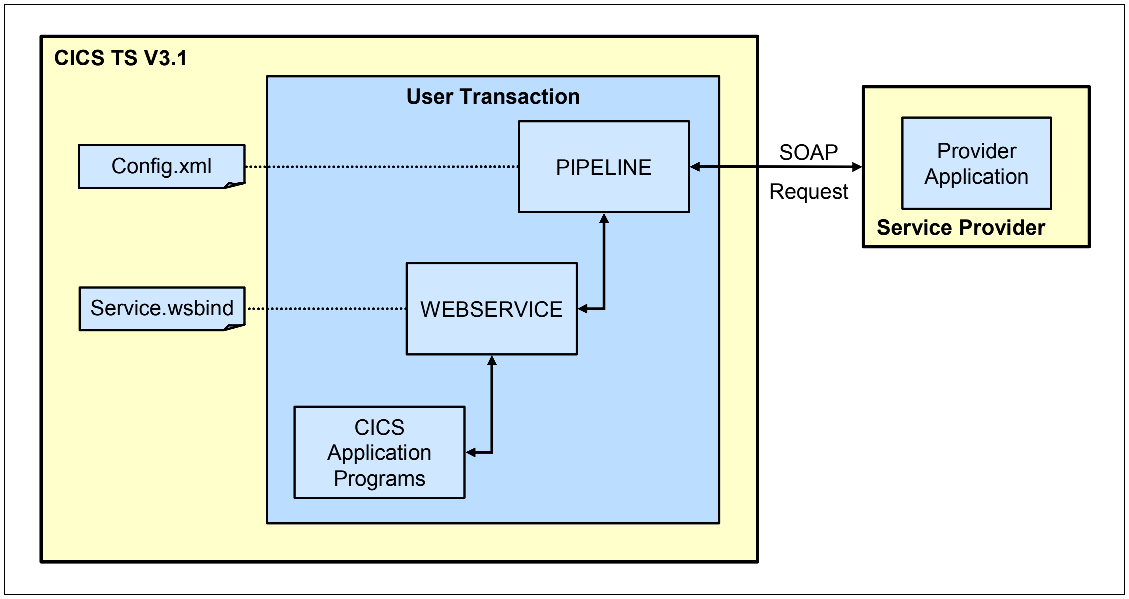


Figure 3-5 Web service requester resources

When the service requester issues the EXEC CICS INVOKE WEBSERVICE command, CICS uses the information found in the WSBIND file that is associated with the specified WEBSERVICE definition to convert the language structure into an XML document. CICS then invokes the message handlers specified in the pipeline configuration file, and they convert the XML document into a SOAP message.

CICS will send the request SOAP message to the remote service provider either via HTTP or WebSphere MQ.

When the response SOAP message is received, CICS will pass it back through the pipeline. The message handlers will extract the SOAP body from the SOAP envelope, and the data mapping function will convert the XML in the SOAP body into a language structure that is passed to the application program in container DFHWS-DATA.

3.1.2 CICS Transaction Gateway

The CICS Transaction Gateway (CICS TG) is a set of client and server software components that allow a remote client application to invoke services in a CICS region. The client application can be either a Java application or a non-Java application using either C, C++, COBOL or COM interfaces (depending on the platform used).

When a Java application is used, then the application can be any type of client (such as a servlet, or an enterprise bean). In the J2EE environment, the application is typically a servlet or enterprise bean that is deployed into a J2EE application server such as WebSphere Application Server.

CICS TG products

With CICS TG V6 and later there are now the following two distinct orderable CICS TG products:

- CICS TG for Multiplatforms
- CICS TG for z/OS

CICS TG for Multiplatforms V6.0

CICS TG for Multiplatforms V6.0 is supported on the following range of operating systems and platforms and is designed to support connectivity to all in-service CICS servers:

- Linux on System z
- Linux on Intel®
- Linux on POWER™
- AIX®
- HP-UX (on PA-RISC)
- Sun™ Solaris™ (on SPARC)
- Windows® XP, Windows 2000, and Windows 2003

This version was announced on 30 November 2004 in a U.S. announcement letter 204-2284, which is available at:

http://www-306.ibm.com/fcgi-bin/common/ssi/ssialias?infotype=an&subtype=ca&supplier=897&letternum=ENUS204-284

CICS TG for Multiplatforms is comprised of the following main runtime components:

- The Gateway daemon, which listens for incoming work and manages the threads and connections necessary to ensure good performance.
- The Client daemon, which provides the communication to CICS servers and the non-Java APIs.
- A Java class library or JCA resource adapter, which is deployed into the client runtime environment. When used in a JCA environment the resource adapter is deployed into the J2EE application server.

A Java client program can connect to a remote Gateway daemon using the TCP or SSL protocols. The Client daemon then provides the transport drivers to connect to the CICS server, as shown in Figure 3-6. Note that because the non-Java APIs are provided by the Client daemon, there is no remote connectivity support for non-Java clients.

Note: The CICS Universal Client V6.0.2, which provides the Client daemon functions within CICS TG, is available as a separate product. CICS Universal Client provides the transport protocols and non-Java programming interfaces for single user access to CICS.

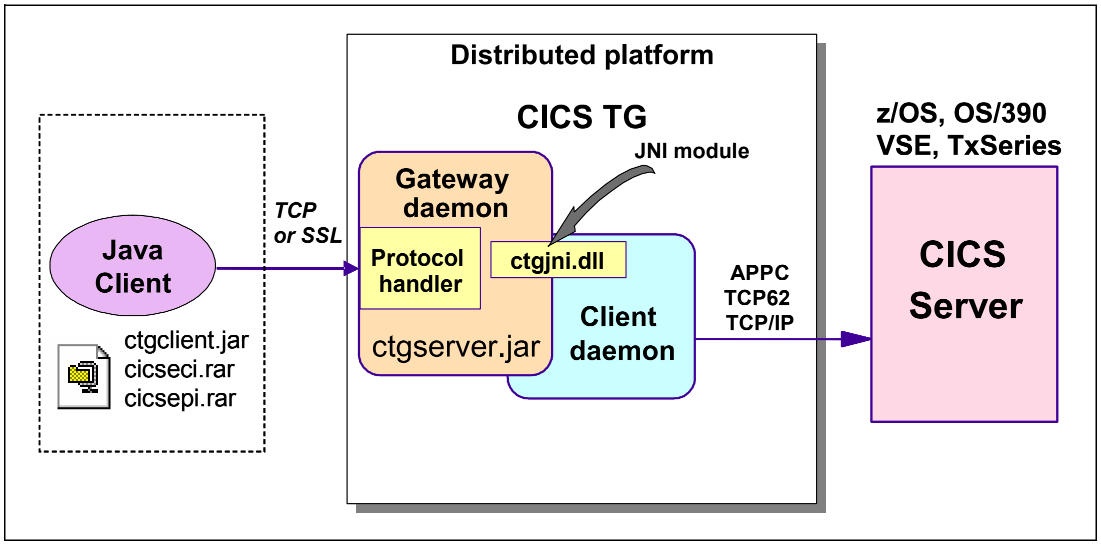


Figure 3-6 Components of CICS TG for Multiplatforms

3.1.3 CICS TG for z/OS V6.1

CICS TG for z/OS V6.1 is the latest version of the z/OS product. It is supported on z/OS V1R4 and later and supports connectivity to CICS TS for z/OS V1.3, V2.2, V2.3, and V3.1.

The product was announced on 04 October 2005 in a U.S. announcement letter 205-248, which is available at:

http://www-306.ibm.com/fcgi-bin/common/ssi/ssialias?infotype=an&subtype=ca&supplier=897&letternum=ENUS205-248

CICS TG for z/OS uses the external communication interface (EXCI) provided by CICS TS to communicate with CICS (Figure 3-7). It does not include the Client daemon and does not provide any support for non-Java based applications because this support is provided through the CICS EXCI interface.

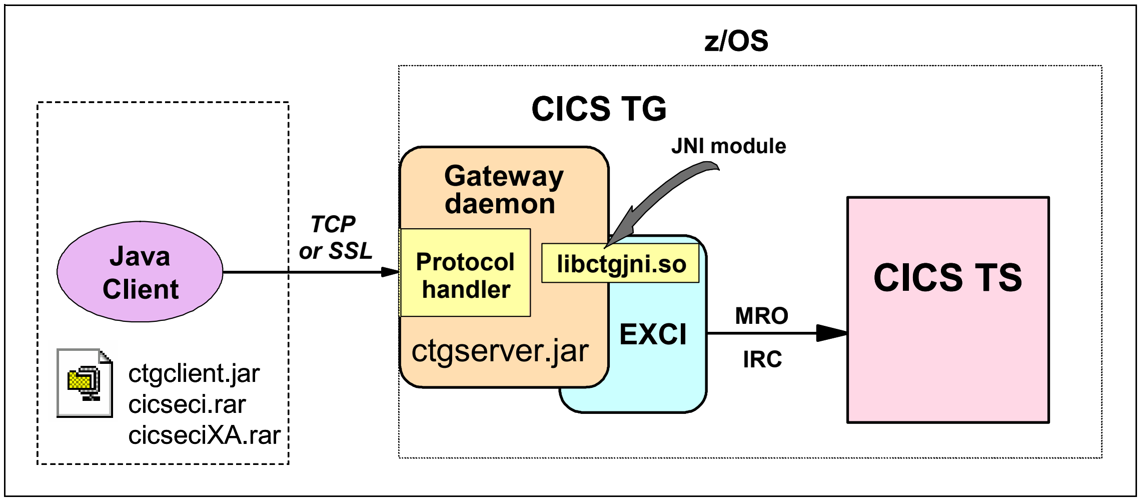


Figure 3-7 Components of CICS TG for z/OS

CICS TG for z/OS V6 has many improvements principally in the area of systems management, usability, and performance as follows:

- ► XA transaction support enables CICS Transaction Server (CICS TS) for z/OS to participate in a global two-phase commit transaction that is initiated in a distributed J2EE V1.4 application server, such as WebSphere Application Server V6. XA transaction support was introduced in CICS TG V6.1.
- ► Management of the Gateway daemon from SDSF provides better system administration capabilities, including a new normal shutdown mechanism.
- ► Direction of output to JES provides improved management for runtime messages.
- ► An option to limit the number of EXCI pipes to one per thread improves reliability and scalability with multiple CICS TORs.

CICS TG application programming interfaces

CICS TG for Multiplatforms provides the following programming interfaces for accessing CICS applications:

- ► External Call Interface (ECI)
- ► External Presentation Interface (EPI)
- ► External Security Interface (ESI)

Note: The CICS TG on z/OS supports only the ECI, because of its reliance on the CICS EXCI function that provides the call interface and connectivity between the Gateway daemon and the CICS TS region.

External Call Interface (ECI)

The ECI is used for calling COMMAREA-based CICS programs. The COMMAREA is the buffer that is used for passing the data between the client and the CICS server. CICS sees the client request as a distributed program link (DPL) request.

The ECI enables a user application to call a CICS program synchronously or asynchronously. It enables the design of new applications to be optimized for client-server operation, with the business logic on the server and the presentation logic on the client.

An ECI request can be invoked from a Java application using a variety of different interfaces:

- The ECIRequest class that is provided by the CICS TG base classes.

  This interface provides a simple procedural type interface to the ECI. It is supported in any Java environment (such as an stand-alone application) and provides similar capabilities to the JCA. However, it does not provide the same qualities of service (such as XA transaction support).

- The Common Client Interface (CCI) that is provided by the CICS ECI resource adapters (cicseci.rar or cicseciXA.rar).

  These classes define a standard architecture for connecting the Java 2 Platform Enterprise Edition (J2EE) platform to a heterogeneous EIS such as CICS. Java applications interact with resource adapters using the Common Client Interface (CCI), which is a common framework of classes extended by each resource adapter to allow communication with a specific EIS.

Note: Before the existence of the JCA, IBM recognized a need for a common way to connect to EIS systems and introduced the Common Connector Framework (CCF) through the IBM VisualAge® for Java product. CCF was similar in concept to JCA. However, CCF was not an open specification and did not provide the same support for system contracts and qualities of service (such as transaction support and connection pooling).

CICS TG V6.1 no longer provides support for the CCF. However, a SupportPac (CE52) is available that provides runtime support for CCF using a CICS TG V6 Java client. It is available at:

`http://www.ibm.com/software/htp/cics/ctg/support`

External Presentation Interface (EPI)

The EPI is used for invoking 3270-based transactions. A terminal is installed in CICS, and CICS sees the request as running on a remote terminal controlled by the CICS TG.

An EPI request can be invoked from a Java application using one of three different interfaces:

- The EPIRequest class provided by the CICS TG base classes.

  This class provides a Java interface to the EPI, and is used for invoking 3270-based transactions. Due to its low-level nature, using it for developing EPI applications requires a strong knowledge of CICS and 3270 data streams.
- The EPI support classes, which provide high-level constructs for handling 3270 data streams.

  A wide range of classes is provided including AID, FieldData, Screen, Terminal, Map and MapData. These are used to represent the interface to a CICS 3270 terminal, and the resulting 3270 response.
- The Common Client Interface (CCI) provided by the CICS EPI resource adapter (cicsepi.rar).

EPI is typically used when it is not possible to separate the presentation logic from the business logic of an application, and allows the reformatting of the user interface, without modification of the CICS application.

External Security Interface (ESI)

The ESI is used for verifying and changing the user ID and password information held in the CICS external security manager (ESM), such as RACF®. It is based on the CICS Password Expiration Management (PEM) function.

ESI calls to CICS can only be made using the ESIRequest class that is provided by the CICS TG base classes. This class provides a Java interface to the ESI, and provides two simple PEM requester functions:

Verify Password	Allows a client application to verify that a password matches the password for a given user ID that is stored by the CICS ESM.
Change Password	Allows a client application to change the password that is held by the CICS ESM for a given user ID.

There is no other interface available for the ESI, although both the EPI and ECI allow user IDs and passwords to be flowed within the actual requests. In this case the user ID and password is authenticated either within CICS, if using a Multiplatform CICS TG, or within the CICS TG, if using the CICS TG for z/OS.

CICS TG for z/OS modes of operation

There are two principle modes of operation for the CICS TG: remote and local.

Remote mode of operation

The *remote mode* of operation uses the Gateway daemon as a long running task which listens on specified ports for incoming ECI requests and then forwards them to the CICS server via the EXCI protocol. The Gateway daemon runs in its own MVS address space and provides connection and thread management.

Local mode of operation

If the CICS TG is to be used on the same machine as WebSphere Application Server, it is more efficient to use the CICS TG classes within WebSphere Application Server to provide the gateway functionality. This mode of operation allows WebSphere Application Server to manage the connections and threads and reduces the communications overhead. This configuration is known as the *local mode* of operation.

CICS TG and the JCA

The CICS TG is the preferred J2EE connector for CICS TS, and in conjunction with IBM WebSphere Application Server provides, a high performing, secure, scalable and tightly integrated access method in CICS.

The JCA system-level contracts between a J2EE application server, such as WebSphere Application Server, and a resource adapter determine the scope of the JCA managed environment. The standard contracts include a connection-management contract, transaction-management contract and a security-management contract. These contracts provide the mechanisms by which the management of connections, security and transactions are performed:

- The connection-management contract enables the application server to pool and re-use connections into CICS, enabling a more scalable and efficient environment that can support a large number of concurrent accesses to a CICS region.
- The transaction-management contract defines the scope of transactional integration between a J2EE application deployed in WebSphere Application Server and a CICS program.
- The security-management contract defines how security context information is passed between the application server and CICS.

Using the CICS ECI resource adapter with different topologies

The JCA system contracts are the key to the qualities of service provided by WebSphere Application Server and the CICS ECI resource adapter. However, the qualities of service vary depending on the topology in use. The three most common topologies shown in Figure 3-8 are:

Topology 1	WebSphere Application Server and the CICS TG are both deployed on a distributed (non-System z) platform.
Topology 2	WebSphere Application Server is deployed on a distributed platform and the CICS TG is deployed on a z/OS system.
Topology 3	Both WebSphere Application Server and the CICS TG are deployed on System z.

These topologies are discussed in more detail further in this section.

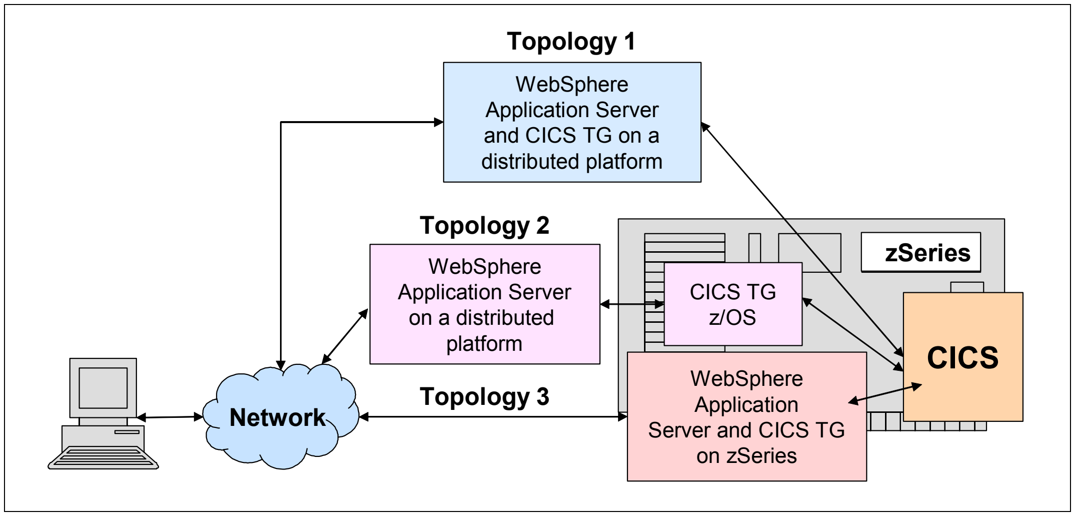

Topology 1
WebSphere Application Server and CICS TG on a distributed platform
Topology 2
WebSphere Application Server on a distributed platform
CICS TG z/OS
zSeries
CICS
Network
Topology 3
WebSphere Application Server and CICS TG on zSeries

Figure 3-8 Common topologies for using CICS TG with WebSphere Application Server

1. WebSphere Application Server and CICS TG deployed on distributed platforms

   In topology 1 (Figure 3-9), both WebSphere Application Server and CICS Transaction Gateway are deployed on one of the distributed platforms, such as a Windows or UNIX® platform. Both ECI and EPI resource adapters can be used in this configuration. Figure 3-9 shows an EJB application using the ECI resource adapter to access a COMMAREA-based CICS COBOL application.

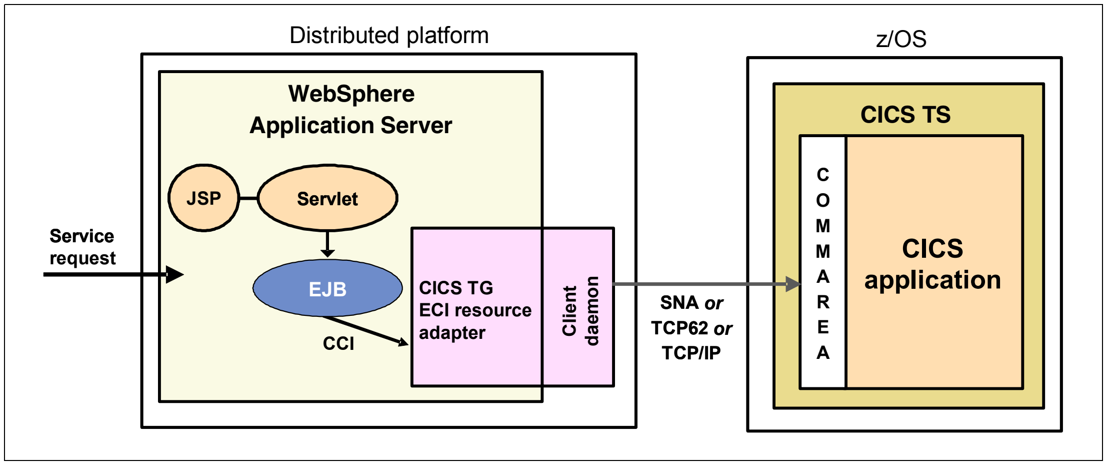


Figure 3-9 CICS TG Topology 1

In this configuration, the Gateway daemon is not required because the CICS TG *local:* protocol is used to invoke the Client daemon directly from the enterprise bean (this protocol parameter is specified as a setting on the connection factory using the WebSphere Administrative Console).

A variation of this topology is where the WebSphere Application Server and CICS TG are installed on one of the distributed platforms but reside on different machines. The main difference in this variation is the connection between the application server and the CICS TG is a physical network connection rather than a local binding.

Using the *local:* protocol is the most efficient option for topology 1 because it avoids a network flow and reduces the systems administration since there is no Gateway daemon to manage.

2. Remote Gateway daemon on z/OS

   In topology 2, where WebSphere Application Server is deployed on one of the distributed platforms, it is possible to access CICS through a Gateway daemon running on z/OS, as shown in Figure 3-10 on page 57.

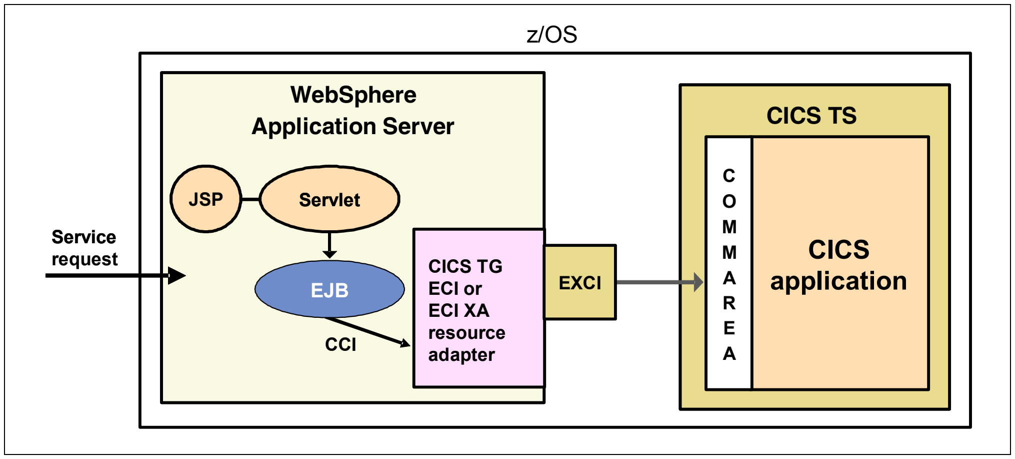


Figure 3-10 CICS TG Topology 2

In this configuration, the protocol used is one of the remote protocols (TCP, HTTP, SSL or HTTPS). The communication from the CICS Transaction Gateway on z/OS to the CICS server utilizes EXCI.

This configuration is widely used today for the following reasons:

- Topology 1 only supports native TCP/IP connections into System z systems for CICS TS V2 onwards whereas topology 2 supports TCP/IP connections into System z for CICS TS V1 onwards.
- When CICS security is enabled, topology 1 requires that a user ID and password are flowed with each ECI request. In some situations, user ID authentication by CICS does not easily fit within the security design of a project. Using topology 2 can help to avoid this problem because CICS TG for z/OS allows a pre-authenticated user ID to be flowed into CICS without a password.
- Topology 2 enables integration with z/OS IP workload-management functions, including Sysplex Distributor and TCP/IP port sharing.
- Deploying the CICS TG on z/OS can leverage the existing CICS systems-management skills within the enterprise.
- Topology 2 provides XA transaction support, to enable resources to participate in two-phase commit transactions. The two-phase commit capability for this topology is provided through MVS resource recovery services (RRS), an integral part of z/OS. RRS acts as the external transaction coordinator for z/OS in managing the transaction scope between WebSphere Application Server, CICS and other z/OS subsystems, including IMS/TM, IMS/DB, IBM DB2 and WebSphere MQ systems.

3. WebSphere Application Server and CICS TG deployed on System z

   In a System z topology, WebSphere Application Server can be deployed on either a z/OS system or on a Linux operating system. The qualities of service which will be discussed in subsequent chapters of this book differ significantly between these two topologies.

   a. WebSphere Application Server and the CICS TG on z/OS (Figure 3-11)

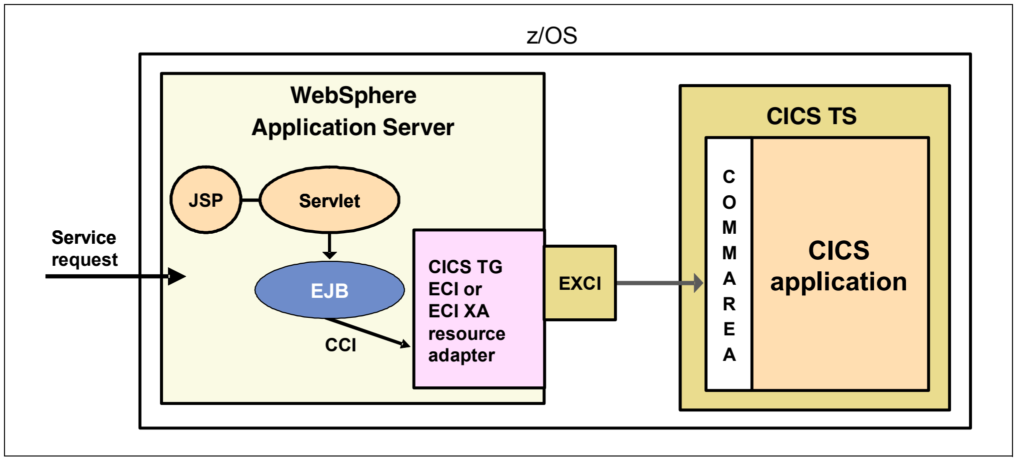


Figure 3-11 CICS TG Topology 3a

Topology 3a supports the CICS ECI resource adapter but not the EPI resource adapter. As in topology 1, the most common z/OS configuration makes use of a local CICS TG. On z/OS, this results in a direct cross-memory EXCI connection between the application server and CICS. Figure 3-11 shows the application deployed to WebSphere Application Server for z/OS using a local CICS TG.

CICS TG also supports remote connections for this topology which allows a connection from the application server to a z/OS Gateway daemon. This functionality was introduced primarily for z/OS.e customers. z/OS.e is a specially priced offering of z/OS that provides select z/OS function for the System z 800 system. Traditional workloads, such as CICS TS, are not licensed for z/OS.e. Because the CICS TG relies on the EXCI interface provided by CICS TS for z/OS, the CICS TG must be installed in a full-function z/OS LPAR. This means a remote connection must be used from WebSphere Application Server into the CICS TG.

The highest qualities of service, however, can be achieved when a local connection between the application server and the CICS region is used. In particular, this topology is the only topology that provides thread identity support.

The unique thread identity support in WebSphere Application Server for z/OS allows the application server to automatically pass the user ID of the thread (for example, the caller's identity) to CICS when using the ECI resource adapter. This satisfies a common end-to-end security requirement of automatically propagating the authenticated caller's user ID from WebSphere Application Server to CICS.

b. WebSphere Application Server and CICS TG on Linux on System z

Topology 3b (see Figure 3-12) is where WebSphere Application Server is deployed within Linux on System z.

The JCA qualities of service for this configuration are almost identical to those described for topology 1 because Linux on System z (within a JCA and CICS TG scenario) can be treated as a distributed platform. A significant exception to this generalization is that HiperSockets™ can be utilized to provide a highly efficient cross-memory transport for TCP/IP-based communication into CICS (using the ECI over TCP/IP function of CICS TS V2.2 and higher). In addition, the APPC protocol is currently not supported when deploying the CICS TG on Linux. As a result, the TCP62 protocol must be used if support is required for EPI requests.

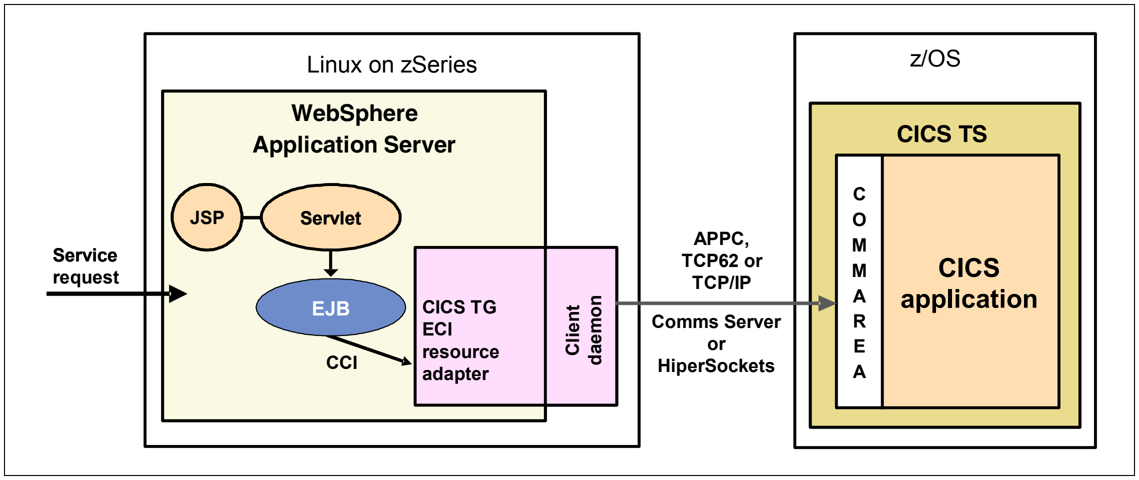


Figure 3-12 CICS TG Topology 3b

The same choice of where to deploy the CICS TG exists for implementations of WebSphere Application Server on Linux on System z as it does for implementations of WebSphere Application Server on distributed platforms. CICS TG can be deployed within the same Linux partition as WebSphere Application Server (in which case a local connection from WebSphere Application Server is used) or on z/OS as in topology 2 (in which case a remote connection is used).

The CICS TG topologies are referred to in subsequent chapters, in particular, in terms of how they determine the qualities of service offered by the JCA system contracts.

3.1.4 CICS Web support

CICS Web support is a set of resources supplied with CICS TS for z/OS that provide CICS with a subset of the HTTP serving functions found in a general-purpose Web server. This allows CICS applications to be invoked by and reply to HTTP requests. A summary of how a CICS application can be Web-enabled using the CICS Web support is illustrated in Figure 3-13.

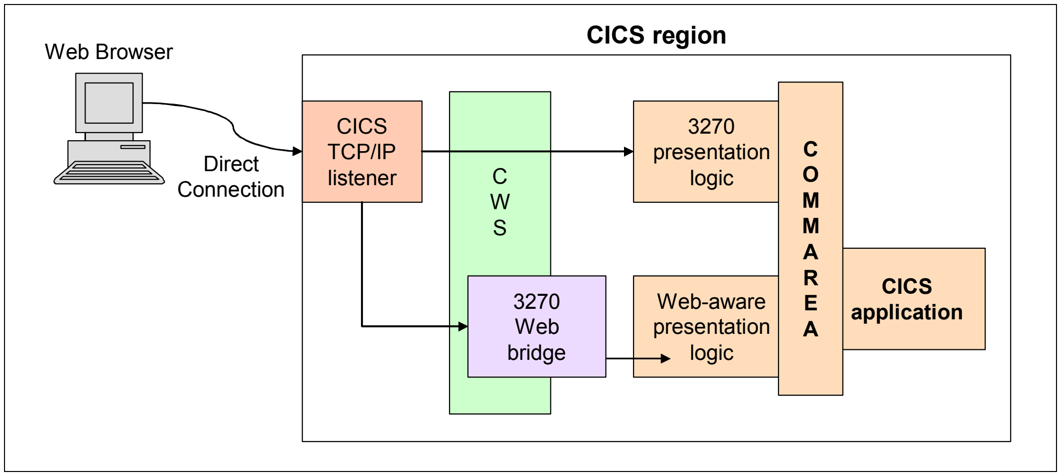


Figure 3-13 CICS Web support overview

CICS Web support provides a native HTTP interface to CICS; this interface can be used by both 3270-based transactions and applications that provide a callable COMMAREA interface. Two different configurations can be used to route the HTTP requests into the CICS region. Both configurations allow the use of the same facilities in CICS, although the configuration of the two options is significantly different. These configurations are as follows:

- A *direct connection* from a Web browser to CICS. This uses the facilities of the CICS TCP/IP listener to pass the request directly into CICS Web support.
- Through the IBM HTTP Server using the facilities of the *CICS WebServer Plugin*. This is a CICS-supplied Go Webserver API (GWAPI) extension to the IBM HTTP Server. It routes requests from the HTTP Server into the CICS region using the EXCI. CICS supplies two GWAPI modules called DFHWBAPI and DFHWBDLL. It you use the CICS WebServer plugin, you are recommended to use DFHWBDLL because it has support for the FORMS API commands (for example, **`EXEC CICS WEB READ FORMFIELD`**).

  We will not be discussing the CICS WebServer Plugin any further in this book since it is no longer a widely used configuration.

CICS Web support can be used to invoke two types of CICS applications:

1. To invoke a *3270 transaction*, the facilities of the 3270 bridge are used. The 3270 transaction remains unchanged, and the 3270 output is converted to HTML. This function is known as the *3270 Web bridge*.
2. To invoke an existing application that provides a callable COMMAREA interface, some new CICS presentation logic must be written. This logic uses CICS facilities to interpret and act upon the HTTP request, and then build and return the HTTP response. We will refer to application code containing such logic as *Web-aware*. This Web-aware logic can be contained either within the CICS Web support converter Encode and Decode routines, within the original program, or in a separate Web presentation module that links to the original program. CICS provides two different methods to create this Web-aware presentation logic:

   – WEB API

     The WEB API, together with the DOCUMENT API and TCP/IP API, provide a rich set of functions to interpret, manipulate, and build the HTTP data streams within a CICS application. They are described in more detail in the *CICS Application Programming Guide,* SC34-6231, *CICS Application Programming Reference,* SC34-6232, and Chapter 2 in *CICS Transaction Server for OS/390 Version 1, Release 3: Web Support and 3270 Bridge*, SG24-5480.

– COMMAREA manipulation

The COMMAREA manipulation technique was originally introduced with CICS Web Interface support in CICS/ESA® V4.1. It uses the CICS COMMAREA as a buffer for transferring the HTTP data stream along with a range of utility programs to manipulate the data stream. The CICS Web support HTML template manager program (DFHWBTL) can be used to build the response. This technique is still available, but for ease of use and higher functionality, we recommend the use of the WEB API.

CICS Web support: Direct connection

Figure 3-14 illustrates the major components of CICS Web support when using Web-aware presentation logic via a direct connection to CICS. The light shading is for CICS Web support components that run under the Web attach transaction; the darker shading is for CICS Web support components that run under the alias transaction.

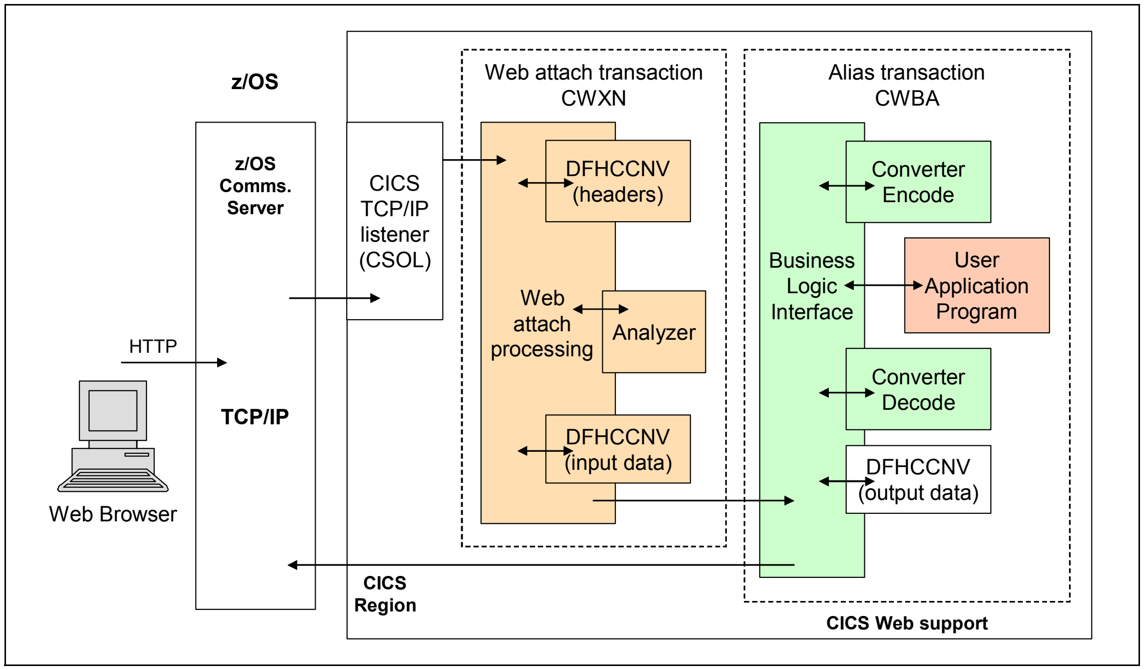


Figure 3-14 CICS Web support

Using the CICS Web support direct connection and passing data in the COMMAREA allows greater than 32 KB of data to be returned from the application to the Web browser, although only 32 KB of data can be received by the application. If the application uses the **`WEB`** and **`DOCUMENT API`** commands then it can both send and receive greater than 32 KB.

CICS TCP/IP listener

The CICS TCP/IP listener is part of the CICS Sockets domain, and runs as the CSOL (Sockets listener) system task. It provides TCP/IP support to handle requests for internal CICS functions that use TCP/IP services, currently HTTP, IIOP, and ECI over TCP/IP support. As such, it is not a component of CICS Web support, but a service used by CICS Web support.

> **Note**: The CICS TCP/IP listener is completely separate from, and not to be confused with, the TCP/IP Socket Interface for CICS which provides an application level TCP/IP socket interface to CICS applications, and is described further in 3.2.5, "CICS sockets" on page 73.

Web attach transaction (CWXN)

The Web attach transaction (CWXN) performs the Web attach processing. It invokes the DFHCCNV code page conversion routine to convert HTTP headers, links to the analyzer, invokes DFHCCNV again to convert user data, and then starts the alias. The CWXN task will terminate after invoking the alias, unless persistent HTTP connections are used. The CWXN task also stores the data it received from the client in temporary storage (TS) queues for later use by the **`WEB API`** commands.

DFHCCNV

The DFHCCNV code page conversion routines are invoked by the Web attach processing to convert the ASCII HTTP headers and user data of the Web browser client to EBCDIC, and by the alias transaction to convert EBCDIC output back to ASCII. This conversion only affects the data in the COMMAREA. It does not affect the data in the TS queues that will be used by the **`WEB API`** commands.

Analyzer

The purpose of the analyzer is to analyze the incoming HTTP request. It runs under the CWXN transaction, and decides if the request will be accepted, and if so, what parameters will be set. Among other things, it can set the name of the alias, converter, user ID, and user program.

This has now been superseded by the URIMAP definition, which allows URIs to be mapped to programs using CICS RDO.

Alias

The alias transaction is invoked by the Web attach processing. The default alias transaction code is CWBA, but this can be modified. The alias initially invokes the program DFHWBA, which links to the business logic interface.

Business logic interface

The business logic interface (BLI) is an externally callable interface to CICS Web support. It is implemented by the module DFHWBBLI. It provides a mechanism for implementing Web-aware presentation logic in the *converter*. The converter provides Decode and Encode routines to receive the HTTP request and construct a COMMAREA for the target application and to take the COMMAREA from the application and send out an HTTP response. Note that it is possible to bypass the converter and implement the Web-aware logic in a separate module which would communicate directly with the business logic through a COMMAREA interface. The business logic interface and the converter, when present, provide the function which we call the *message adapter*.

3.2 Other solutions

Because there are so many combinations of browsers, servers, network connections, CICS applications, and user environments, there may be a requirement for access to CICS that none of the previously described technologies meets completely. There are a number of other options which you may also want to consider in these situations. We describe some of these options in this section.

- WebSphere MQ
- CICS Enterprise Java Bean support
- SOAP for CICS feature
- WebSphere Host Access Transformation Services
- CICS Sockets

3.2.1 WebSphere MQ

WebSphere MQ is a family of products available on all major operating system platforms. It provides an open, scalable, industrial-strength messaging and information infrastructure enabling enterprises to integrate business processes.

WebSphere MQ provides Java Message Service (JMS) APIs and native WebSphere MQ APIs for use by clients on a wide variety of platforms. This makes it suitable for communication between a range of different types of application and CICS applications.

Figure 3-15 on page 65 shows a typical scenario for access of a CICS application using WebSphere MQ.

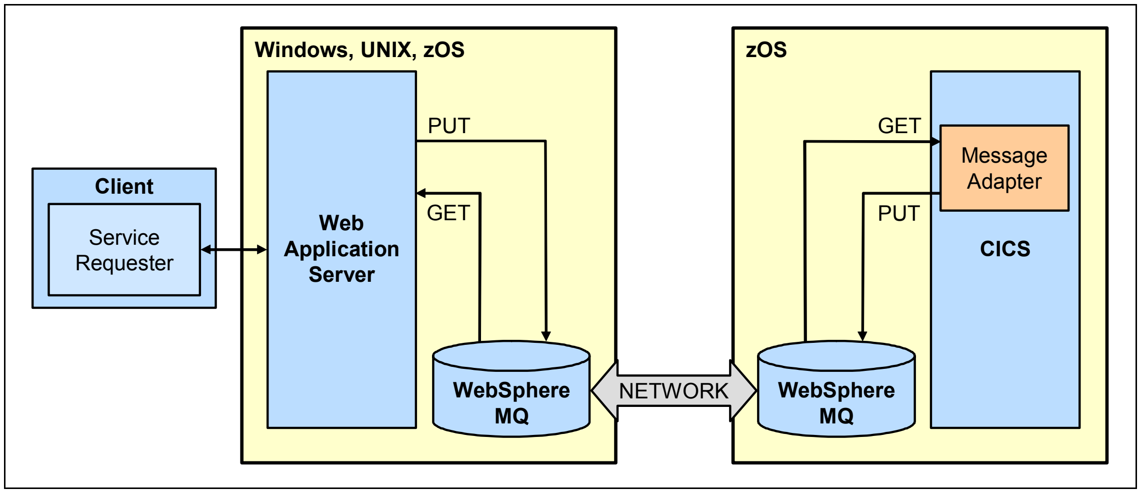


Figure 3-15 Using WebSphere MQ as a CICS access component

A message adapter program, which uses the WebSphere MQ native APIs, is required to receive the message, transform it if required and call the business logic program in CICS.

Another approach which avoids the need to write a message adapter program yourself, is to use the WebSphere MQ DPL bridge.

The WebSphere MQ DPL bridge provides a CICS transaction that can monitor a particular message queue across the connection between the CICS server and a queue manager. A client can use one of the WebSphere MQ APIs to write a structured message to this queue. This message must contain information in a predefined format that the monitoring transaction can use to decide how to handle the message.

Several message formats are possible, each starting with a block of data called an MQMD header. The MQMD contains control information used by the monitoring transaction like the message format type, along with optional information, such as a reply-queue identifier and a user ID. The information that follows the MQMD field can simply be the name of the application program to run within CICS, with the option to include data that the program can receive as its COMMAREA.

Figure 3-16 on page 66 shows the main components of this architecture.

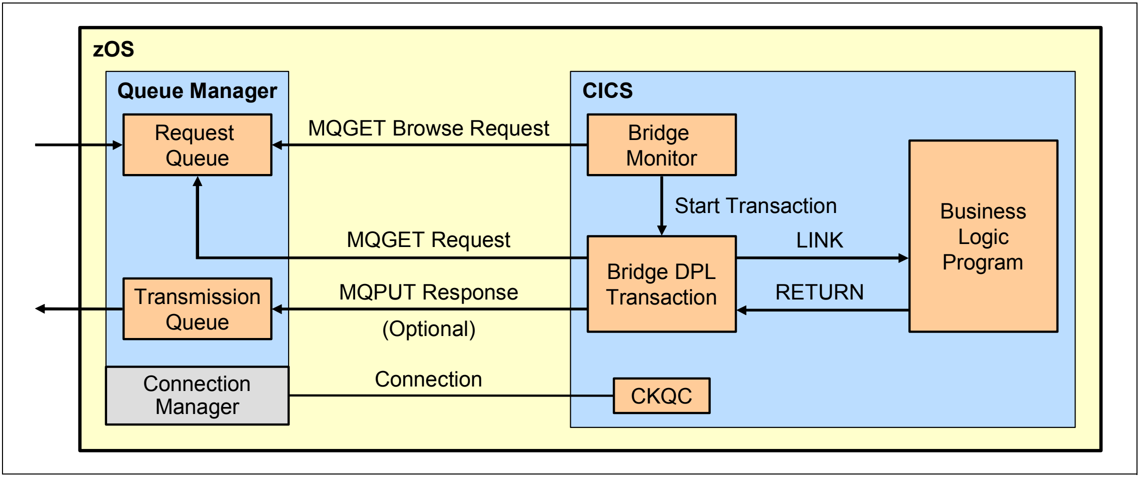


Figure 3-16 WebSphere MQ-CICS bridge for DPL requests

A connection must exist between the queue manager and CICS region, both running in the same MVS system. You can achieve this connection using the adapter, under the control of the CKQC transaction.

You must start a bridge-monitoring transaction (CKBR) to look for messages arriving on a particular queue. Several monitoring transactions can run concurrently to manage a set of queues in this way. When the message arrives, the monitoring transaction detects it and starts a bridge DPL transaction to process the message. The monitoring transaction continues to look for other messages arriving on the queue. You can set up the bridge DPL transaction to run requests using a system user ID or the user ID of the requester.

The DPL bridge transaction reads the message from the queue. From within the message, the task finds the name of a CICS application program, any input data it requires and, optionally, the name of a message queue to send a response to. Next, the DPL bridge transaction sets up a COMMAREA containing the input data and links to the named program, returning any output COMMAREA to the reply queue after the program has ended. The client application can use this mechanism either synchronously or asynchronously. It can wait for a response by monitoring a particular queue, or it may send a message and continue processing other transactions or even terminate without waiting for CICS to process the request. If the client runs on the same system as the CICS server, it can monitor the transmission queue for a response. If it is remote to the CICS server, the client can use WebSphere MQ to route the response message from the transmission queue to one defined to its local queue manager.

Each DPL request executes in isolation, and no state is preserved in CICS to tie up a series of requests that a client might make. So, if the client calls CICS more than once, each piece of work is treated separately because the server doesn't retain any information to tie them together. The WebSphere MQ monitoring transaction and DPL bridge transaction run within the same CICS region. The program targeted by the DPL request can be made eligible for routing to another CICS region. All to help balance the workload more effectively.

By using the Link3270 bridge with the WebSphere MQ DPL bridge you can also enable access to a CICS terminal-oriented program.

For further information about WebSphere MQ refer to the following publications:

- *WebSphere MQ for z/OS V5.3 Concepts and Planning Guide*, GC34-6051
- *WebSphere MQ for z/OS V5.3 System Setup Guide*, SC34-6052
- *WebSphere MQ for z/OS V5.3 System Administration Guide*, SC34-6053
- *WebSphere MQ Application Programming Guide*, SC34-6064

3.2.2 CICS Enterprise JavaBeans support

JavaBeans and Enterprise JavaBeans are component architectures for the Java language.

A JavaBean is a self-contained, reusable software component, written in Java, usually intended for use in a desktop or client application. Typically, desktop JavaBeans have a visual element, and execute within some type of visual container, such as a form, panel, or Web page.

The Enterprise JavaBeans architecture supports server components. Server components are application components that run in an application server such as WebSphere Application Server and CICS. Unlike desktop components, they do not have a visual element and the container they run in is not visual. Server components written to the Enterprise JavaBeans specification are known as enterprise beans. They are portable across any EJB-compliant application server.

CICS provides support for session beans which are instantiated by a client and represent a single conversation with the client. In most cases, this conversation only lasts as long as the client is active. From this point of view, the session bean is very similar to a pseudo-conversational transaction in CICS.

Note: CICS TS V3.1 supports session beans at the Sun Microsystems™' Enterprise JavaBeans Specification, V1.1 level but does not have support for other types of Enterprise JavaBeans such as entity beans and message-driven beans.

To be useful, server components require access to the application server's infrastructure services, such as its distributed communication service, naming and directory services, transaction and security management services, data access and persistence services, and resource-sharing services. Different application servers implement these infrastructure services using different technologies. However, an EJB-compliant application server provides an enterprise bean with access to these services through standard interfaces, and manages many of them on behalf of the bean.

In the EJB architecture a Java client is able to invoke methods on the Enterprise JavaBean across a network. The Java client can invoke the bean once it obtains a reference to it using a JNDI (Java Naming and Directory Service Interface) call to lookup the bean's home interface. The Java Object Request Broker (ORB) acts as an intermediary between the client and the server by transparently marshalling and unmarshalling parameters.

Figure 3-17 on page 69 shows how CICS EJB support can be used to enable SOA access to a CICS business logic program. When an EJB request is received the ORB decodes the Remote Method Invocation (RMI). The ORB in turn calls the appropriate methods of the Enterprise JavaBean (a session bean) to process the request. The session bean can call a business logic program either using the CCI API (a CCI Connector is provided by CICS TS) or the JCICS classes. The CCI is recommended because it is a non-CICS specific interface and therefore enhances the portability of the session bean component. If the message needs to be transformed, a message adapter could be called prior to calling the business logic.

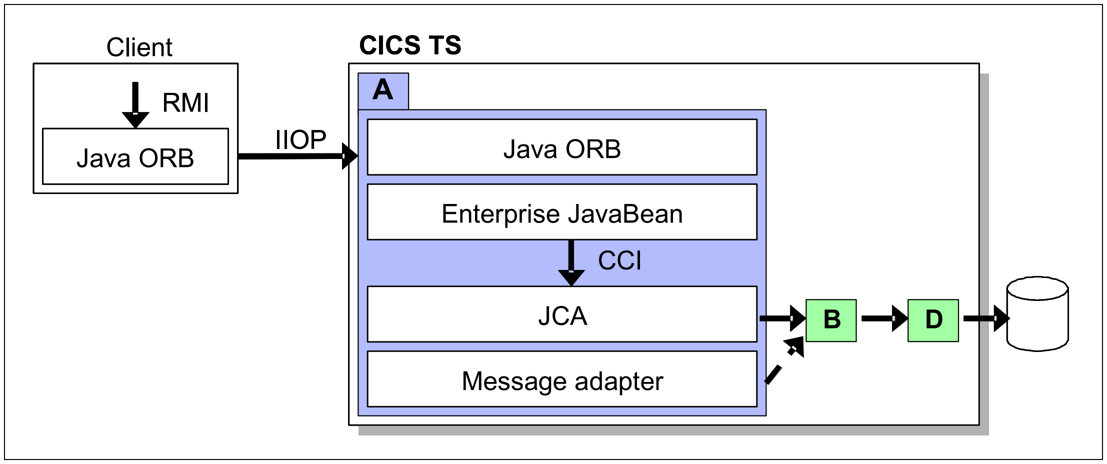


Figure 3-17 Access to CICS using EJBs

The session bean in CICS has the option to be stateless between invocations or stateful, in which case CICS will save the session bean data into a VSAM file and restore the state automatically.

The EJB support provided by CICS supports SSL encryption and full transaction coordination using the two-phase commit protocol. When used with WebSphere Application Server for z/OS, the EJB support in CICS enables automatic propagation of security credentials from the application server to CICS. This functionality is known as *asserted identity support*.

The EJB architecture is considered a tightly coupled connection because both ends need to be implemented by compatible J2EE technologies and EJB interfaces.

For more information see Appendix A, "CICS EJB support" on page 307.

3.2.3 SOAP for CICS feature

Note: In CICS TS V3.1 this feature has been superseded by the base CICS Web services support.

The SOAP for CICS feature allows CICS to operate as a service requester or service provider within a Web services architecture.

Using the SOAP for CICS feature CICS business logic programs can interact securely and reliably with other Web services, exchanging XML based messages, independently of platform, or application language. Within the Service Orientated Architecture (SOA), standard definitions are used to describe the message formats than can be sent. Thus developers can rapidly build open standards based applications independently of the CICS business logic program they will interact with. This is the essence of the service orientated architecture, and the reason that is deemed to be loosely coupled.

The SOAP for CICS functionality was originally introduced as supportpac CA1M which has now been replaced by an optional feature for CICS Transaction Server V2.2 or V2.3.

The service provider and service requester functions within CICS are run as *pipelines* utilizing CICS Business Transaction Services (BTS) functionality. User application code running as part of the pipeline needs to be aware that it will be within a BTS activity so that it correctly handles events and containers.

SOAP messages are sent as XML documents, for these messages to be understood by an existing CICS application, they must be parsed, and so user application code will need to be added to the pipeline to parse the XML. Enterprise COBOL V3 and Enterprise PL/I V3 both provide built-in XML parsing functionality for the PL/I and COBOL languages, and this can simplify the development of parsing code. Alternatively, parsing code can be generated automatically using a tool such as WebSphere Developer for System z.

For more information see Appendix B, “SOAP for CICS feature” on page 383.

3.2.4 WebSphere Host Access Transformation Services

IBM WebSphere Host Access Transformation Services (HATS) is a toolkit, or plug-in, for the Rational Software Development Platform that enables customers to provide access to 3270 or 5250 host applications using a Web browser and WebSphere Application Server. Using the Java code that is deployed by the toolkit, host screens are dynamically converted to a point-and-click Web interface and delivered as HTML to the Web browser. This conversion takes place in real time as the end user interacts with the host application.

The HATS toolkit provides a global default transformation that can convert all screens of a host application to a graphical user interface (GUI), so there is no requirement for host screens to be customized.

HATS also has the capability to provide for fully customized host screens. This is the great value to customers; HATS can convert all screens of a host application to GUIs with little or no effort required while also allowing any screen to be individually customized to meet the customer's requirements. This permits the customization effort to be focused on the small percentage of screens that require enhancement while providing for later phased transition of remaining screens.

Beyond simply customizing individual screens, HATS provides tools to make access to host applications easier and more effective. HATS functions allow customers to skip screens, implement shortcuts, combine screens, enter data on behalf of the end user, store data in global variables, create tabbed folders, generate bar and line graphs, redirect to other URLs, and add new business logic to a host application. Therefore HATS can transform not only the look and feel of an earlier application, but provides the ability to redesign the user access path. Most of these improvements can be accomplished without any programming.

HATS merges the function provided by Host Publisher into a single, intuitive, and powerful environment that retains the strengths of both products. HATS provides a comprehensive solution that allows transactions with host systems to be encapsulated into reusable integrated business services, such as Web services, Java Beans, and Enterprise Java Beans. HATS can be used to create Web pages that invoke the HATS integrated business services.

HATS has two components:

- HATS Studio, running on a client. Using HATS Studio, the application developer creates a Web application in which host screens are transformed into Web pages. The developer can start by building a simple HATS project and then customize it iteratively. HATS Studio is installed in the same directory path as Rational Software Development Platform and appears as a perspective in the workbench window.
- The HATS runtime code, running inside one or more HATS applications deployed on WebSphere Application Server and sending data back and forth between the user and an application on the host. End users interact with the HATS application through their Web browsers. After you have developed a HATS application, you deploy it on the WebSphere Application Server the same way you deploy applications developed using other WebSphere tools. A WebSphere system administrator can monitor HATS applications using the WebSphere Administrative Console.

Figure 3-18 on page 72 shows the HATS application developed in HATS Studio, deployed to WebSphere Application Server, and used by an end user to access a host application.

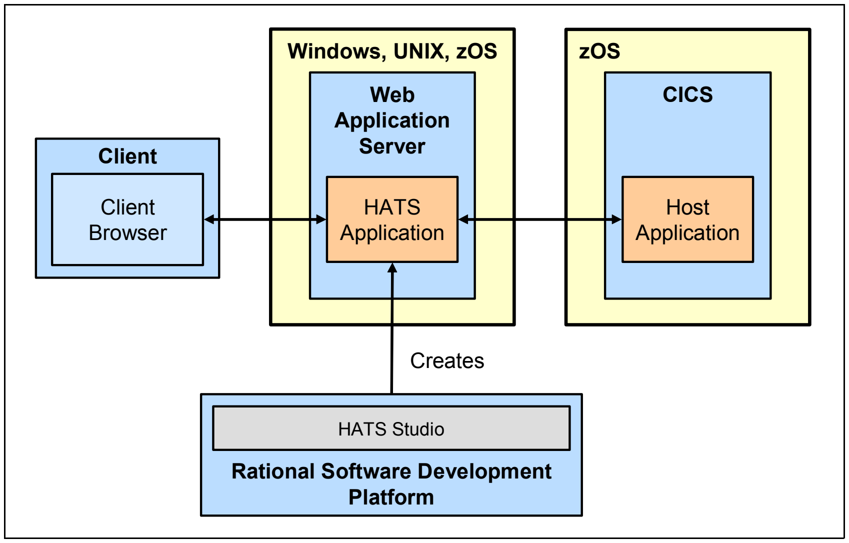


Figure 3-18 HATS application creation and deployment

For further details on Host Access Transformation Services, refer to Redpapers *HATS Concepts and Architecture*, REDP-3706 and *Creating and Modifying HATS Projects*, REDP-3698, and the redbook *Using WebSphere Host Access transformation Services V5*, SG24-6099.

3.2.5 CICS sockets

The TCP/IP Socket Interface for CICS (also known as CICS sockets) is a feature of z/OS Communications Server that brings the TCP/IP sockets API to your CICS applications. It is illustrated in Figure 3-19.

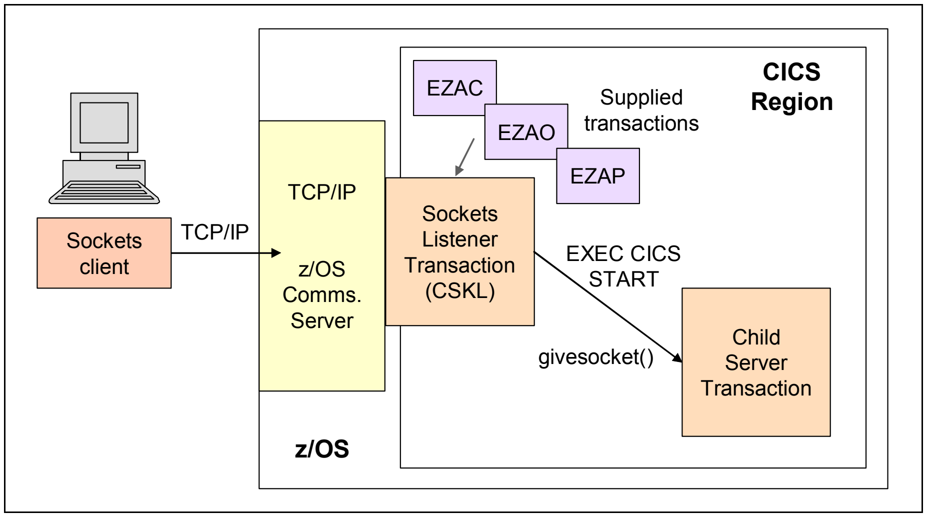


Figure 3-19 CICS to TCP/IP Sockets Interface

The main function of the CICS to TCP/IP Sockets Interface is provided by the Sockets Listener transaction (CSKL). This is a long running task that listens for incoming TCP connection requests on a specified port. The provided EZAC transaction can be used to configure the Sockets Listener, and in addition the EZAO and EZAP transactions can be used for operational requests. It uses a CICS Task Related User Exit (TRUE) to enable the use of native socket functions which would otherwise cause CICS to be suspended while waiting for outstanding socket requests.

Once CSKL receives a TCP connection request, it performs an **`EXEC CICS START`** command for a child server transaction, and passes control of the socket conversation using the `givesocket()` call. The child server transaction is the user-written socket application, and must retrieve the socket data using an EXEC CICS RETRIEVE and take control using a `takesocket()` call. This user application can be written in any language supported by CICS, including C, COBOL, Assembler or PL/I. The design of the client/server communication is entirely up to the user application, which is responsible for all design issues including authentication and data conversion issues.

Note: The child server transaction is also capable of initiating outbound TCP connections as well as receiving incoming requests.

For further details, refer to the Communications Server manual, *IP CICS Sockets Guide,* SC31-8518, and to the redbook *CICS/ESA and TCP/IP for MVS Sockets Interface,* GG24-4026.

3.3 CICS solution set table

Table 3-3 on page 75 is intended as a starting point in selecting a solution for access to your CICS applications. Proceed through the columns in the table from left to right.

1. The first column in the table (application interface) reflects the factor that will influence your decision most fundamentally.
2. After selecting the application interface that applies to your environment, choose an appropriate connection architecture from the second column. Note if just a standard transport is utilized column two will not indicate a connection architecture.
3. Column three indicates the transport that implements the selected connection architecture.
4. Lastly, column four contains the CICS technology solution set options that apply to your environment.

Table 3-3 CICS solution set table

Application interface	**Connection architecture**	**Transport (connector or protocol)**	**CICS solution set**
COMMAREA	JCA	CICS TG	CICS ECI resource adapter
	-	HTTP	CICS Web Support (Web-aware[a] application)
	Web services	HTTP	CICS Web support and CICS Web services
	Web services and messaging	WebSphere MQ	WebSphere MQ and CICS Web services
	messaging	WebSphere MQ	WebSphere MQ CICS DPL bridge
3270	JCA	ECI	CICS ECI resource adapter and Link3270 bridge[b]
	JCA	EPI	CICS EPI resource adapter (not supported with CICS TG on z/OS)
	-	HTTP	CICS Web support (Web-aware application) and Link3270 bridge
	Web services	HTTP	CICS Web support, CICS Web services, and Link3270 bridge
	Web services and messaging	WebSphere MQ	WebSphere MQ, CICS Web services, and Link3270 bridge
	Messaging	WebSphere MQ	WebSphere MQ CICS DPL bridge and Link3270 bridge
	-	Telnet	WebSphere Host Integration: HATS
EJB (session bean)	EJB	RMI over IIOP	CICS EJB support

a. A Web-aware application is a CICS application that has been written to use the CICS command-level API to interact directly with a HTTP client.
b. Any solution using Link3270 bridge may also use an internal adapter to format and process bridge vectors.

4

CICS application access technologies

Prior chapters presented architectures and technologies used to access CICS regions from external clients and platforms. These included Web services and JCA architectures, and CICS TG, CICS Web Support, and CICS Web services technologies. These technologies are especially useful for accessing linkable application business logic, otherwise known as COMMAREA applications.

However, once inside CICS, another layer of technology is required for accessing terminal-oriented applications, and aggregating terminal-oriented interactions into business process flows, or creating business process flows by aggregating both terminal-oriented applications and COMMAREA applications.

This chapter explores the CICS Service Flow Feature and Link3270 bridge strategic technologies.

4.1 CICS Service Flow Feature

The CICS Service Flow Feature is a composition of the WebSphere Developer Service Flow Modeler (SFM) and the CICS Service Flow Runtime (SFR). CICS SFR and SFM are used to provide business process integration inside CICS.

It is important to note that SFR uses the Link3270 bridge technology to access terminal-oriented applications that use CICS Basic Mapping Support (BMS).

4.1.1 Wrapping process integration in CICS Web service requests

The WebSphere Developer V6.0 Service Flow Modeler (SFM) tooling and the CICS TS V3.1 Service Flow Runtime (SFR) together enable distributed applications to make business requests of existing CICS 3270 and COMMAREA applications as callable services. They enable customer-written J2EE applications to integrate seamlessly with business-critical 3270 and COMMAREA applications.

4.1.2 CICS Service Flow Runtime

Vast amounts of business transactions occur every day through the massive installed base of CICS applications and data. CICS SFR can be used to capture and redeploy this business information in modern mixed-workload environments, involving packaged applications written by independent software vendors and J2EE applications that are based on the popular IBM WebSphere Application Server. In addition, CICS SFR can be used to integrate existing CICS business value into the service-oriented architectures (SOAs) that are used to speed business process integration and to enable better responsiveness to changing business environments and emerging business opportunities.

Much of this CICS business value is currently accessed via 3270-based applications. However, 3270 applications are designed for human-to-computer interaction and have made program-to-program integration difficult. CICS SFR is a strategic solution that is used to avoid being forced into programming-intensive solutions that are prone to error, especially when the host CICS applications are changed for maintenance or upgrade.

CICS SFR supports two modes of operation:

- Passthrough mode

  In pass-through mode, 3270 screens are passed directly to the distributed application. That application can then manipulate the data and send responses to the CICS system as required. Passthrough is intended as a migration aid for customers who have already implemented distributed screen scraping solutions. Passthrough mode does not require tooling and consequently, unlike the tooled requests, 3270 screen navigation must be performed in the client.

- Tooled mode

  In tooled mode, CICS SFR uses SFM-generated adapter services, or navigators, to provide the sequencing of 3270 screens in the CICS application. Thus, when a service request comes to CICS from a distributed application, CICS SFR navigates the appropriate 3270 screen sequences, formulates a consolidated response, and sends a single service response to the requester. Flow sequences for Link3270 Bridge, FEPI, DPL (COMMAREA), and WebSphere MQ may be aggregated in a single service request as shown in Figure 4-1.

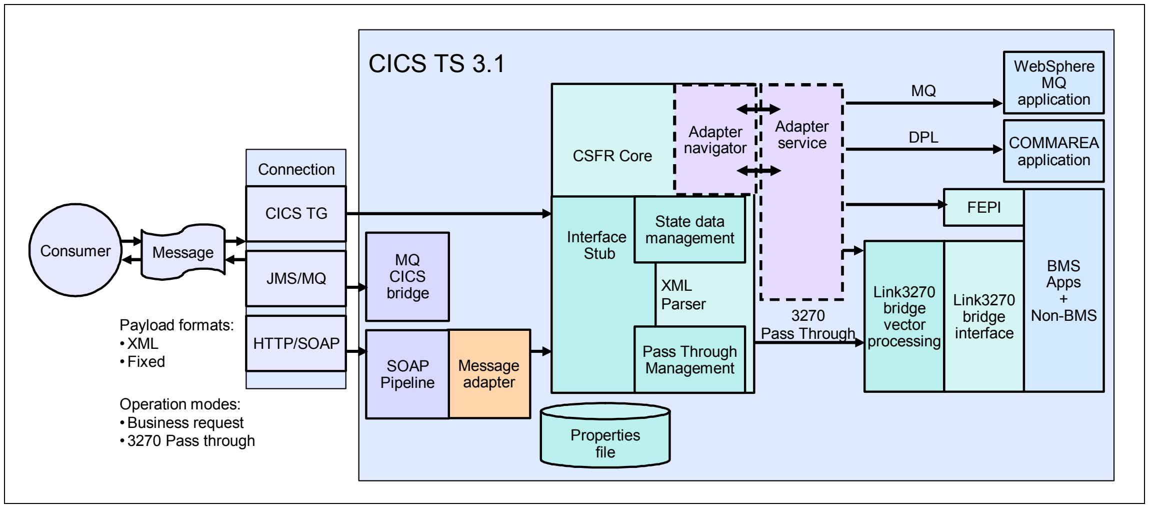


Figure 4-1 CICS SFR runtime

4.1.3 Service Flow Runtime and Link3270 bridge

The Link3270 bridge is described in 4.2, “Link3270 bridge” on page 84. Here we mention additional function provided by SFR when it is used to access terminal-oriented applications under the Link3270 bridge. This additional function supplements the function provided by the Link3270 bridge.

SFR receives the business request from the client and, with input mapping from the SFM generated service adapter, casts the request into bridge vectors for the Link3270 bridge.

As the service adapter interacts with the application through the Link3270 bridge, SFR always provides a view of the current 3270 screen. This not only includes the business data, but also field attributes (such as protect, color, and highlighting). In addition SFR provides internal 3270 screen emulation at the terminal level, using the Application Data Structure (ADS) and state management. This is required to insure:

- That protected fields are not inadvertently updated.
- That specific fields are returned on the next leg of a pseudo-conversation, even if the fields were not updated by the end user. This is indicated by the application setting the Modified Data Tag (FSET) in the field base attribute.
- That all required fields are presented to the service adapter (screen navigation). When the application issues a Send erase, and then on the next leg of the pseudo-conversation issues a Receive, Send no erase. This is a common technique to reduce transmission overhead and assumes that the Send no erase is only providing data to update an existing screen.

It may be necessary or preferable to have the business client connect to a single CICS *Routing region*. Transactions are received in the Routing region and routed to an appropriate Application Owning Region (AOR). The Link3270 bridge performs the routing either to a specific AOR where the application resides or to one of a group of AOR's for load balancing.

Multiple transactions in a single pseudo-conversation may execute in different AOR's. The CICS Service Flow Runtime transfers the terminal context (TCTUA and Return COMMAREA) between AOR's in a pseudo-conversation.

The Link3270 bridge provides a router program (DFHL3270). The router program provides load balancing of transactions defined to CICS as dynamic (Figure 4-2 on page 81).

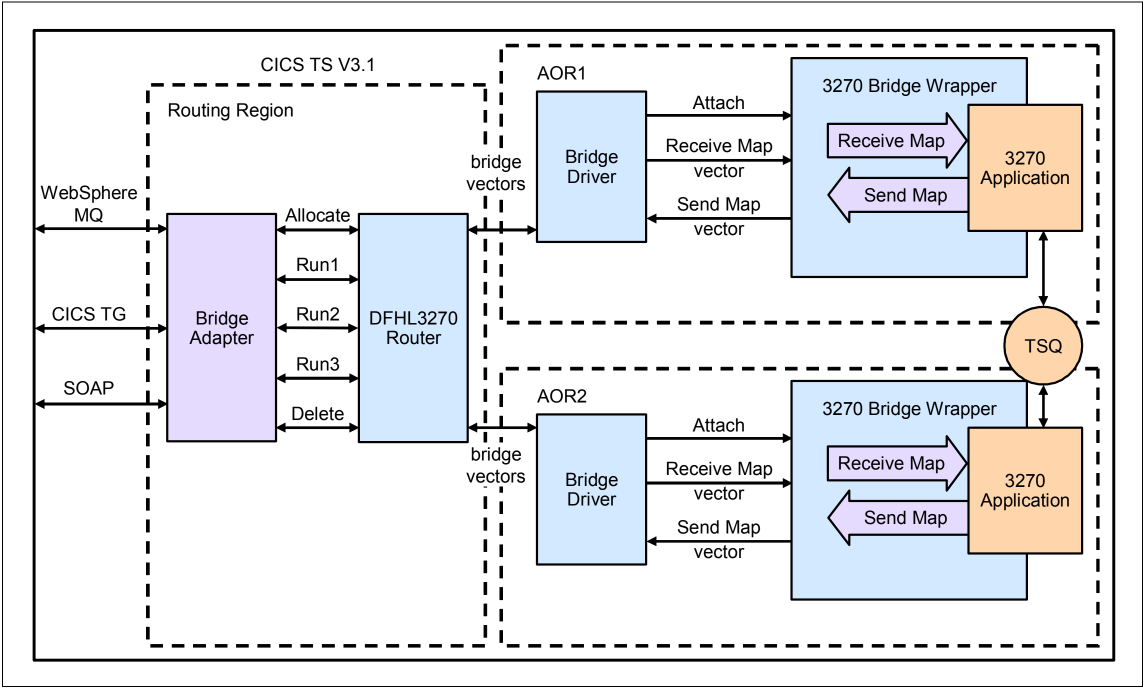


Figure 4-2 Shared application TS Queues

The Link3270 bridge in conjunction with the Terminal Autoinstall URM and the CICS Service Flow Runtime provides the ability to maintain the same terminal ID across AOR's in a pseudo-conversation. This allows application TS queues using the terminal ID as part of the name to be shared across AOR's.

4.1.4 Service Flow Modeler

Although the importance of host applications has not lessened, the mechanisms for accessing them have struggled to keep up with the modern trend - especially the SOA model. Some of the reasons for this lag are:

- Reluctance of owners to move away from the tried and trusted
- Unavailability of standard means to transform applications and data
- Dependencies between applications create coupling difficulties
- Escalating conversion costs even when transformations are feasible
- Growing attrition of expertise related to host systems

CICS has made available a variety of mechanisms to access applications. Figure 4-3 shows traditional terminal-based access to CICS applications.

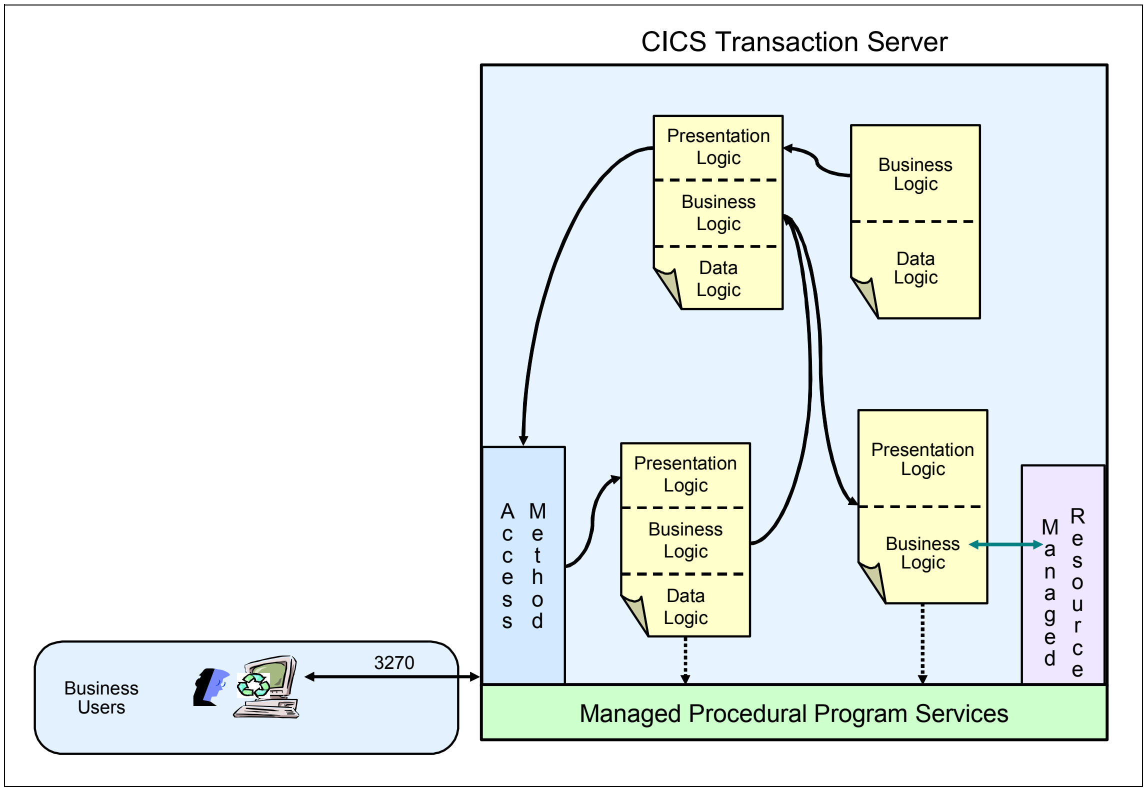


Figure 4-3 Traditional 3270 application processing

Figure 4-4 shows how CICS has responded to modern business technology needs and has evolved to enable access for both terminal and non-terminal paths.

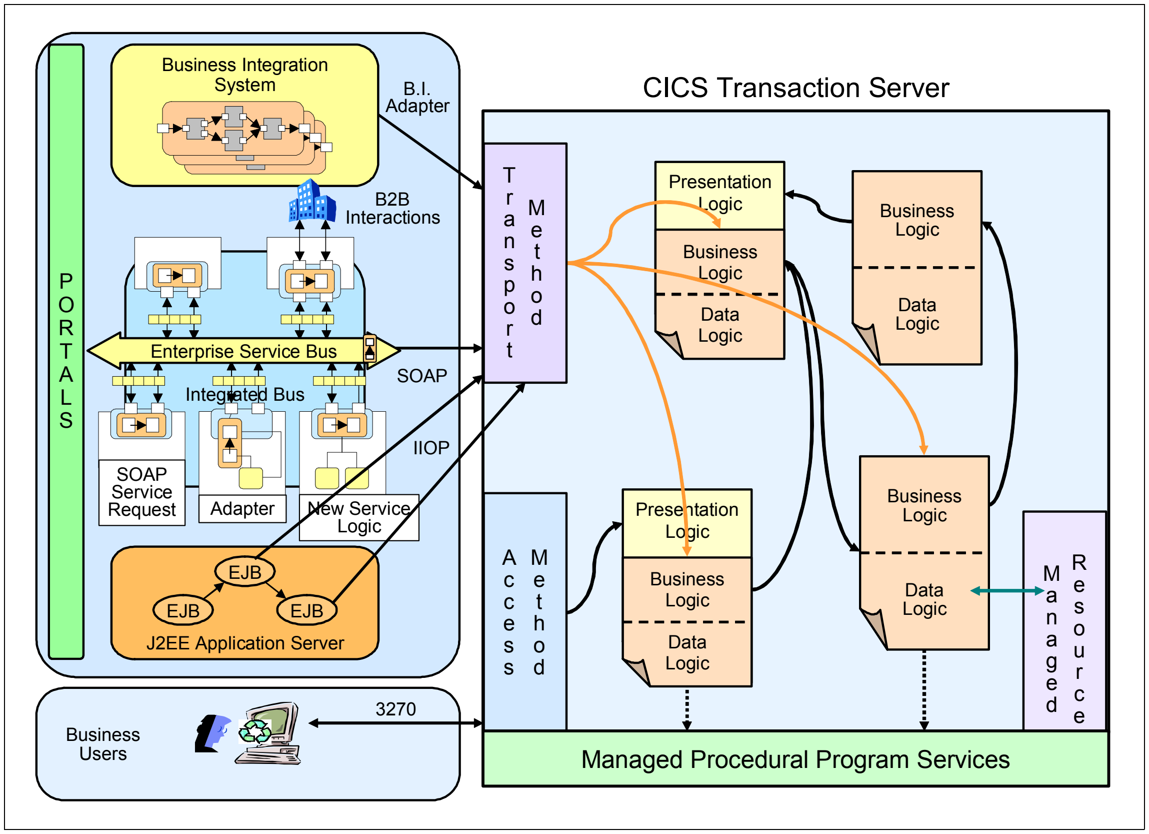


Figure 4-4 Integrating CICS Web services with 3270

We now have all of the access mechanisms that we could possibly want to integrate business systems in an unprecedented manner. Now SFM provides a means to re-use existing 3270 and COMMAREA applications by modeling business processes:

- Application flow, conversion, and integration are orchestrated in a Studio that is rich with host tools.
- Round-trip connectivity to the host system is provided for development as well as deployment
- Support is provided by Web service generation facilities in WebSphere Developer.

4.2 Link3270 bridge

The Link3270 bridge provides a callable interface to allow you to run 3270-based CICS transactions without emulating a 3270 terminal. In a Web-enabling environment this has the benefit that 3270 based user transactions can be retained within CICS where necessary, and access to them exposed via a more simple callable interface.

The 3270 terminal and end-user are replaced by an application program, known as the bridge client application. Commands for the 3270 terminal in the 3270 user transaction are intercepted by CICS and replaced by a messaging mechanism that provides a bridge between the client application and the 3270 user transaction.

Note: While you may write your own bridge client application, there is a fair amount of complexity involved in this effort. We strongly recommend using the CICS Service Flow Runtime which provides the bridge client function.

This mechanism was introduced in CICS TS V2.2 and provides a simplified link interface that can be accessed using a distributed program link (DPL), an ECI request or an EXCI request. All messages have a fixed vector format. This mechanism supports CICSPlex® SM load balancing; bridge facilities are shared between CICS regions on the CICSPlex.

The bridge client requests services of the Link3270 bridge using COMMAREA messages in a prescribed format called bridge vectors, and the Link3270 bridge returns results to the bridge client in formatted messages.

The Link3270 bridge consists of several components as shown in Figure 4-5 on page 85.

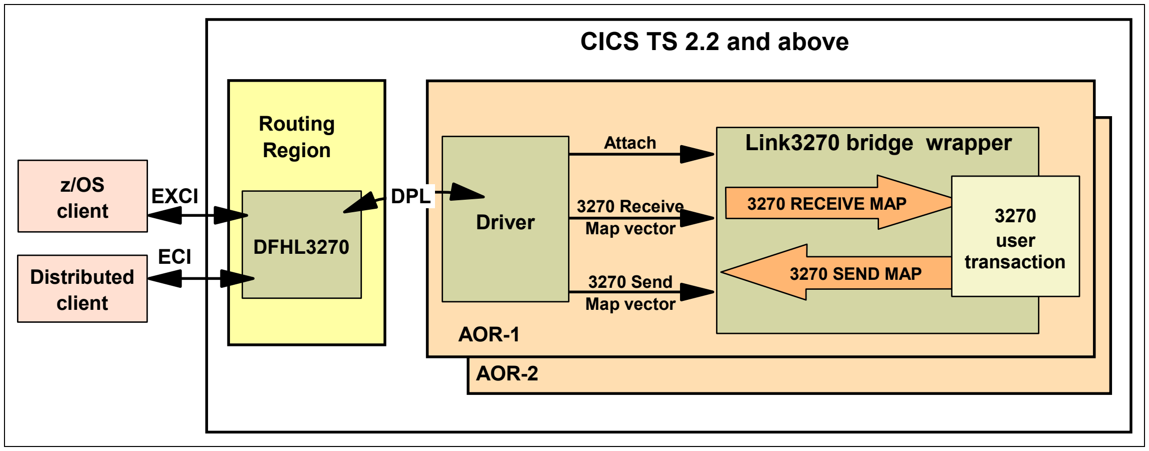


Figure 4-5 Link3270 bridge

The components in Figure 4-5 are discussed here:

- 3270 user transaction

  The user transaction can be any 3270-based CICS transaction. It is accessed from a bridge client application through the Link3270 bridge mechanism.

- Bridge facility and FACILITYLIKE definition

  The bridge facility is a virtual terminal that provides the appearance of a terminal to the user transaction. The FACILITYLIKE definition is the name of a real terminal definition from which the bridge facility takes some of its properties.

- Link3270 bridge router (DFHL3270)

  The Link3270 bridge router, DFHL3270, dynamically routes the request to the bridge driver task, which may be in the same or another CICS region. Load balancing can be implemented in this way.

- Link3270 bridge driver

  The Link3270 bridge driver starts the bridge wrapper, which creates the bridge environment for the 3270 application transaction.

- Link3270 bridge wrapper

  The Link3270 bridge wrapper is also invoked by CICS when terminal commands are issued by the existing 3270 application in the bridge environment.

The Link3270 bridge was first made available in CICS TS 2.2 and enhanced in CICS TS 2.3.

CICS TS 2.3 and above Link3270 features

There are two levels of support for the Link3270 bridge in CICS TS 2.3.

- Link3270 bridge with basic support provides the same support as that provided in CICS TS 2.2. If you want to continue using basic support, you do not need to take any action. Basic support is provided automatically.
- Link3270 bridge with extended support provides support for the ACCUM option on EXEC CICS SEND TEXT, EXEC CICS SEND MAP, and EXEC CICS SEND CONTROL, in addition to the basic support. To support the ACCUM option, there are two new outbound vectors, SEND PAGE and PURGE MESSAGE. If you want to take advantage of the extended support provided by CICS TS 2.3, you must recompile your Link3270 bridge client programs using the extended copybooks instead of the basic copybooks.

4.2.1 Link3270 bridge security

We will briefly examine the security issues when using the Link3270 bridge.

The rules governing Link3270 bridge security are fairly simple and are summarized in the following list:

- Authorization to DFHL3270 is controlled by the transaction security settings which are in effect for the specified transaction.
- Resource security can be used to protect the LINK requests to the DFHL3270 router program and the Link3270 bridge driver program.
- Transaction security is enforced for the 3270 user transaction running in the AOR under the Link3270 bridge.
- When using session mode, the security context is established on the initial (allocate) request to DFHL3270 and cannot change during the session.
- The 3270 application program cannot issue the **`EXEC CICS SIGNON`** command.

If using the ECI to call the Link3270 bridge, the authentication and authorization of the user ID used to invoke the bridge depends on the standard security procedures for ECI requests. If using a CICS TG on a distributed platform, the user ID and password are flowed in each ECI request and will be verified by the CICS external security manager, such as RACF. If using the CICS TG on z/OS, the Gateway daemon can authenticate the user ID and password, and then authorization is controlled by the user ID flowed to CICS.

For a distributed program link (DPL), issued from within CICS, the standard MRO security rules apply, including the considerations for link security and equivalent systems. For further details, refer to the redbook *Securing Web Access to CICS*, SG24-5756.

4.2.2 Link3270 bridge transactional scope

An external client transaction may transmit a request to CICS that results in a CICS transaction being executed under the Link3270 bridge. In this case the external client executes under a transactional scope defined by its local resource manager. However, the transactional scope of the CICS transaction executing under the Link3270 bridge conforms to the scope for a traditional CICS transaction which issues an implicit sync point at every task termination. This is illustrated in Figure 4-6.

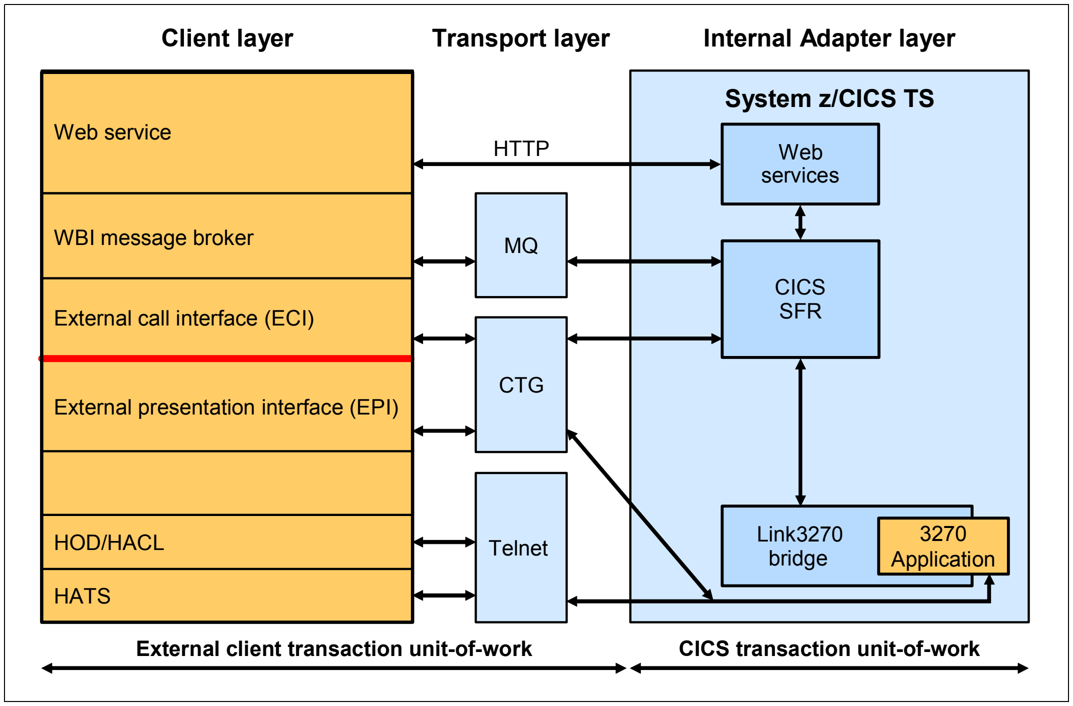


Figure 4-6 Single request resulting in two units of work

Updates to recoverable resources for transactions run under the Link3270 bridge may be backed out in either of two ways.

- Prior to the conclusion of the transaction the application program may issue a SYNCPOING ROLLBACK command.
- If the CICS-transaction has terminated a subsequent request must be submitted that contains the appropriate compensation logic.

Note that updates to recoverable resources for CICS transactions run under the Link3270 bridge do not participate in the transactional context of the calling application. For instance a if a ECI call from the CICS TG is used, the 3270 transaction that is invoked by the Link3270 bridge will *not* be backed out, if the ECI call issues a roll-back.

4.2.3 Link3270 bridge performance

The principle advantage of the Link3270 bridge is that it accesses 3270 transaction data by using the Application Data Structure (ADS) rather than screen scraping. Therefore, 3270 emulation is performed at the API level in CICS. This avoids generating and transmitting an entire 3270 data stream, including screen constants (headers and titles), to an external platform where it must be interpreted.

In addition, 3270-application screen navigation logic may be performed in a bridge client within CICS. This approach can dramatically reduce flows to an external business client, resulting in improved performance and scalability compared with other screen scraping solutions. WebSphere Developer SFM generates service adapters for this purpose.

4.2.4 Link3270 bridge application development

The Link3270 bridge provides an interface to enable 3270-based CICS transactions to be invoked using a LINK request, and without the facilities of a 3270 terminal. As such, it is not a means of providing Web access (since there is no direct means of generating HTML), but it is an important enabling technology because it allows a client module to link to existing 3270 transactions.

Link3270 bridge router, driver, and wrapper

The client requests services of the Link3270 bridge via COMMAREA messages in a prescribed format. The Link3270 bridge router invokes the 3270 user transaction and returns results to the client in formatted messages.

The client application LINKs to program DFHL3270, a component of the Link3270 bridge, with a message in the COMMAREA. The LINK can be in any form that CICS supports: a standard local LINK, a distributed program link, a CICS client ECI call from another platform, or an EXCI from a non-CICS region in the same z/OS sysplex.

Programming modes

There are two programming modes for using the Link3270 bridge: single transaction mode and session mode.

Single transaction mode

The single-transaction mode is a simple, one-shot interface, in which each LINK to a DFHL3270 request results in the execution of one transaction under the bridge, and each request is independent of previous ones. The client has to send all of the information required to run the transaction in the inbound COMMAREA, and all of the transaction output has to fit into the COMMAREA upon return. The key characteristic of single-transaction mode is that the Link3270 bridge creates a new bridge facility for each request and discards it upon completion. The effect of this is that no state is maintained between requests, because the bridge facility (the terminal) is the anchor for state data: the TCTUA, next transaction identifier, COMMAREA, START data, and so forth.

Session mode

In this mode, the bridge facility is kept over a series of transactions, making it suitable for transaction sequences that depend on state being maintained between transactions. This is illustrated in Figure 4-7 on page 90. The most common requirement for preservation of state data between transactions is a pseudo-conversational sequence, and for this reason the Link3270 bridge returns the next transaction identifier to the client as well as the transaction output and status information. Some applications and some installations associate state data with a terminal over even longer sequences, and session mode also meets this requirement.

Note: CICS Service Flow Runtime uses the Link3270 session mode.

There is more flexibility about the flows between the client and the Link3270 bridge in session mode. The client can provide input for a transaction across multiple LINKs, as might be necessary in a conversational transaction where the client cannot provide the input without inspecting the previous output. Output that exceeds the length of the COMMAREA can be delivered piecemeal, and output lost because of a link failure can be recovered and resent.

The session begins with an allocate request, in which the client LINKs to DFHL3270 with a message that requests a new bridge facility (essentially the bridge equivalent of auto-installing a terminal). CICS provides a default message, which can be used unchanged for this purpose, but the client can also change fields in the message to define the characteristics of the bridge facility. The client can name an installed terminal to be used as a model, and can specify the terminal name, NETNAME, user ID, and keep time. The keep time is the length of time that the bridge facility can remain idle between transactions before CICS discards it.

The Link3270 bridge responds to an allocate request with a token that represents the bridge facility. The client must return this token on every subsequent request in the session to identify the session and the bridge facility to which state data is anchored. The TERMID and NETNAME are also returned, because they will have changed if the client did not specify them or if they were modified by the autoinstall URM.

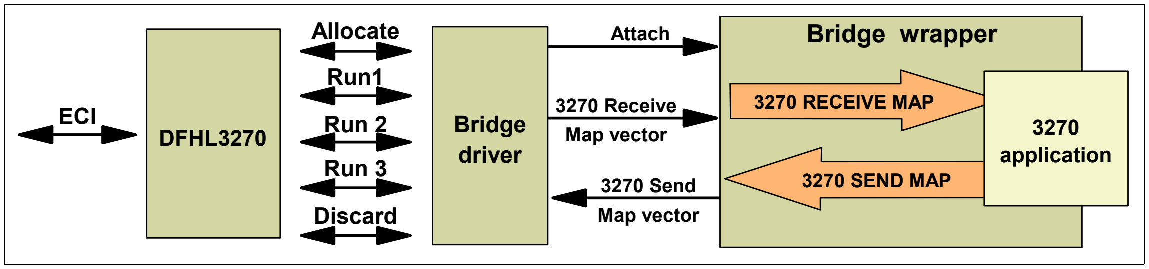


Figure 4-7 Link3270 bridge, session mode

The client now starts running transactions in sequence. It initiates each one by LINKing to DFHL3270 with a run-transaction request message as in Figure 4-7. This run-transaction message contains the transaction code to be run, the bridge facility token from the allocate request, and input for any terminal input commands (**RECEIVE**, **RECEIVE MAP**, **CONVERSE**) in the transaction. Session mode offers considerable flexibility to the client for providing input, as we will describe later, but usually the input is in the run message. The client can specify a number of other values as well, although defaults are provided. These include values for the attention identifier and cursor position at task attach time, the start code (how the transaction should think it was initiated), and so on.

The messages used to communicate between the client and the bridge router consist of a bridge header followed by one or more bridge vectors. Each vector corresponds to input for one CICS command in an inbound message, and to output from one CICS command or a request for input for one command in an outbound message. Figure 4-8 on page 91 illustrates the bridge vector format.

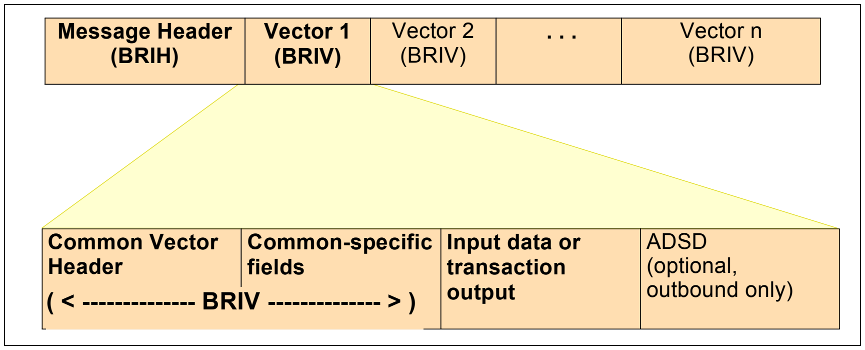


Figure 4-8 Link3720 bridge, vector format

Client design considerations

The 3720 bridge client function may be implemented within CICS, in another z/OS address space, or a distributed platform. Client function may include screen navigation logic and business logic.

Generally, there are two client models. With the *combined-client* model, the client contains both the business and screen navigation logic and the formatting and interpretation logic for the Link3270 messages. Since variable ADS formats containing both binary and display data are contained in the messages, complex code page conversion is required for the combined-client model. The top portion of Figure 4-9 on page 92 shows the combined-client model for the Link3270 bridge.

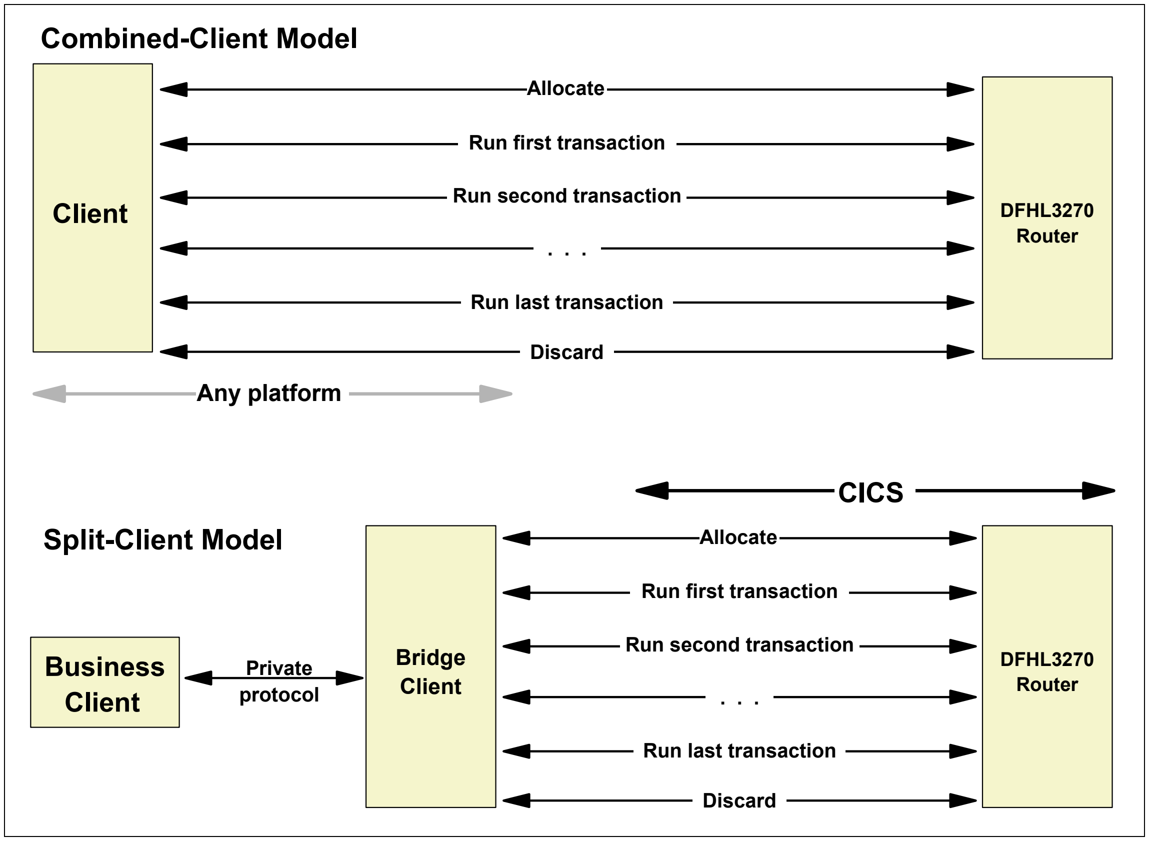


Figure 4-9 Two client models

The bottom portion of Figure 4-9 shows the *split-client* model. With the split-client model, the client is broken into two parts: a business client and a bridge client. The business client is responsible for the business logic.

Note: CICS SFR is an implementation of the split-client model with enhanced bridge client function.

Navigation logic, for 3270 screens, may be contained in either or both clients. For example, static navigation might consist of a fixed series of screens where the screen input and function keys (such as Enter, PF1, and so on) are known ahead of time. Static navigation might be implemented in the bridge client. Variable navigation logic is required when data returned from a response screen is used to determine the field input or function key for the next request. Variable navigation logic might be considered business logic and therefore, appropriately implemented in the business client. However, implementing variable navigation logic in the bridge client can dramatically reduce flows between the two clients, resulting in improved performance.

The business client passes its requests and receives its results through the bridge client. The interface between the two is private to them and can be anything on which they agree.

The bridge client casts the requests into Link3270 request messages or bridge vectors, does the necessary LINKs to DFHL3270 to run the 3270 transaction, converts and combines the responses to the private format and responds back to the business client.

The advantage of splitting the client is that the two parts can be written by different programmers, perhaps in different languages, allowing installations to exploit existing programmer skills. In addition, the two parts of the client can run on different platforms. For example, the business client might be a Java servlet, and the bridge client might be a CICS program. For some applications, a single, general-purpose bridge client could provide an application-specific API for a variety of business clients.

We recommend that a split-client design be used with the Link3270 bridge, such as that implemented by CICS SFR. However, the Link3270 bridge does also support code page conversion for a combined-client design. For additional information refer to Chapter 3 “Using the Link3270 bridge” in the *CICS External Interface Guide*, SC34-6449.

Screen design considerations

CICS application programs have traditionally used screen design and interaction techniques that should be taken into account when running 3270 transactions in the 3270 bridge environment. The issues that arise from the following scenarios can be appropriately handled in the bridge client of a split-client model.

3270 field attributes

Each field on a formatted 3270 screen has a base attribute that controls display intensity, the ability of the operator to enter data into the field, and whether supplied data is returned to the program if the operator does not overtype the field. Optionally, a field may also have extended attributes that control text color and highlighting. The base and extended field attributes may be given initial values in the Basic Mapping Support (BMS) source.

A BMS source example is shown in Table 4-1 on page 94.

```
DFHOCGB  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=COBOL,TIOAPFX=YES,EXTATT=MAPONLY
DETAIL   DFHMDI SIZE=(12,40)
TITLE    DFHMDF POS=(1,15),LENGTH=12
         DFHMDF POS=(3,1),LENGTH=8,INITIAL='NUMBER:',COLOR=BLUE
NUMB     DFHMDF POS=(3,10),LENGTH=6
         DFHMDF POS=(3,17),LENGTH=1
         DFHMDF POS=(4,1),LENGTH=8,INITIAL='NAME:   ',COLOR=BLUE
NAME     DFHMDF POS=(4,10),LENGTH=20,ATTRB=(UNPROT,IC)
         DFHMDF POS=(4,31),LENGTH=1
         DFHMDF POS=(5,1),LENGTH=8,INITIAL='ADDRESS:',COLOR=BLUE
ADDR     DFHMDF POS=(5,10),LENGTH=20,ATTRB=UNPROT
         DFHMDF POS=(5,31),LENGTH=1
         DFHMDF POS=(6,1),LENGTH=8,INITIAL='PHONE:   ',COLOR=BLUE
PHONE    DFHMDF POS=(6,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(6,19),LENGTH=1
         DFHMDF POS=(7,1),LENGTH=8,INITIAL='DATE:    ',COLOR=BLUE
DATE     DFHMDF POS=(7,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(7,19),LENGTH=1
         DFHMDF POS=(8,1),LENGTH=8,INITIAL='AMOUNT:  ',COLOR=BLUE
AMOUNT   DFHMDF POS=(8,10),LENGTH=8,ATTRB=NUM
         DFHMDF POS=(8,19),LENGTH=1
         DFHMDF POS=(9,1),LENGTH=8,INITIAL='COMMENT:',COLOR=BLUE
COMMENT  DFHMDF POS=(9,10),LENGTH=9,ATTRB=UNPROT
         DFHMDF POS=(9,20),LENGTH=1
MSG1     DFHMDF POS=(11,1),LENGTH=39
MSG3     DFHMDF POS=(12,1),LENGTH=39
         DFHMSD TYPE=FINAL
         END
```

Table 4-1 BMS source example

The 3270 application program may, at execution time, override these initial attribute values. Bridge client applications that receive ADS vectors from the 3270 bridge should be able to handle 3270 field attributes as in the following scenario.

A typical program may use a BMS map to collect input data when opening a new customer account. The program would set the base field attributes to `UNPROTect` (allowing data entry) and normal intensity with a default extended color attribute of `green`. If operator input errors are detected when editing the data, the program might return the error field to the screen with the base attribute set to high intensity, extended color attribute set to the color red. The cursor could be positioned to the error field by the setting the length field to `a-1`, which normally results in the generation of an Insert Cursor order in the 3270 data stream. Fields that passed editing might be returned with the base field attribute set to `PROTect` (do not allow data entry) with a modified data tag (MDT/FSET) set to `on`. This prevents an operator from updating a field that was successfully edited and will also return the field as though it had been modified.

The preceding scenario is typical of many applications and indicates both a cosmetic (field color and intensity) and operational (allowing or restricting data entry) requirement for dynamic 3270 field attribute modification by CICS applications. For this reason, we see that 3270 field attributes can convey information in addition to the actual field data.

While the 3270 field attribute information can be sent to the business client in a split-client solution, the modified data tag information can be maintained as state data in an ADS screen buffer.

Attention ID and relative cursor position

Some 3270 applications rely on detecting the screen relative cursor position (RCP) when the operator presses an Attention ID key (Enter, PA, or PF key). This is a technique that is more common when processing a menu screen. The operator tabs to a protected menu item and presses the **Enter** key. The relative cursor position is available to the application in the EIBCPOSN. The EIBCPOSN field is set by the Link3270 bridge in BRIH-CURSORPOSITION. If RCP values are required either the business or bridge client must supply them.

The AID key value is available to the application in the EIBAID field. The EIBAID field is set by the Link3270 bridge in BRIH-ATTENTIONID. Either the business or bridge client may supply AID key values.

BMS initial values

In a BMS source map initial data for a field may be supplied by using the `INITIAL=` parameter on the DFHMDF macro. The value of this parameter is used to supply data to the field if a **`EXEC CICS SEND MAP MAPONLY`** is issued or if no data is supplied from the application program, signified by the first character in the field being set to NULL.

When an application issues a **`SEND MAP MAPONLY`** command, no application data or field attributes are supplied to the Link3270 bridge. The bridge client or business client may need to supply default data, which, for 3270 transactions *not* running under the Link3270 bridge, are supplied from the 3270 physical map.

When the application issues a **`SEND MAP`** (without **`MAPONLY`**), the bridge exit receives the data supplied by the application program in the ADS (symbolic map).

However, if the application program does not supply data for a field (NULL value in the first character) then initial values, normally supplied by the physical 3270 map, may need to be supplied by the bridge client, or business client.

Overlaying an existing screen with SEND MAP

3270-display hardware retains the screen image in the hardware buffer after an attention key (Enter key, PF keys or PA keys) is pressed. If the application program subsequently issues a **`SEND MAP DATAONLY`** without specifying the ERASE option, then the 3270 device places the newly supplied data on the screen overlaying any previous data in the same screen positions. Previous screen data that is not overlaid continues to be displayed. This is a commonly used technique when editing input screens where only error messages are returned to the display. The operator can correct the error, and change any other information, before re-submitting the input screen.

Since the CICS application program assumes that it is communicating with a 3270 display and expects that the previous presentation image will be overlaid with newly supplied data, the bridge client may need to emulate the operation of the 3270 display. This can be accomplished by saving the screen contents of the output screen on a **`SEND MAP`**, **`RECEIVE MAP`**, **`SEND MAP DATAONLY`** (without erase) sequence. Data sent on the **`SEND MAP`** must be saved in an ADS screen buffer. Data that is returned from the browser must be formatted in the ADS to satisfy the **`RECEIVE MAP`**, and it must also be merged with the saved ADS screen buffer. New data supplied by the application on the next **`SEND MAP DATAONLY`** (without erase) must also be merged with the saved ADS screen buffer. The ADS screen buffer can then be returned as the response from the bridge client.

Note: CICS SFR provides support for all the screen design considerations mentioned above.

CICS application design considerations

CICS application programs have traditionally used design techniques that should be taken into account when running 3270 transaction in the 3270 bridge environment.

3270 data stream

Most CICS 3270 application programs are written using CICS Basic Mapping Support (BMS) commands. These application programs send data to a terminal by formatting an Application Data Structure (ADS), containing data fields and 3270 attributes, and then issuing a **`SEND MAP`** command. CICS uses the application supplied ADS and the physical map, from assembled BMS source, to generate a 3270 data stream. CICS transforms an inbound 3270 data stream into an ADS, which is used to satisfy a **RECEIVE MAP** command issued by the application. When using the 3720 bridge, with application programs using BMS, the ADS is supplied or received to satisfy the **SEND MAP** and **RECEIVE MAP** commands.

Some application programs process their own 3270 data streams. These application programs send the 3270 data stream to a terminal by issuing a **SEND** command, and receive the 3270 data stream by issuing a **RECEIVE** command. The IBM Development Management System for CICS (DMS/CICS) was developed using this design. When running applications like DMS/CICS under the 3720 bridge, a 3270 data stream must be supplied or received to satisfy **SEND** or **RECEIVE** commands. Table 4-2 illustrates the contents of an example inbound 3270 data stream.

Table 4-2 Example inbound 3270 data stream

Location	Contents	Description
1	X'7D'	AID, in this case the Enter key
2-3	X'C3C5'	Cursor address: line 3, column 38, where the operator left it after the last data key stroke.
4	X'11'	Start Buffer Address (SBA), indicating that a buffer address follows.
5-6	X'C26E'	Address of line 3, column 15, which is the starting position of the field to follow.
7-12	'123456'	Input, the employee number entered by the operator.
13	X'1D'	Start Field (SF), indicating another input field follows.
14-19	‘ABC987'	Input field, license plate number.

While the Link3720 bridge supports the **SEND** and **RECEIVE** commands, the bridge client must be able to parse and generate 3270 data streams.

Note: CICS SFR supports a **SEND** command for a 3270 data stream containing a single line of text.

Link3270 bridge restrictions

There are a few minor restrictions when using the Link3720 bridge. These are documented in Chapter 2 “Link3270 programming considerations” in *CICS External Interface Guide*, SC34-6449.

4.2.5 Summary

To summarize, the Link3270 bridge offers you a powerful new tool for extending the lifetime of existing 3270-based transactions. It allows you to use the business logic in large bodies of existing code in new combinations, from new platforms, with new user interfaces. You do not need to modify the old code, and you can use it in the original 3270 form at the same time as you use it in the new environment.

There are few restrictions on how you build your interface to the new source of input and output, and yet the bridge overhead is relatively low, probably comparable to the existing code running with 3270.

While developing a full function bridge client to interact with the Link3270 bridge can be complex and challenging, using the WebSphere Developer Service Flow Modeler with CICS Service Flow Runtime provides graphical tooling and an enhanced bridge client implementation.

Part 2

CICS Web technology decisions

In Part 2, we provide an in-depth analysis of the important factors that are likely to affect your choice of CICS Web-enablement technology. This includes factors that affect security, transactional scope, performance, application development, and data conversion.

5

Security

When you open your business to the Internet, you might change the landscape of users on your system dramatically. Whereas you have very strict control over the access to your applications through conventional 3270 terminals, your users of applications might be in remote locations, and you cannot be sure what intentions they will have. You might have better control over users in an intranet or extranet environment, but you still will be faced with new security issues, such as how to authenticate users and how to authorize access to your data.

When you consider the security design for your application, you will need to weigh the following key issues:

- Do you want the runtime infrastructure to authenticate the users of the applications or will the applications do it themselves?
- Will CICS do authentication itself or will an external server, such as WebSphere Application Server, do authentication?
- What authorization mechanisms will be used to protect access to the CICS system and access to resources such as transactions, files, and databases?
- How will you protect the confidentiality of data which is transported between the different tiers of the physical configuration?

In this chapter, we first provide an introduction to security technologies as they relate to securing access to CICS applications. Then we discuss the security considerations for each of the main CICS SOA technologies.

5.1 Security concepts

The five parts of a security architecture are described by the International Organization for Standardization in standard ISO 7498-2:

- Authentication
- Authorization
- Data integrity
- Confidentiality
- Non-repudiation

Authentication

Authentication means confirming someone's identity. Typically, this involves a sign-on mechanism based on a predefined secret such as a user ID and password. RACF is a product that provides this type of authentication. For Internet transmissions, authentication also means confirming the identity of the message participants.

Authorization

Authorization involves checking that the authenticated users have access to the system resources needed by the tasks they are performing. These resources can include computer systems, application functions, transactions, programs, databases, files, and other CICS resources.

Data integrity

Data integrity means ensuring that data is not lost or tampered with. Although this includes stored data, we are interested mainly in data transmission over the Internet and ensuring that the data received is the same as the data that was sent. This involves checking that no data has been lost in the transmission and that the data has not been modified.

Confidentiality

Confidentiality is also referred to as privacy. It means ensuring that data is not disclosed to people or systems not authorized to read that data. Although privacy issues include stored data, our main concern is with data transmissions over the Internet. A common technique to provide privacy in data transmissions is *encryption*.

Non-repudiation

Non-repudiation means ensuring that an action cannot be denied by the person who performed it. For messages over the Internet, this means the ability to prove that someone actually sent a message, so that they cannot deny it later. A key factor for non-repudiation is *logging* to have this *proof after event*.

5.2 Introduction to digital security

In this section, we introduce the main concepts of cryptography and digital signatures and then discuss the major security technologies that can provide building blocks for designing your secure CICS system.

5.2.1 Cryptography and digital signatures

Cryptography is the scientific discipline for the study and development of ciphers, in particular, encryption and decryption algorithms. These cryptographic procedures are the essential components that will enable secure communication to take place across non-secure networks.

We briefly introduce the following terminology:

- Secret (or symmetric) keys
- Public/private (or asymmetric) keys
- Digital signatures

Secret (or symmetric) keys

Secret key cryptography means that the sender and receiver share the same (hence *symmetric*) key which is used to encrypt and decrypt the message, as shown in Figure 5-1.

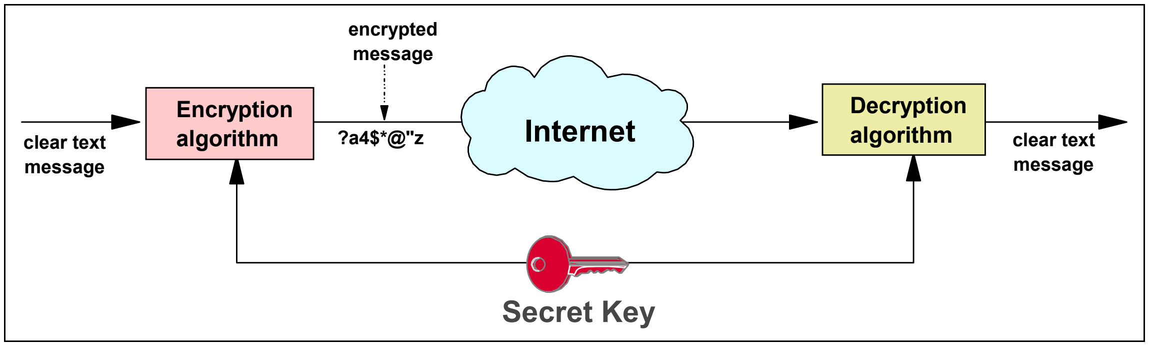


Figure 5-1 Secret key encryption and decryption

Secret key encryption/decryption performs reasonably fast and is often used to provide privacy for high-volume data transmissions. However, the symmetric key is a *shared secret* and it requires a secure channel for exchanging the key initially; that is why symmetric keys are often used in combination with asymmetric keys.

Public/private (or asymmetric) keys

Public/private key cryptography uses an asymmetric algorithm. The encryption flow is visualized in Figure 5-2. The private key is known only by its owner and is never disclosed. The corresponding public key can be known by anyone. The public key is derived from the private key, but it cannot be used to deduce the private key. Either key of the pair can be used to encrypt a message, but decryption is only possible with the other key.

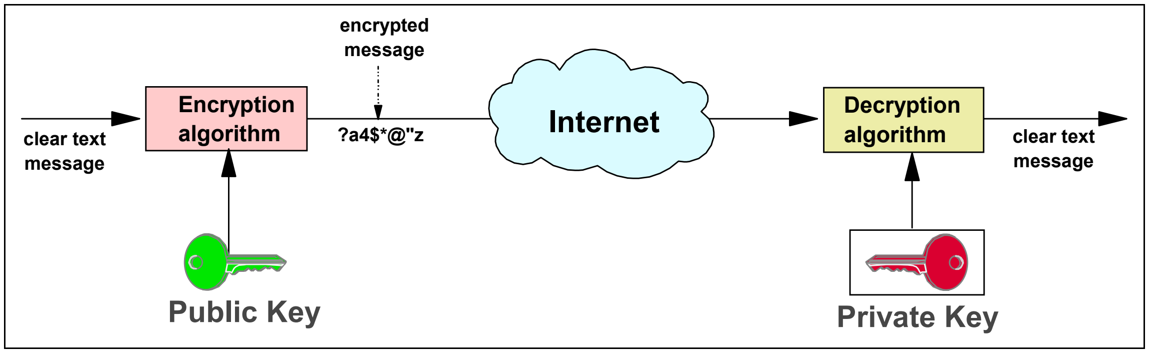


Figure 5-2 Public/private key encryption and decryption

Digital signatures

A message digesting algorithm (or *hash*) condenses a block of data into a shorter string, the *message digest* (MD). The principle behind MDs is that the message cannot be recovered from its digest, and that it is very hard to construct two different blocks of data with the same MD. When the digest is sent with the message, the receiver can use it as a data integrity check. However, this will usually require that the digest itself be secured against third-party attacks by encrypting it. The message digest is then called a *message authentication code* (MAC).

A MAC can also provide non-repudiation if the sender's private key is used to encrypt the MD. The MAC is then called a *digital signature* and will carry the proof of ownership of the transmitted message (Figure 5-3 on page 105).

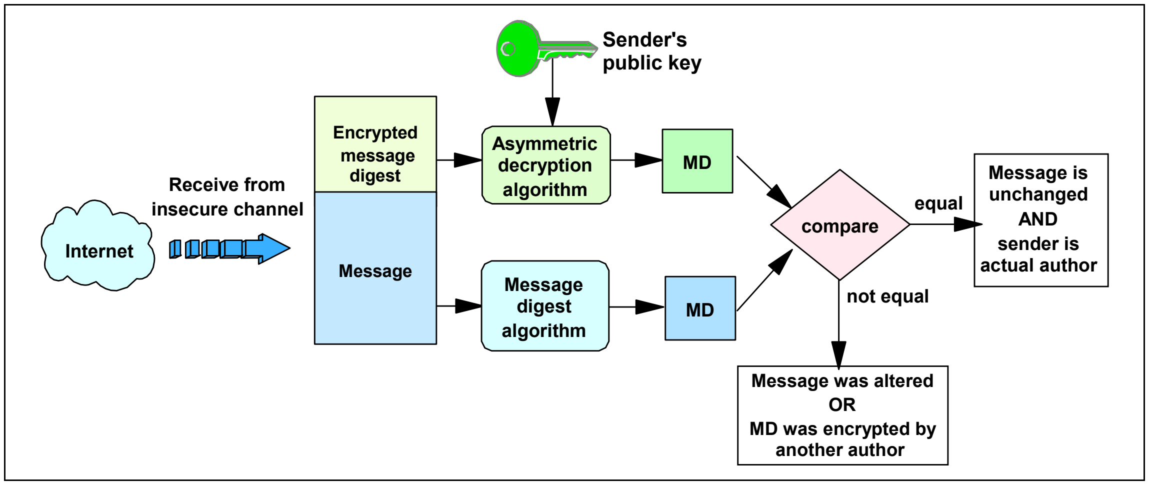

Sender's public key
Encrypted message digest
Asymmetric decryption algorithm
MD
Receive from insecure channel
Internet
Message
compare
equal
Message is unchanged AND sender is actual author
Message digest algorithm
MD
not equal
Message was altered OR MD was encrypted by another author

Figure 5-3 Digital signature verification

5.2.2 Security technologies

We will discuss these technologies in the following sections:

- Firewall
- De-militarized zone (DMZ)
- Digital certificates and certification authorities
- Secure Sockets Layer (SSL)

Firewall

A firewall, as shown in Figure 5-4, secures an internal, trusted network against some external, untrusted network such as the Internet.

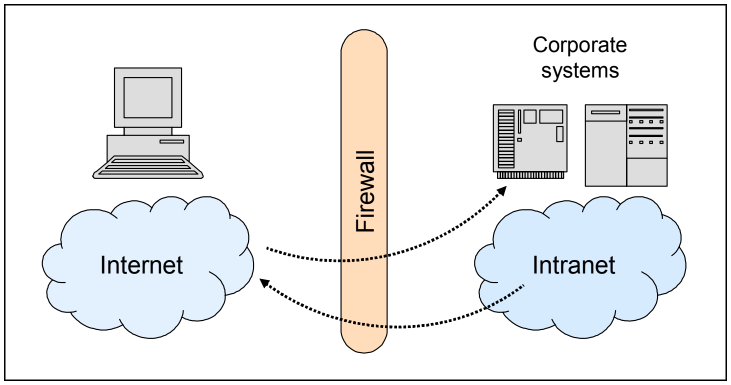


Figure 5-4 What is a firewall?

The firewall may be created on a stand-alone, dedicated computer configured with an IP packet-filtering router, proxy servers for various TCP/IP applications such as HTTP and FTP, and a SOCKS server to provide a secure gateway for TCP/IP sockets.

De-militarized zone (DMZ)

This is a type of firewall configuration that places your security software and hardware between two firewalls, as shown in Figure 5-5. The outer firewall controls traffic between the Internet and the DMZ, and its job is to filter out unwanted protocols and restrict inbound traffic to a limited set of IP addresses and ports. It only advertises the DMZ to the external network.

The inner firewall controls the traffic between the DMZ and the internal network, and prevents unauthorized access from the DMZ to your secure network, and conversely restricts access from the secure network to the non-secure network or Internet.

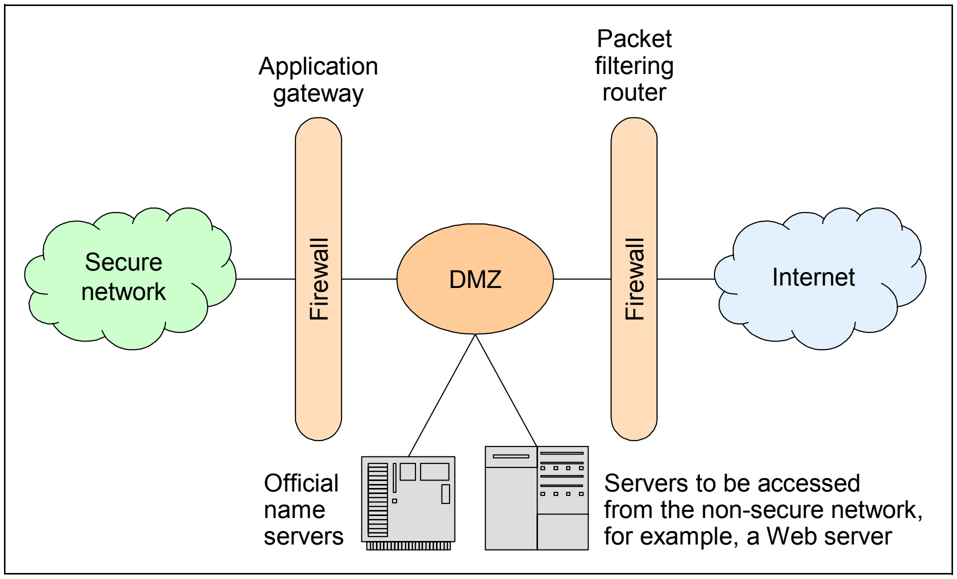


Figure 5-5 A de-militarized zone (DMZ)

The DMZ is where you place the servers for the applications that you want available on the Internet, such as Web servers, proxy servers, and SOCKS servers.

Digital certificates and certification authorities

One of the problems with digital signatures is the secure transmission of the public key. Digital certificates address this problem by binding an identity to a public key. This identity is one of a trusted third party, the *certificate authority* (CA) which, rather like a government passport office, certifies the public key and its owner's identity.

The resulting digital certificate is a file, containing someone's public key and identification, which has been encrypted with the private key of the CA. It conforms to the X.509 format and is also referred to as an X.509 certificate. Since nobody knows the CA's private key, nobody can falsify a digital certificate. See the diagram in Figure 5-6.

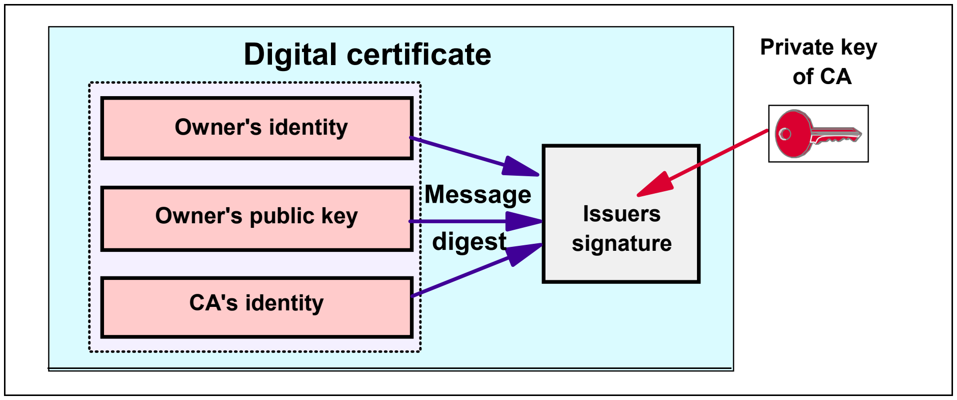


Figure 5-6 A digital certificate

To further complicate matters, there are many different CAs and one CA may not trust the others. The solution to this problem is a *trust-chain* hierarchy of CAs, where each CA has a digital certificate signed by a CA higher up the chain. You simply start with the digital certificate and, using the public keys of the CAs, work up the chain until you find a CA that you trust, as shown in Figure 5-7 on page 108.

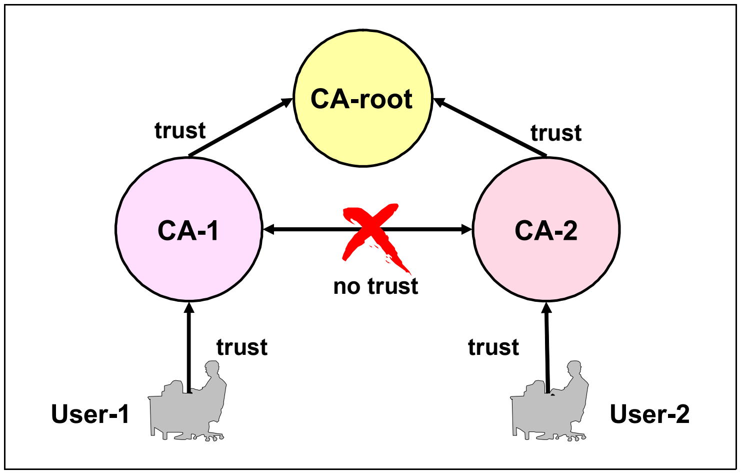


Figure 5-7 Certificate authority trust-chain

Digital certificates signed by a CA are *externally signed* certificates. Digital certificates can also be *self-signed*. A self-signed certificate cannot be used for secure public key distribution because the public key required to decrypt the certificate is the same as the key encrypted in it. The exception to this is a root certificate which by definition is self-signed, implying a degree of trust in the authority that distributed the certificate (and claimed to sign it). Apart from that, both types are technically equivalent, and can be used with Secure Sockets Layer protocol, which we discuss next.

Secure Sockets Layer (SSL)

SSL is a security protocol developed by Netscape Communications and RSA Data Security, which provides a private communication channel over a TCP/IP connection. It implements several of the cryptographic techniques discussed above. The client initiates an SSL Web connection by using a URL starting with `https:` instead of `http:`. With SSL, the data flowing back and forth between client and server are encrypted using a secret key algorithm. The exchange of the secret key occurs at the start of the communication during the *SSL handshake*. This is illustrated in Figure 5-8 on page 109.

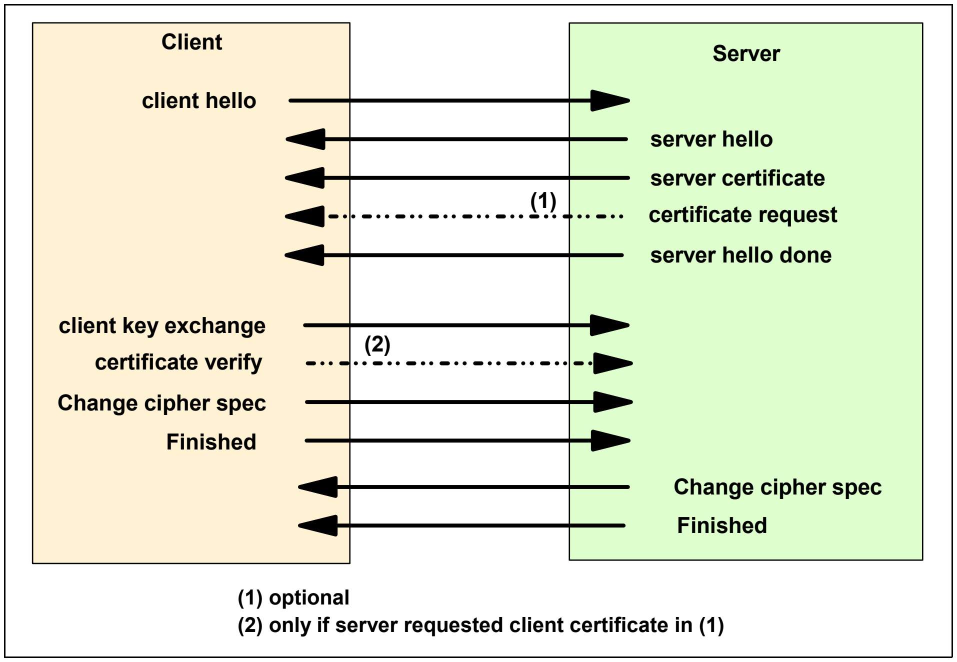


Figure 5-8 SSL handshake

The server's certificate contains the server's public key. The client uses this public key to encrypt an initial value for a secret key to be sent to the server. Once both client and server have obtained the same secret key value, the handshake ends, and they will change their encryption algorithm to the new and less intensive secret key procedure. With Version 3 of SSL, the server can also request a digital certificate from the client, to verify its identity. Note that the SSL handshaking (due to its use of secret key cryptography) is highly processor intensive.

In summary, the SSL protocol provides:

- Authentication, through its use of public/private keys and digital certificates in the handshaking phase.
- Data integrity, through its use of a MAC, generated from the secret key.
- Confidentiality, through its use of the secret key encryption for the data flowing during the SSL session.
- Non-repudiation, if you use client certificates and implement the appropriate logging. Note that your logging application on the server will have to take into account that the symmetric key only exists for one SSL session, so any logging you perform to provide non-repudiation will need to record not only the symmetric key, but also the digital certificates and ciphers you used to

generate the symmetric key. Storing this data in itself introduces some level of security risk.

Transport Layer Security (TLS)

The IETF TLS Working Group was established in 1996 to standardize a transport layer security protocol, and in 1999, RFC 2246, TLS Protocol Version 1.0 was published as a Proposed Standard.

TLS work has been based on SSL V3.0 by Netscape. However, the differences between TLS and SSL V3.0 are significant enough that TLS 1.0 and SSL V3.0 do not interoperate.

Most of the SSL-enabled products are in the process of migrating to TLS support as an alternate option to SSL. The same secure ports as for SSL are usually used.

Backward compatibility with SSL is supported:

- TLS clients who want to negotiate with SSL 3.0 servers should send client “hello” messages indicating support of SSL 3.0 and TLS 1.0. The server responds with the highest protocol it supports.
- A TLS server which agrees to interoperate with clients that support only SSL 3.0 should accept SSL 3.0 client “hello” messages and respond with an SSL 3.0 server “hello.”

5.2.3 z990, z890 and z9 hardware support for cryptography

The z990 includes both standard cryptographic hardware and optional cryptographic features, to give flexibility and growth capability. IBM has a long history of providing hardware cryptographic solutions, from the development of Data Encryption Standard (DES) in the 1970s to delivering the only integrated cryptographic hardware in a server to achieve the US Government's highest FIPS 140-2 Level 4 rating for secure cryptographic hardware.

The z990 cryptographic functions include the full range of cryptographic operations needed for e-business, e-commerce, and financial institution applications. In addition, custom cryptographic functions can be added to the set of functions that the z990 offers.

Clear and secure keys

Encryption is the process of scrambling data using a cryptographic algorithm controlled by a key. The security of this key is therefore important. If a third party gains access to this key, they can decrypt data secured by it.

IBM cryptographic hardware can use both secure keys or clear keys:

- Secure key

  A key which has itself been encrypted by another (master) key. In a z/OS environment, the secure key must first be decrypted in System z cryptographic hardware before it can be used to encrypt and decrypt data. When the key is not in use, it can be stored securely outside of the cryptographic hardware.

- Clear key

  A key which has not itself been encrypted.

Secure keys provide additional protection for the key, but comes at the cost of significant additional performance overhead.

Cryptographic hardware

There are two cryptographic devices available:

- CP Assist for Cryptographic Function (CPACF)

  The CPACF is built onto the system board, and can be enabled with a non-chargeable feature. There is a CPACF for each CP available in the machine.

  It provides synchronous cryptographic support (so when the CPACF is processing a request the general purpose CP cannot process other work). CPACF is a clear key device.

  CPACF provides clear key hardware encryption for:

  – Data Encryption Standard (DES) and Triple DES (TDES)
  – Advanced Encryption Standard (AES)
  – Secure Hash Algorithm-1 (SHA-1)

- Crypto Express2 (CEX2)

  CEX2 is a PCI card and work passed to it is performed asynchronously (meaning the general purpose CP is freed up to process other work). It supports DES, TDES, SHA-1 and RSA.

  CEX2 features two cryptographic processors:

  – A cryptographic accelerator (CEX2A)

    Used with clear keys to accelerate the handshake phase of an SSL session.

  – A cryptographic coprocessor (CEX2C)

    Encrypts and decrypts secure keys in hardware by using a master key. Also supports the SSL handshake API supported by CEX2A, but with less efficiency than CEX2A.

5.3 CICS security

We begin this section by providing a high-level overview of CICS security. Then we discuss CICS's support for cipher suites and key rings. We show you how to activate support for SSL in your CICS region, and finally discuss how SSL can be used with HTTP in a CICS environment.

5.3.1 Overview of CICS security

In a CICS environment, the assets you want to protect are the application programs, the application data, and the application output. To prevent disclosure, destruction, or corruption of these assets, you must first safeguard the CICS system components themselves.

There are two distinct areas from which exposures to the CICS system can arise. The first of these is from sources external to CICS. You can use RACF data set protection to prevent unauthorized access, from either TSO users or batch jobs, to the data sets CICS manages.

The other potential area of exposure arises from CICS users. CICS provides a variety of security and control mechanisms. These can limit the activities of a CICS user to only those functions that the user is authorized to use:

- Transaction security
  This ensures that users who attempt to run a transaction are entitled to do so.
- Resource security
  This ensures that users who use CICS resources are entitled to do so.
- Command security
  This ensures that users who use CICS system programming commands are entitled to do so.
- Surrogate security
  A surrogate user is a RACF defined user who is authorized to act on behalf of another user.

When CICS security is active, requests to attach transactions, and requests by transactions to access resources, are associated with a user ID. When a user makes a request, CICS calls the external security manager to determine if the user ID has the authority to make the request. If the user ID does not have the correct authority, CICS denies the request.

In many cases, a user is a human operator, interacting with CICS through a terminal or a workstation. However, this is not always the case: the user can also be a Web browser user or a program executing in a client system. In general, a CICS user is an entity that is identified by a *user identifier* (or *user ID*).

For a complete discussion of CICS security, refer to *CICS TS V3.1 RACF Security Guide,* SC34-6454.

5.3.2 Support for cipher suites in CICS

There are many different algorithms which can be used for encrypting data, and for computing the message authentication code. Some provide the highest levels of security, but require a large amount of computation for encryption and decryption; others are less secure, but provide rapid encryption and decryption. The length of the key used for encryption affects the level of security; the longer the key, the more secure the data.

To allow users to select the level of security that suits their needs, and to enable communication with others who may have different needs, SSL defines *cipher suites*, or sets of ciphers. When an SSL connection is established, the client and server exchange information about which cipher suites they have in common. They then communicate using the common cipher suite that offers the highest level of security. If they do not have a cipher suite in common, secure communication is not possible.

The cipher suites supported by z/OS 1.4 and CICS TS 3.1 are shown in Table 5-1 on page 114.

Table 5-1 Cipher suites supported by z/OS 1.4 and CICS TS 3.1

Cipher suite	**Encryption algorithm**	**Key length**	**MAC algorithm**
01	No encryption		MD5
02	No encryption		SHA
03	RC4	40 bits	MD5
04	RC4	128 bits	MD5
05	RC4	128 bits	SHA
06	RC2	40 bits	MD5
09	DES	56 bits	SHA
0A	Triple DES	168 bits	SHA
2F	AES	128 bits	SHA
35	AES	256 bits	SHA
The terms used in this table are: ▶ MD5 - Message Digest algorithm ▶ SHA - Secure Hash algorithm ▶ RC2 - Rivest encryption ▶ RC4 - Rivest encryption ▶ DES - Data Encryption Standard ▶ Triple DES - DES applied three times ▶ AES - Advanced Encryption Standard			

5.3.3 Building a key ring

In CICS, the required server certificate and related information about certificate authorities are held in a key ring in the RACF database. The key ring contains your system's private and public key pair, together with your server certificate and the certificates for all the certificate authorities that might have signed the certificates you received from your clients.

Before you can use SSL with CICS, you need to create a key ring which contains a private and public key pair and a server certificate.

To create a RACF key ring, you can either migrate certificates you have already built in a key data base with the *gskkyman* utility or you can generate new certificates using the RACF **RACDCERT** command.

If you are creating your own self-signed certificate, you may want to use the CICS-supplied sample REXX exec called DFH$RING. DFH$RING will create several certificates and a RACF keyring, and will then add the new certificates as well as the CA certificates to the keyring. DFH$RING is supplied in the SDFHSAMP dataset.

You can optionally designate one certificate in the key ring as the default certificate. When a client requests a certificate from CICS, the default certificate is used unless you have specified otherwise:

- For inbound HTTP and IIOP requests, specify the certificate in the CERTIFICATE parameter of the TCPIPSERVICE definition
- For outbound HTTP requests, specify the certificate in the CERTIFICATE parameter of the URIMAP definition.
- For outbound IIOP requests, specify the certificate in the CERTIFICATE parameter of the CORBASERVER definition

5.3.4 Activating SSL support

To activate SSL support in a CICS TS V3.1 and newer region, you must specify values for the following system initialization parameters:

- ENCRYPTION={WEAK | MEDIUM | STRONG}

  Specifies the cipher suites that are available for CICS to use, as shown in Table 5-2.

Table 5-2 Available cipher suites for each ENCRYPTION parameter value

ENCRYPTION parameter value	Cipher suites available
WEAK	01, 02, 03, 06
MEDIUM	01, 02, 03, 06, 09
STRONG	01, 02, 03, 04, 05, 06, 09, 0A, 2F, 35

- KEYRING=`key-database-pathname`

  Specifies the fully qualified name of the key ring within the RACF data base.

- SSLDELAY={ 600 | *number*}

  Specifies the length of time in seconds for which CICS retains session ids for secure socket connections. While CICS retains a session ID, it can continue to communicate with the client without the significant overhead of an SSL handshake.

- MAXSSLTCBS={8 | *number*}

  Specifies the number of CICS subtask TCBs (S8) that are available in the SSL pool. A maximum of 1024 can be specified. In previous releases the number of S8 TCBs directly limited the number of concurrent SSL connections CICS could process. This was because each task retained exclusive use of the S8 TCB for its lifetime. In CICS TS V3.1 the S8 TCBs are in a pool and get used for a single operation (such as a send or receive) and get returned to the pool afterwards. This dramatically increases the number of concurrent SSL connections able to be processed by CICS.

To activate SSL support for a given connection, you must set the value of the SSL parameter on the connection's TCPIPSERVICE definition to one of the following:

- YES

  If you set the value to `YES`, CICS will send a server certificate to the client.

- CLIENTAUTH

  If you set the value to `CLIENTAUTH`, CICS will send a server certificate to the client and the client must send a client certificate to CICS.

5.3.5 Transport-level security using HTTP with CICS

In this section we review how the different transport-level security mechanisms can be used to secure a CICS application solution. This section is particularly relevant when CICS Web services or CICS Web support solutions are selected.

When a CICS application is invoked using HTTP (using either CICS Web support or a SOAP/HTTP message to the CICS Web services feature), standard HTTP security mechanisms can be used to authenticate the client and to ensure message integrity and confidentiality.

Basic authentication

HTTP basic authentication is a simple challenge and response mechanism with which a server can request authentication information (a user ID and password) from a client. The client passes the authentication information to the server in an HTTP Authorization header. The authentication information is in base-64 encoding.

The AUTHENTICATE attribute on the CICS TCPIPSERVICE resource definition specifies the authentication and identification scheme to be used for inbound TCP/IP connections for the HTTP protocol. HTTP basic authentication is enabled with CICS by specifying BASIC for the AUTHENTICATE attribute.

A CICS service provider application may be protected by HTTP basic authentication. However, the HTTP basic authentication scheme can only be considered a secure means of authentication when the connection between the client and the CICS region is secure. If the connection is insecure, the scheme does not provide sufficient security to prevent unauthorized users from discovering and using the authentication information for a server. If there is a possibility of a password being intercepted, basic authentication should be used in combination with SSL, so that SSL encryption is used to protect the user ID and password information.

SSL/TLS with HTTP

SSL/TLS (Secure Sockets Layer or Transport Layer Security) is a popular way to encrypt communication between business partners over the Internet. The TLS 1.0 protocol is the latest industry standard SSL protocol and is based on SSL 3.0. The TLS 1.0 specification is documented in RFC2246 and is available on the Internet at:

http://www.ietf.org/rfc/rfc2246.txt

SSL/TLS provides for transport-level protection. It simply creates a secure connection between two nodes and encrypts all traffic flowing between the nodes.

SSL/TLS provides a straightforward way to provide confidentiality. It also includes a built-in communication integrity check. Connection layer authentication is achieved by the client always authenticating the server, and optionally being authenticated by the server, through the exchange of X.509 certificates.

The client initiates an HTTPS connection by using a URL starting with `https:` instead of `http:`. With SSL/TLS, the data flowing back and forth between client and server are encrypted using a secret key algorithm. The exchange of the secret key occurs at the start of the communication during the *SSL handshake*.

CICS support for SSL/TLS

CICS supports both the SSL 3.0 and TLS 1.0 protocols. HTTPS connections will automatically use the TLS protocol, unless the client specifically requires SSL 3.0.

A CICS service provider application may be secured using HTTPS. HTTPS has the following advantages.

- It can be used to provide a very fast and secure transport for CICS applications.

- It provides for authentication through either HTTP basic authentication or a client X.509 certificate.
- It provides integrity between the service requester and CICS by using asymmetric key cryptography to establish authenticity of server and client and to securely share a secret key.
- It provides confidentiality between the service requester and CICS through efficient shared key cryptography.
- It can be used with hardware cryptographic devices which can significantly reduce the cost of SSL handshakes. You can customize your encryption settings to use only the cipher suites that use the Integrated Cryptographic Facility (ICSF), such as the DES and SHA-1 cipher suites.
- It is mature and similarly implemented by most vendors, and therefore, is subject to few interoperability problems.

In the case of CICS Web services, a CICS service requester application may also use HTTPS to invoke a service provider application, with similar advantages to those listed above.

Enabling CICS support for SSL/TLS

To activate SSL/TLS support in a CICS TS V3.1 region, you must specify values for the following system initialization parameters:

- ENCRYPTION={<u>STRONG</u> | WEAK | MEDIUM}

  Specifies the cipher suites that CICS uses for secure TCP/IP connections. When a secure connection is established between the client and CICS, the most secure cipher suite supported by both is used.

  – Use ENCRYPTION=STRONG when you can tolerate the overhead of using high encryption if the other system requires it.

  – Use ENCRYPTION=WEAK when you want to use encryption keys up to 40 bits in length.

  – Use ENCRYPTION=MEDIUM when you want to use encryption keys up to 56 bits in length.

  For more information about cipher suites, see the *CICS RACF Security Guide*.

- KEYRING=`keyring-name`

  Specifies the name of a key ring in the RACF database that contains keys and certificates used by CICS. It must be owned by the CICS region user ID. You can create an initial key ring with the DFH$RING exec in the CICS-supplied SDFHSAMP target library.

- MAXSSLTCBS={8 | *number*}

  Specifies the maximum number of S8 TCBs that are available to CICS to process secure sockets layer connections. The S8 TCBs are created and managed in the SSL pool. An S8 TCB is only used by a task for the duration of the SSL processing.

- SSLDELAY={600 | *number*}

  Specifies the length of time in seconds for which CICS retains session IDs for secure socket connections in the SSLCACHE. Session IDs are tokens that represent a secure connection between CICS and an SSL client. The session ID is created and exchanged between the SSL client and CICS during the SSL handshake.

  While the session ID is retained by CICS within the SSLDELAY period, CICS will re-establish an SSL connection with a client without the significant overhead of another SSL handshake. The value is a number of seconds in the range 0 through 86400. The default value is 600.

  Increasing the value of the SSLDELAY parameter retains the session IDs in the cache for longer, therefore, optimizing the time it takes to perform SSL negotiations.

- SSLCACHE={CICS | SYSPLEX}

  Specifies whether CICS should use the local SSL cache in the CICS region, or share the cache across multiple CICS regions by using the coupling facility. If the cache is shared between a number of CICS regions, the throughput of SSL connections will improve.

  Caching across a sysplex can only take place when the regions accept SSL connections at the same IP address.

Activating SSL for a TCPIPSERVICE

To activate SSL support for a given connection, you must set the value of the SSL attribute on the connection's TCPIPSERVICE definition to one of the following:

- YES

  If you set the value to YES, CICS will send a server certificate to the client.

- CLIENTAUTH

  If you set the value to CLIENTAUTH, CICS will send a server certificate to the client and the client must send a client certificate to CICS.

  If you specify SSL(CLIENTAUTH) you can also specify AUTHENTICATE(CERTIFICATE) in order to map the client certificate to a RACF user ID which is then used as the identity for the client.

5.4 CICS Web services

To provide the basis for the need of Web service security, we first discuss general security and possible security exposures. Further, we provide detailed information about security concerns specific to Web services. We explain the available techniques and standards to set up a secure Web service environment.

We cover transport channel security solutions and then devote most of this section to the discussion of the WS-Security specification in CICS TS V3.1

5.4.1 Web services security exposures

Web services security is one of the most important Web services subjects. When using Web services, similar security exposures exists as for other Internet, middleware-based applications, and communications.

To explain the Web services security exposures, let us use a bank teller scenario as an example shown in Figure 5-9. The bank teller (Web service requester) connects over the Internet to the bank's data center (Web service provider). We assume there is no security applied at all, which is not realistic, but needed for the example.

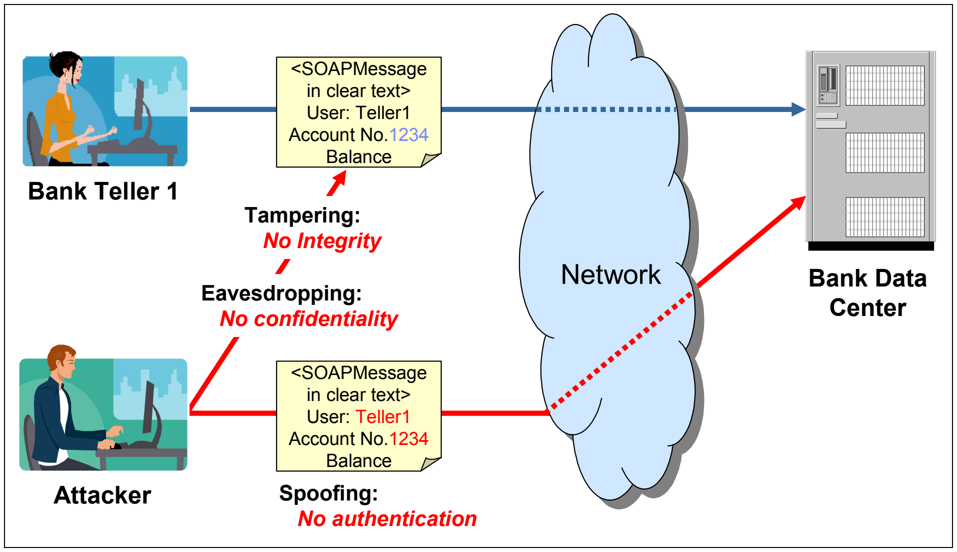

Bank Teller 1
<SOAPMessage in clear text> User: Teller1 Account No.1234 Balance
Tampering: No Integrity
Eavesdropping: No confidentiality
Network
Bank Data Center
Attacker
<SOAPMessage in clear text> User: Teller1 Account No.1234 Balance
Spoofing: No authentication

Figure 5-9 Common security exposures in a sample Web services application

The three major risk factors in this example are:

- **Spoofing—no authentication:** An attacker could send a modified SOAP message to the service provider, pretending to be a bank teller, to get confidential information, or to withdraw money from another customer's account.

  By applying authentication to the Web service, this security exposure can be eliminated.

- **Tampering—no integrity:** The SOAP message is intercepted between the Web service requester and provider. An attacker could modify the message, for example, deposit the money into another account by changing the account number. Because there is no integrity constraint, the Web service provider does not check if the message is valid and will accept the modified transaction.

  By applying an integrity mechanism to the Web service, this security exposure can be eliminated.

- **Eavesdropping—no confidentiality:** An attacker can intercept the SOAP message and read all contained information. Because the message is not encrypted, confidential customer or bank information can go the wrong people. This exposure exists because the account number and balance information is sent over the network in plain text.

  By applying a confidentiality mechanism to the Web service, this security exposure can be eliminated.

To prevent the described security exposures, the following mechanisms can be applied to secure a Web services environment (Figure 5-10 on page 122):

- Transport-level security—TLS/SSL
- Message-level security—Web services security (WS-Security)

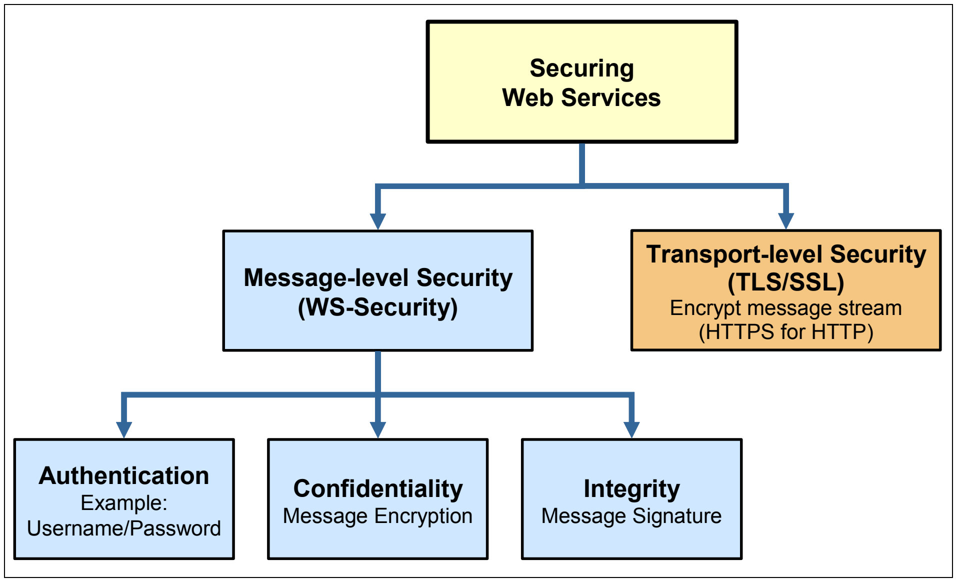

Securing Web Services
Message-level Security (WS-Security)
Transport-level Security (TLS/SSL)
Encrypt message stream (HTTPS for HTTP)
Authentication
Example: Username/Password
Confidentiality
Message Encryption
Integrity
Message Signature

Figure 5-10 Securing Web services

Depending on the required level of application security, one or more of these security mechanisms can be applied. Also depending on other non-functional requirements, a combination of message-level security and transport-level security can be implemented.

When designing a Web services security solution, keep in mind that security has an impact on the following non-functional requirements:

- **System capacity**—Any applied security mechanism has impact on system resource usage (for example, CPU and memory usage). So, when planning a Web service environment, the required *security overhead* must be considered in the system capacity and volume planning.

  The non-functional requirements, capacity and volume, cover the number of concurrent users and the number of transactions per second. This has influence on the required system infrastructure (hardware, network).

- **Performance**—Security mechanisms and functions also impact the application's response time. When defining the Web service system response time requirements, keep in mind that the response time will be affected when applying security.

  The performance requirement for a system defines the response time for a main application operation (for example, less than one second for 90% of all transactions).

Note: Applying security is not only a question of feasibility; the additional system resources and the influence on the response time also must be considered.

5.4.2 Transport level security

Web services messaging relies on two protocol layers, the transport layer and the SOAP layer. Security can be implemented within either, or both, of these layers.

In this section we review how the different transport-level security mechanisms can be used to secure a CICS Web services solution. In 5.4.6, “Comparison of transport versus SOAP message security” on page 141 we look at the main decision factors for choosing between transport-level and SOAP message security.

HTTP transport

When a CICS Web service is invoked using HTTP, standard HTTP security mechanisms can be used to authenticate the Web service requester and to ensure message integrity and confidentiality.

These mechanisms, and how they are implemented in CICS, are discussed in 5.3.5, “Transport-level security using HTTP with CICS” on page 116.

Setting the user ID on the URIMAP

It is possible to specify a user ID on the URIMAP to be used on behalf of a Web service requester. This is done by setting the USERID attribute of the URIMAP definition for a request. It specifies the 1-8 character user ID under which the Web services pipeline alias transaction is attached.

Note: If you define and install URIMAP resource definitions explicitly using CEDA, and you are using the CICS Web services assistant to create the Web services, you cannot take advantage of the dynamic installation of URIMAP resources when the PIPELINE resource is installed.

A user ID that you specify in the URIMAP definition is overridden by any user ID that is obtained directly from the client.

Important: It is important to note that if you use a URIMAP definition to set a user ID, there is no authentication of the client's identity. You should only do this when communicating with your own client system, which has already authenticated it's users, and communicates with the server in a secure environment.

Determining the user ID order of precedence when using HTTP

It is possible that for a single Web service request transported by HTTP, multiple methods for setting the user ID will be used at the same time. In this event, the following order of precedence is used for setting the user ID under which the target business logic program runs:

1. A user ID specified by a message handler, or a SOAP header processing program, that is included in the pipeline which processes the SOAP message. For example, a SOAP header processing program could extract a username from the SOAP message and specify that the CICS task should run with this user ID.
2. A user ID obtained from the Web client using basic authentication, or a user ID associated with a client certificate.
3. A user ID specified in the URIMAP definition for the request.
4. The CICS default user ID, if no other can be determined.

5.4.3 WebSphere MQ Security

To control security checking performed by WebSphere MQ, you must define *switch profiles*. When a queue manager is started (or when the WMQ REFRESH SECURITY command is issued), WebSphere MQ first checks the status of RACF and the MQADMIN class. It sets the subsystem security switch off if it discovers one of these conditions:

- RACF is inactive or not installed
- The MQADMIN class is not defined
- The MQADMIN class has not been activated

If both RACF and the MQADMIN class are active, WebSphere MQ checks the MQADMIN class to see whether any of the switch profiles have been defined. If subsystem security is not required, WebSphere MQ sets the internal subsystem security switch off, and performs no further checks. The sequence of subsystem security checks is shown in Figure 5-11 on page 125.

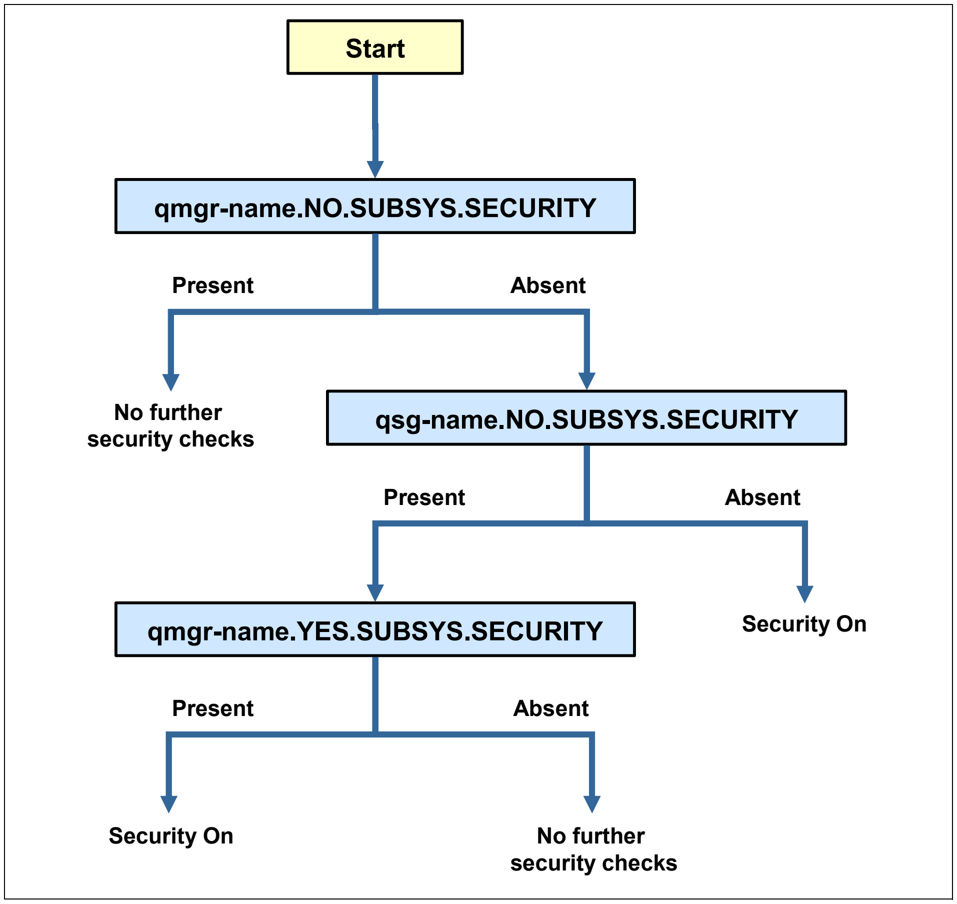

Start
qmgr-name.NO.SUBSYS.SECURITY
Present
Absent
No further security checks
qsg-name.NO.SUBSYS.SECURITY
Present
Absent
qmgr-name.YES.SUBSYS.SECURITY
Security On
Present
Absent
Security On
No further security checks

Figure 5-11 Sequence for deciding if security is on for WebSphere MQ

Switch profiles can be set at the queue manager level and at the queue-sharing group level, but the queue manager level is always checked first. If your queue manager is not a member of a queue-sharing group, then no queue-sharing group checks are made. Switch profiles are not subject to any access list checks and are merely used to indicate to WebSphere MQ whether a particular security switch is on or off. A number of switch profiles exist which can be used to control the security checking for your WebSphere MQ environment.

When using WebSphere MQ as the transport mechanism for accessing Web services in CICS, you need to consider the following points.

- The SOAP MQ inbound listener transaction (CPIL) is started by the trigger monitor using the same user ID as the trigger monitor transaction. This user

ID must have UPDATE authority to the request queue and the backout queue (if this is specified).

- If AUTH=IDENTIFY is specified in the USERDATA parameter of the WebSphere MQ PROCESS definition for CPIL, then the user ID under which CPIL runs must have surrogate authority to allow it to start transactions on behalf of the user IDs in the MQ message descriptors (MQMDs) of the messages.

More information about security for WebSphere MQ can be found in *WebSphere MQ Security*, SC34-6588 and in the redbook *WebSphere MQ Security in an Enterprise Environment*, SG24-6814.

SSL/TLS with WebSphere MQ

SSL/TLS can be used to secure SOAP messages which are transported using WebSphere MQ. WebSphere MQ supports Version 3.0 of the SSL protocol. You specify the cryptographic algorithms that are used by the SSL protocol by supplying a CipherSpec as part of the channel definition. WebSphere MQ also supports Version 1.0 of the Transport Layer Security (TLS) protocol.

See *WebSphere MQ Security*, SC34-6588 for more information about using SSL/TLS with WebSphere MQ.

Determining the user ID order of precedence when using WebSphere MQ

It is possible that for a single Web service request transported by WebSphere MQ, multiple methods for setting the user ID will be used at the same time. In this event, the following order of precedence is used for setting the user ID under which the target business logic program runs:

1. A user ID specified by a message handler, or a SOAP header processing program, that is included in the pipeline which processes the SOAP message. For example, a SOAP header processing program could extract a username from the SOAP message and specify that the CICS task should run with this user ID.
2. A user ID obtained from the MQ message descriptor.

   A message can contain message context information, such as a user ID. This information is held in the message descriptor and can be generated by the queue manager when a message is put on a queue by an application. or by the application itself. This allows the receiving application to run with the same identity as the application that put the message on the queue.
3. The CICS default user ID, if no other can be determined.

5.4.4 WS-Security

The WS-Security specification provides message-level security, which is used when building secure Web services to implement message content integrity and confidentiality.

The advantage of using WS-Security over SSL is that it can provide end-to-end message-level security. This means that the message security can be protected even if the message goes through multiple services, called intermediaries.

Additionally, WS-Security is independent of the transport layer protocol; it can be used for any Web service binding (for example, HTTP, SOAP, RMI). Using WS-Security, end-to-end security can be obtained (Figure 5-12).

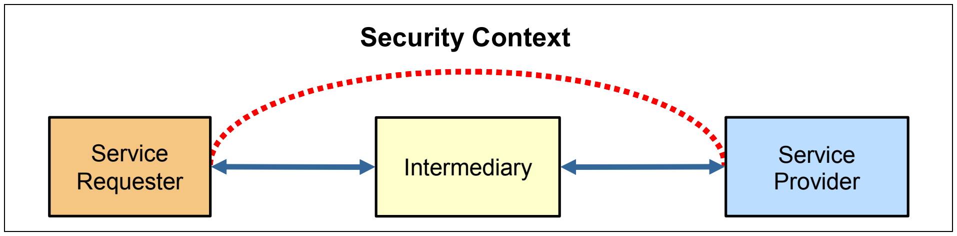


Figure 5-12 End-to-end security with message-level security

The WS-Security specification, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), is proposed by the OASIS WSS Technical Committee. This specification defines a standard set of SOAP extensions. The specification is flexible and is designed to be used as the basis for securing Web services within a wide variety of security models, including PKI, Kerberos, and SSL. It provides support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies based on XML signature and XML encryption to provide integrity and confidentiality.

The specification includes security token propagation, message integrity, and message confidentiality. However, these mechanisms by themselves do not address all the aspects of a complete security solution. Therefore, WS-Security represents only one of the layers in a complex, secure Web services solution design.

Important: With WS-Security V1.0 the wire format changed in ways that are not compatible with previous WS-Security drafts. Also, interoperabilty between implementations based on previous drafts and Version 1.0 is not possible.

The WS-Security specification defines the usage of XML signature and XML encryption:

- Message integrity is provided by XML signature in conjunction with security tokens to ensure that modifications to messages are detected. For more information, refer to:

  `http://www.w3c.org/Signature`

- Message confidentiality leverages XML encryption in conjunction with security tokens to keep portions of a SOAP message confidential. For more information, refer to:

  `http://www.w3c.org/Encryption`

Evolution of the WS-Security specification

Figure 5-13 shows the evolution of the WS-Security specification.

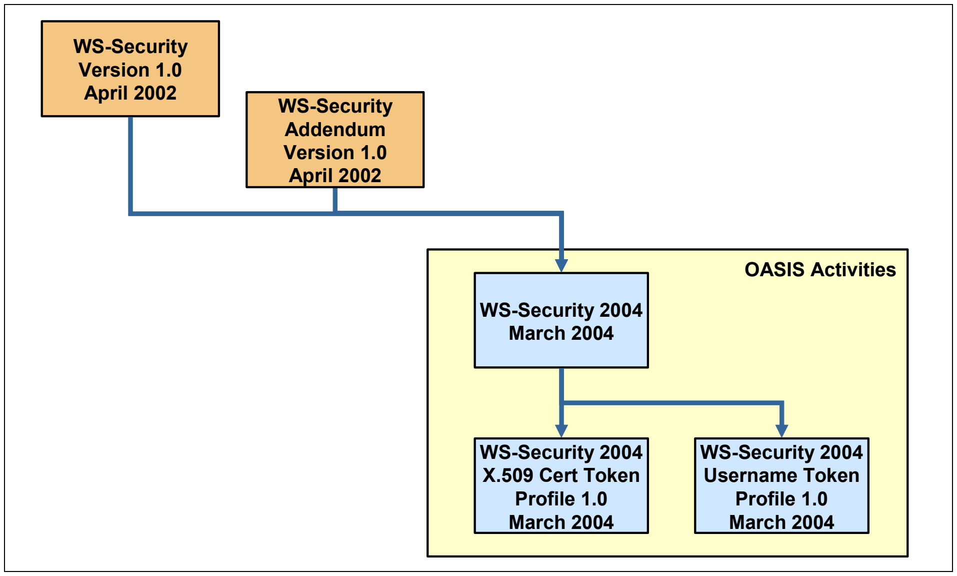

WS-Security Version 1.0 April 2002
WS-Security Addendum Version 1.0 April 2002
OASIS Activities
WS-Security 2004 March 2004
WS-Security 2004 X.509 Cert Token Profile 1.0 March 2004
WS-Security 2004 Username Token Profile 1.0 March 2004

Figure 5-13 Evolution of Web services security

The first version of the WS-Security specification was proposed by IBM, Microsoft, and VeriSign in April 2002. After the formalization of the April 2002 specifications, the specification was transferred to the OASIS consortium:

`http://www.oasis-open.org`

In OASIS activities, the core specification and many profiles that describe the use of a specific token framework in WS-Security have been discussed. The latest specification and profiles of WS-Security were proposed in March 2004 as an OASIS standard.

The latest core specification, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) was standardized in March 2004. The two profiles, Web Services Security UsernameToken Profile 1.0 and Web Services Security X.509 Certificate Token Profile 1.0, were standardized at the same time.

There are other token profiles on which, at the time of writing, OASIS is currently working:

- Web Services Security: SAML Token Profile
- Web Services Security: Rights Expression® Language (REL) Token Profile
- Web Services Security: Kerberos Token Profile
- Web Services Security: Minimalist Profile (MProf)
- Web Services Security: SOAP Message with Attachments (SwA) Profile

To read more about these standards, refer to:

- Specification: Web Services Security (WS-Security) Version 1.0 (April 2002):

  `http://www.ibm.com/developerworks/webservices/library/ws-secure/`

- Web Services Security Addendum (August 2002):

  `http://www.ibm.com/developerworks/webservices/library/ws-secureadd.html`

- Web Services Security: SOAP Message Security V1.0 (March 2004):

  `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf`

- Web Services Security: UsernameToken Profile V1.0 (March 2004):

  `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf`

- Web Services Security: X.509 Token Profile V1.0 (March 2004):

  `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf`

WS-Security road map

As previously mentioned, the WS-Security specification addresses only a subset of security services for all security aspects. A more general security model is required to cover other security aspects, such as logging and non-repudiation. The definition of those requirements is defined in a common Web services security model framework, a security white paper *Web Services Security Roadmap*, proposed by IBM and Microsoft. We describe this road map in the following section.

Web services security model framework

The Web services security model introduces a set of individual interrelated specifications to form a layering approach to security. It includes several aspects of security: identification, authentication, authorization, integrity, confidentiality, auditing, and non-repudiation. It is based on the WS-Security specification, co-developed by IBM, Microsoft, and VeriSign.

The Web services security model is schematically shown in Figure 5-14.

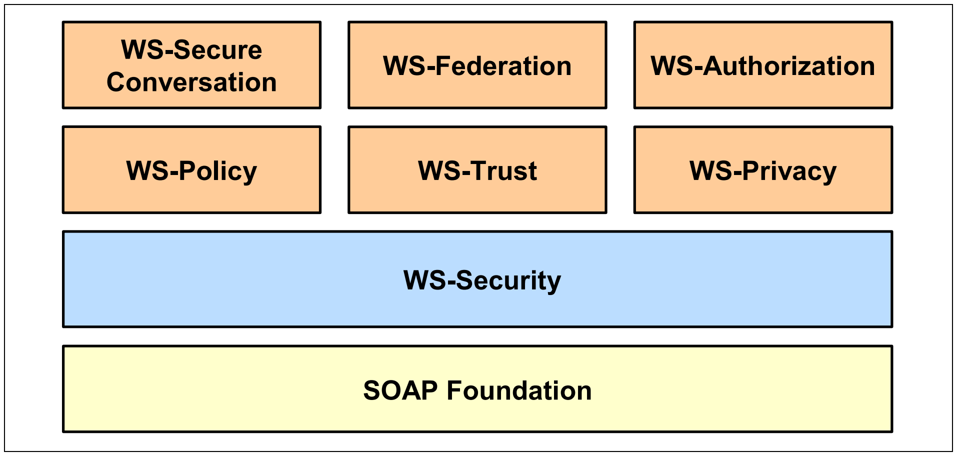


Figure 5-14 WS-Security road map

These specifications include different aspects of Web services security:

- **WS-Policy**—Describes the capabilities and constraints of the security policies on intermediaries and endpoints (for example, required security tokens, supported encryption algorithms, and privacy rules).
- **WS-Trust**—Describes a framework for trust models that enables Web services to securely interoperate, managing trusts and establishing trust relationships.
- **WS-Privacy**—Describes a model for how Web services and requesters state privacy preferences and organizational privacy practice statements.
- **WS-Federation**—Describes how to manage and broker the trust relationships in a heterogeneous federated environment, including support for federated identities.
- **WS-Authorization**—Describes how to manage authorization data and authorization policies.
- **WS-SecureConversation**—Describes how to manage and authenticate message exchanges between parties, including security context exchange and establishing and deriving session keys.

The combination of these security specifications enables many scenarios that are difficult or impossible to implement with today's more basic security mechanisms such as transport securing or XML document encryption.

Authentication

Web services security provides a general-purpose mechanism to associate security tokens with messages for single message authentication. A specific type of security token is not required by Web services security. Web services security is designed to be extensible and support multiple security token formats to accommodate a variety of authentication mechanisms. For example, a client might provide proof of identity and proof of a particular business certification.

Example 5-1 shows a sample SOAP message without applying WS-Security. The SOAP message is an Order request for our sample catalog application.

Example 5-1 SOAP message without applying WS-Security

```
<soapenv:Envelope
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header/>
  <soapenv:Body>
    <p635:DFH0XCMN
xmlns:p635="http://www.DFH0XCMN.DFH0XCP5.Request.com">
      <p635:ca_request_id>01ORDR</p635:ca_request_id>
      <p635:ca_return_code>0</p635:ca_return_code>
      <p635:ca_response_message></p635:ca_response_message>
      <p635:ca_order_request>
        <p635:ca_userid>srthstrh</p635:ca_userid>
        <p635:ca_charge_dept>hbhhhh</p635:ca_charge_dept>
        <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
        <p635:ca_quantity_req>1</p635:ca_quantity_req>
        <p635:filler1 xsi:nil="true" />
      </p635:ca_order_request>
    </p635:DFH0XCMN>
  </soapenv:Body>
</soapenv:Envelope>
```

In Example 5-2 on page 132, the SOAP message does not have any SOAP headers. We will apply WS-Security by inserting a SOAP security header.

WS-Security defines a vocabulary that can be used inside the SOAP envelope. The XML element `<wsse:Security>`[1] is the container for security-related information.

When using WS-Security for authentication, a security token is embedded in the SOAP header and is propagated from the message sender to the intended message receiver. On the receiving side, it is the responsibility of the server security handler to authenticate the security token and to set up the caller identity for the request.

In Example 5-2, we show the same SOAP message but this time with authentication. As you can see, we have user name and password information contained in the `<UsernameToken>` element.

Example 5-2 SOAP message with WS-Security

```
<soapenv:Envelope
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-ws
s-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>WEBUSER</wsse:Username>
        <wsse:Password
         Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordText">REDBOOKS</wsse:Password>
        </wsse:UsernameToken>
      </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <p635:DFHOXCMN
xmlns:p635="http://www.DFHOXCMN.DFHOXCP5.Request.com">
      <p635:ca_request_id>01ORDR</p635:ca_request_id>
      <p635:ca_return_code>0</p635:ca_return_code>
      <p635:ca_response_message></p635:ca_response_message>
      <p635:ca_order_request>
        <p635:ca_userid>srthstrh</p635:ca_userid>
        <p635:ca_charge_dept>hbhhhh</p635:ca_charge_dept>
        <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
        <p635:ca_quantity_req>1</p635:ca_quantity_req>
```

[1] wsse stands for “Web services security extension”

```
        <p635:filler1 xsi:nil="true" />
      </p635:ca_order_request>
    </p635:DFHOXCMN>
  </soapenv:Body>
</soapenv:Envelope>
```

The `<UsernameToken>` element of the SOAP message in Example 5-2 on page 132 contains credentials which can be used to authenticate the user WEBUSER.

The simplest form of security token is the UsernameToken which is used to provide a user name and password for basic authentication. A header processing program can extract a UsernameToken from a SOAP header, validate the username and password and set the user ID of the CICS task to the username passed in the header.

A signed security token is one that is cryptographically signed by a specific authority. For example, an X.509 certificate is a signed security token.

Security token usage for Web services security is defined in separate profiles such as the Username token profile and the X.509 token profile.

To read more about these security token standards, refer to:

- Web Services Security: UsernameToken Profile V1.0 (March 2004):

  `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf`

- Web Services Security: X.509 Token Profile V1.0 (March 2004):

  `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf`

Integrity

Integrity is applied to a message to ensure that no one modifies the message while it is in transit. Essentially, integrity is provided by generating an XML digital signature on a part of the SOAP message. If the message data changes, the signature would no longer be valid.

A *digital signature* is a number attached to the message. This signature establishes the following information:

- The integrity of the message:

  Is the message intact? That is, has the message been modified between the time it was digitally signed and now?

- The identity of the signer of the message:

  Is the message authentic? That is, was the message actually signed by the user who claims to have signed it?

A digital signature is created in two steps.

1. The first step distills a part of the SOAP message (for example, the body) into a large number. This number is the *digest code* or *fingerprint*.

   Several options are available for generating the digest code, for example, the MD5 message digest function and the SHA1 secure hash algorithm. Both these procedures reduce a message to a number.

   The crucial aspect of distilling the document to a number is that if the message changes, even in a trivial way, a different digest code results. When the recipient gets a message and verifies the digest code by recomputing it, any changes in the document result in a mismatch between the stated and the computed digest codes.

2. In the second step, the digest code is encrypted with the sender's private key.

This two step process creates the digital signature. The digital signature is appended to the SOAP message before being sent to the service provider.

When the service provider receives the message, it follows these steps to verify the signature:

1. Recomputes the digest code for the message.
2. Decrypts the signature by using the sender's public key. This decryption yields the original digest code for the message.

   The receiver normally obtains the sender’s public key from the sender’s X.509 certificate which is sent as a security token in the SOAP message.

3. Compares the original and recomputed digest codes. If these codes match, the message is both intact and authentic. If not, something has changed and the message is not to be trusted.

Important: The use of digital signatures has a significant impact on the system CPU resource usage.

XML encryption

Confidentiality is the process in which a SOAP message is protected so that only authorized recipients can read the SOAP message. Confidentiality is provided by encrypting the contents of the SOAP message using *XML encryption*. If the SOAP message is encrypted, only a service that knows the key for confidentiality can decrypt and read the message.

The XML encryption standard specifies a process for encrypting data and representing the result in XML. XML encryption can be used to encrypt any part of a SOAP message, normally sensitive data such as bank account numbers or user credentials. The result of encrypting data is an XML encryption element that contains or references the cipher data.

XML-Encryption was published as a W3C recommendation in December 2002. More information can be found at:

`http://www.w3.org/Encryption/2001/`

5.4.5 CICS support for WS-Security

Support for WS-Security is provided by the CICS WS-Security message handler, DFHWSSE1, which was shipped by APAR PK22736. More information regarding the signature validation and signature generation algorithms, and the decryption and encryption types that CICS supports can be found in the *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458.

DFHWSSE1 provides support for digital signing and encryption of the entire SOAP body for outbound messages. It also provides support for the body, or elements of the body and header to be encrypted or digitally signed for inbound messages.

Note: CICS TS V3.1 does not support Web Services Security for atomic transactions (WS-AT).

WS-Security in CICS can be implemented via the configuration file referenced in the CICS PIPELINE resource definition CONFIGFILE attribute. You do this by adding a WS-Security message handler to your pipeline configuration files. We discuss this in more detail in “Pipeline configuration file” on page 136.

Options for securing a SOAP message in CICS

CICS support for WS-Security provides both signing and encrypting of SOAP messages. There are several options available, and which ones you choose will depend on the level of security required for the data and the transmission path of the data. CICS specific support for the following options is described here:

- Basic authentication

  CICS supports service provider mode. The inbound SOAP message header can contain a username token (*UsernameToken*) consisting of a user name and password.

  Username tokens are not supported for outbound SOAP messages or with CICS as a service requester.

- Signing with X.509 certificates

  CICS supports both service provider and service requester modes. You can provide an X.509 certificate in the SOAP message header to sign the body of the SOAP message for authentication.

- Encrypting

  CICS supports both service provider and service requester modes. You can encrypt the SOAP message body using a symmetric-key algorithm such as Triple DES or AES.

  For inbound SOAP messages, an element in the SOAP body can be encrypted and then the entire SOAP body encrypted. If CICS receives a SOAP message with two levels of encryption, CICS will decrypt both levels automatically. This is not supported for outbound SOAP messages.

  CICS does not support inbound SOAP messages that only have an encrypted element in the message header and no encrypted elements in the SOAP body.

- Signing and encrypting

  CICS supports both service provider and service requester modes. You can sign and encrypt the SOAP message. CICS signs the SOAP message body first and then encrypts it. This provides both message integrity and confidentiality.

Pipeline configuration file

CICS TS uses a pipeline configuration file to handle Web service requests. The configuration file is an XML document and resides in the z/OS UNIX System Services hierarchical file system (HFS). You specify the name of the configuration file on the CONFIGFILE attribute of the PIPELINE definition.

To implement WS-Security in CICS TS, you need to include a `<wsse_handler>` message handler element and provide configuration information for the handler. You do this by updating the configuration file for the appropriate pipeline. DFHWSSE1 uses the configuration information specified for the `<wsse_handler>` element. The configuration file is made up of different elements. The elements that may be used for WS-Security are:

- `<wsse_handler>` - Specifies parameters used by DFHWSSE1. It can be used in both a service provider and service requester pipeline. It contains a `<dfhwsse_configuration>` element.
- `<dfhwsse_configuration>` - Specifies configuration information for DFHWSSE1. It can be used in both a service provider and service requester

pipeline. It may contain the following optional elements; each of these is discussed in more detail below:

- `<authentication>`
- `<expect_signed_body/>` - empty
- `<expect_encrypted_body/>` - empty
- `<sign_body>`
- `<encrypt_body>`

- `<authentication>` - Specifies the use of security tokens in the headers of inbound and outbound SOAP messages. It can be used in both a service provider and service requester pipeline. In a service provider pipeline, the element specifies whether CICS should use the security tokens in an inbound SOAP message to determine the user ID under which work will be processed. In a service requester pipeline, it specifies that CICS should add an X.509 certificate to the security header for outbound SOAP messages.

  The `<authentication>` element has two attributes: *trust* and *mode*. These attributes determine whether asserted identity is used and the combination of security tokens used in a SOAP message. The *trust* attribute can be set to either *none*, *basic* or *signature*. The *mode* attribute can also be set to either *none*, *basic*, or *signature*. For more information about the meaning and valid combinations of these attributes, refer to the *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458.

  Asserted identity allows a trusted user to assert, or declare, that work should run under a different identity (the *asserted identity*), without the trusted user having the credentials associated with that identity. Messages contain a *trust token* and an *identity token*. The trust token is used to check that the sender has the correct permissions to assert identities, and the identity token holds the asserted identity (user ID) under which the request is to run.

  Note: If you use asserted identity, it requires that the service provider trusts the requester to make this assertion. In CICS, the trust relationship is established with security manager surrogate definitions: the requesting identity must have the correct authority to start work on behalf of the asserted identity.

  The `<authenticate>` element can contain the following elements:

  - `<certificate_label>` - optional. Specifies the label associated with an X.509 digital certificate. Ignored in a service provider pipeline.
  - `<suppress/>` - optional. For service provider, the handler will not use any security tokens in the message to determine under which user ID to run.

  For service requester, the handler will not add any of the security tokens required for authentication to the SOAP message.

  – `<algorithm>` - Specifies the URI of the algorithm used to sign the body of the SOAP message.

- `<expect_signed_body/>` - Indicates that the `<body>` of the inbound message must be properly signed. If it is not, CICS rejects the message with a security fault.
- `<expect_encrypted_body/>` - Indicates that the `<body>` of the inbound message must be properly encrypted. If it is not, CICS rejects the message with a security fault.
- `<sign_body>` - Directs DFHWSSE1 to sign the body of outbound SOAP messages, and provides information regarding how the messages are to be signed. It can be used in both a service provider and service requester pipeline. It contains the following elements:
  - `<algorithm>` - Specifies the URI of the algorithm used to sign the body of the SOAP message.
  - `<certificate_label>` - Specifies the label associated with an X.509 digital certificate. The digital certificate should contain the private key since this was used to sign the message. The public key associated with the private key is then sent in the SOAP message, which allows the signature to be validated.
- `<encrypt_body>` - Directs DFHWSSE1 to encrypt the body of outbound SOAP messages, and provides information regarding how the messages are to be encrypted. It can be used in both a service provider and service requester pipeline. It contains the following elements:
  - `<algorithm>` - Specifies the URI identifying the algorithm used to encrypt the body of the SOAP message.
  - `<certificate_label>` - Specifies the label associated with an X.509 digital certificate. The digital certificate should contain the public key of the intended recipient of the SOAP message so that it can be decrypted with the private key when the message is received.

Example 5-3 (from the *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458) shows a completed `<wsse_handler>` with all of the optional elements present. You would add this to your configuration file for the pipeline:

Example 5-3 <wsse_handler>

```
<wsse_handler>
 <dfhwsse_configuration version="1">
   <authentication trust="signature" mode="basic">
     <certificate_label>AUTHCERT03</certificate_label>
```

```
      <suppress/>
    </authentication>
    <expect_signed_body/>
    <expect_encrypted_body/>
    <sign_body>
      <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
      <certificate_label>SIGCERT01</certificate_label>
    </sign_body>
    <encrypt_body>
      <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
      <certificate_label>ENCCERT02</certificate_label>
    </encrypt_body>
  </dfhwsse_configuration>
</wsse_handler>
```

The `<wsse_handler>` element is contained in the `<service_handler_list>` element. If you want to modify the pipeline configuration file for the CICS supplied catalog Web sample application, you would add a `<service_handler_list>` containing the `<wsse_handler>`. Example 5-4 shows the original pipeline configuration file `basicsoap11provider.xml` for the EXPIPE01 service provider pipeline.

Example 5-4 CICS supplied sample pipeline configuration file - basicsoap11provider.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd
">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Example 5-5 on page 140 shows how you would modify the pipeline configuration file to add the `<service_handler_list>` and `<wsse_handler>` elements to implement WS-Security. CICS will read the pipeline configuration file and when it finds the `<wsse_handler>` element it will load program DFHWSSE1 from library SDFHWSLD in your DFHRPL concatenation to process the security information. For more information about the elements for the pipeline configuration file, and which ones are contained by other elements (high-level structure diagrams), refer to the *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458.

Example 5-5 WS-Security <wsse_handler> element added to pipeline configuration file - basicsoap11provider.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd
">
  <service>
    <service_handler_list>
      <wsse_handler>
        <dfhwsse_configuration version="1">
          <authentication trust="signature" mode="basic">
            <certificate_label>AUTHCERT03</certificate_label>
            <suppress/>
          </authentication>
          <expect_signed_body/>
          <expect_encrypted_body/>
          <sign_body>
            <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
              <certificate_label>SIGCERT01</certificate_label>
          </sign_body>
          <encrypt_body>
            <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
              <certificate_label>ENCCERT02</certificate_label>
          </encrypt_body>
        </dfhwsse_configuration>
      </wsse_handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

If CICS is the service provider, CICS will decrypt any inbound encrypted SOAP message automatically when it processes the message, provided you have the `<wsse_handler>` element in the pipeline configuration file. The security header in the received message provides all of the information needed for CICS to decrypt it. In other words, the `<encrypt_body>` and `<sign_body>` elements do not need to be specified in the provider pipeline configuration file in order to decrypt the inbound SOAP message. But you can (and probably will want to) include the `<encrypt_body>` and/or `<sign_body>` in the provider pipeline configuration file if you want to encrypt and/or sign the reply body sent back to the requester. This is what we have shown above in Example 5-5 on page 140.

Note: For more information about the pipeline configuration file, details on the elements described above, and the message handlers, refer to the *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458.

RACF and WS-Security in CICS

You need to use RACF, or your external security manager, to create public-private key pairs and X.509 certificates for signing and encrypting outbound SOAP messages, and to authenticate and decrypt signed and encrypted inbound SOAP messages. For more information, refer to *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458 and *CICS Transaction Server for z/OS V3.1 RACF Security Guide*, SC34-6454.

5.4.6 Comparison of transport versus SOAP message security

We have shown in this section that it is possible to implement Web services security at two levels: the transport-level and the SOAP message level. If your Web services environment is simple (for example, it does not span multiple nodes) a security solution based on transport-level security alone may be all that you need. For more complex scenarios, however, it may not be enough on its own.

In this section, we provide general guidelines to help you decide what type of security solution to implement.

- You might chose to use transport-level security only to secure your CICS Web services environment when:
  - No intermediaries are used in the Web service environment.
  - The transport is only based on HTTP.
  - The Web services requester is a stand-alone program.

    WS-Security can only be applied to clients that run in a Web services environment that supports the WS-Security specification

- You might chose to use WS-Security (probably in addition to transport-level security) when:
  - Intermediaries are used.

    Security credentials which flow in the SOAP message can pass through any number of intermediaries. An intermediary can provide an authentication service to CICS, such that the intermediary server authenticates the Web service requester and flows an *asserted* identity to CICS.
  - Multiple transport protocols are used.

    WS-Security works across multiple transports and is independent of the underlying transport protocol.

Note: In order to use WS-Security in a CICS TS environment, cryptographic hardware must be enabled.

5.5 CICS Transaction Gateway

In this section, we will look into the security issues which arise when creating an SOA solution using the CICS TG. For all CICS TG topologies, there is a need to authenticate the user and to ensure that only authorized users have access to the application and its resources. We will focus here on the security mechanisms which apply when using the CICS ECI resource adapter with WebSphere Application Server.

The choice of CICS TG topology has a significant impact on the security management capabilities provided when using the JCA. These differences are outlined in 5.5.3, "Topology decisions" on page 145.

5.5.1 CICS security options

In addition to the normal CICS transaction and resource security checking, additional *intercommunication* and *surrogate* checks are performed.

Intercommunication security

Intercommunication security in CICS is concerned with incoming requests for access to CICS resources. Requests from the CICS TG can arrive via APPC, TCP/IP or EXCI connections and these are treated somewhat differently.

There are three fundamentally different intercommunication security checks that can be performed as follows:

- *Bind security*

  This verifies that the *system* wanting to connect (bind) to CICS is authorized to do so.

- *User security*

  This causes a check to be made against the *flowed* user ID when an inbound request attaches the requested transaction in CICS. The flowed user ID can be considered as the user ID which is associated with the end user. It is the user ID which is displayed in CICS as a result of a **`CEMT INQUIRE TASK`** command. The setting of the flowed user ID is controlled by the ATTACHSEC parameter on the CONNECTION definition.

- *Link security*

  This is an additional level of authorization checking that can apply to the attached transaction. A specific user ID (the *link* user) can be thought of as the user ID which is associated with the server which is passing the request to CICS. The link user ID is often defined statically on the CONNECTION definition for the remote system. This user ID, like the flowed user ID, must be authorized to access all transactions and resources invoked as a result of the request.

Surrogate user security

Surrogate user security within CICS controls the ability of the CICS region itself to work on behalf of another user. It is enabled when the CICS region is started with the `XUSER=YES` System Initialization (SIT) option. However, there is a special use of surrogate user checking on an EXCI connection between CICS and another MVS address space, for example, the CICS TG on z/OS. The EXCI surrogate user ID check controls the ability of the connecting address space to work on behalf of other user IDs flowed through the connection to CICS. The Start/Define process is surrogate (for example, using the user ID DFHSTART) in addition to the user ID DFHEXCI requests when `surrogchk=YES` is specified in DFHXCOPT.

5.5.2 JCA and security

The JCA has specific support for enabling secure access from a J2EE application to an EIS such as CICS. Both container-managed sign-on (in which the J2EE application server is responsible for flowing security context to the EIS) and component-managed sign-on (in which the application is responsible for flowing security context to the EIS) are supported.

In a managed J2EE environment, such as that provided by WebSphere Application Server, container-managed sign-on is recommended because it is good practice to separate the business logic of an application from qualities of service, such as security.

When deploying the component, the deployer must set the res-auth element in the deployment descriptor to indicate which method is being used.

Container-managed security

If you are using container-managed security, you must set the res-auth deployment descriptor element to `Container`. The application deployer must set up the authentication information (for example, the deployer sets the surname (user ID) and password to be used for the connection); in some circumstances, the container can derive the propagated identity from the currently executing Java principal. The application uses the `getConnection()` method of the connection factory and lets the application server manage the security to sign on to CICS.

Component-managed security

For component-managed security, the res-auth element needs to be set to `Application`. The application code can then supply the user ID and password when making the connection. This can be seen in the code sample in Figure 5-15. Note that even with `res-auth=Application` the application can invoke a `getConnection()` without passing a user ID and password.

```
Context ic = new InitialContext();

cnxf = (ConnectionFactory) ic.lookup("java:comp/env/eis/ECICICS1");
// create a connectionSpec to hold the security information
ECIConnectionSpec cs = new ECIConnectionSpec();
// set the user ID/password
cs.setUserName(“user_ID”);
cs.setPassword(“password”);
Connection cxn = cxnf.getConnection(cs);
Interaction ixn = cxn.createInteraction();
ECIInteractionSpec ixnSpec = new
    ECIInteractionSpec(SYNC_SEND_RECEIVE,"CICSPROG");
JavaStringRecord jsr = new JavaStringRecord();
jsr.setText("DATA1");
ixn.execute(ixnSpec, jsr, jsr);
ixn.close();
cxn.close();
```

Figure 5-15 Component managed sign-on

5.5.3 Topology decisions

We will now consider the security issues when using the different CICS TG topologies that were introduced in 3.1.2, “CICS Transaction Gateway” on page 48. In particular, we will take a look at the different mechanisms which can be used for authenticating the end user, authorizing access to resources and ensuring the confidentiality and integrity of the request while it is being transported from one physical tier to another.

CICS TG deployed on a distributed platform

In topology 1, both WebSphere Application Server and CICS TG are deployed on one of the distributed platforms, such as a Windows or UNIX platform.

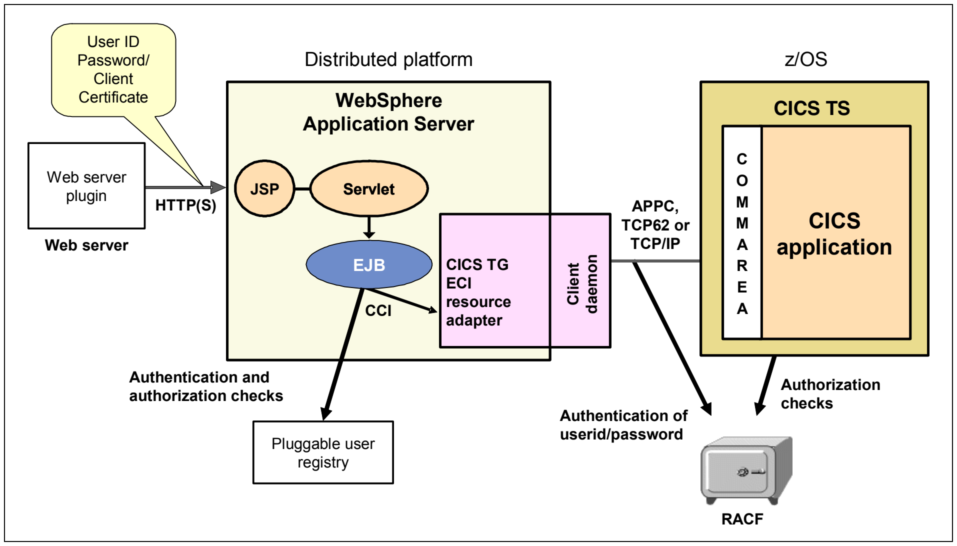


Figure 5-16 Authentication and authorization mechanisms for CICS TG Topology 1

Figure 5-16 shows when authentication and authorization checks are done for an EJB application which uses the ECI resource adapter to access a COMMAREA-based CICS application.

Authentication

An authentication mechanism in WebSphere Application Server typically collaborates closely with a pluggable user registry when performing authentication. The user registry allows you to configure different databases to store user IDs and passwords that are used for authentication and authorization. There are three options:

- Local operating system user registry

  When configured, the application server uses the operating system's users and groups for authentication.

- LDAP user registry

  In many solutions, an LDAP user registry is recommended as the best solution for large scale Web implementations. Most of the LDAP servers available on the market are well equipped with security mechanisms that can be used to securely communicate with WebSphere Application Server.

- Custom user registry

  This is the option of any custom implementation of a user registry database. An application server API provides the User Registry Java interface that can be used to write the custom registry. This interface may be used to access virtually any relational database, flat files and so on.

The authentication mechanism is responsible for creating a credential which is an application server internal representation of a successfully authenticated client user. WebSphere Application Server provides two authentication mechanisms, Lightweight Third Party Authentication (LTPA) and Simple WebSphere Authentication Mechanism (SWAM).

- LTPA is intended for use with multiple application servers and machine environments. It supports the forwarding of credentials and single-sign on.
- SWAM is intended for simple, non-distributed, application server configurations and is less secure than LTPA.

Authorization

Pluggable authorization interfaces will allow the use of different authorization mechanisms for WebSphere applications. WebSphere Application Server standard authorization mechanisms are based on the J2EE security specification and Java Authentication and Authorization Services (JAAS).

The following steps describe the main authentication and authorization events from the point when a Web browser sends the request to the WebSphere application.

1. The Web user requests a Web resource protected by WebSphere Application Server.

2. The Web server receives the request, recognizes that the requested resource is on the application server, and, using the Web server plug-in, redirects the request.
3. The Web server plug-in passes the user credentials to the Web Container of the application server, which performs user authentication against the user registry.
4. After successful authentication, the Web container does authorization checks against the user registry of the user's credentials and the security information contained in the deployment descriptor.
5. Upon subsequent requests, further authorization checks are performed either by the Web Container or the EJB Container with user credentials which are extracted from the established security context.
6. When the EJB uses the ECI resource adapter to make a request to the CICS application, the security credentials (user ID and password) need to be propagated through to CICS. This can be the responsibility of the application (component managed sign-on) or it can be the responsibility of the Web or EJB container (container managed sign-on).

   For both container and component-managed sign-on, the principal means of enabling authentication is by specifying a predefined security credential known as the JAAS authentication alias.
7. After defining the JAAS authentication alias, it can be associated with a particular connection to CICS by selecting it from a drop-down box when defining the connection factory. Another way of determining which user ID/password combination is propagated to CICS is to specify these in a ECIConnectionSpec when the connection is created (see Figure 5-15 on page 144).
8. When the request arrives in CICS, CICS verifies the user ID and password combination against the RACF database. The CICS CONNECTION or TCPIPSERVICE definition must be specified with `ATTACHSEC=VERIFY`.

   This means that there needs to be some form of mapping between the user IDs stored in the user registry used by the application server and the user IDs stored in RACF.
9. After successful authentication by CICS, the CICS application is invoked and CICS resource authorization checking is performed against the flowed user ID (the user ID which is specified in the JAAS authentication alias). The same checks are also performed against the link user ID if one is specified on the

CICS SESSIONS definition. Note that the option of using a link user ID is not available when a TCP/IP connection to CICS is used.

Note: To change the user ID and password information held in the CICS external security manager (ESM) you have to use the External Security Interface (ESI) which is based on the CICS password expiration management (PEM) function. There is no JCA resource adapter support for the ESI (only the CICS TG base classes provide support).

Data integrity and confidentiality

Figure 5-16 on page 145 shows that HTTPS can be used to secure the link between the Web server and the WebSphere Application Server. The Java Secure Socket Extension (JSEE) is the SSL implementation used by WebSphere Application Server. It is a set of Java packages that enable secure Internet communications. It implements a Java version of SSL and Transport Layer Security (TLS) protocols and includes functionality for data encryption, message integrity, server authentication, and client authentication.

The link between the application server and CICS, however, cannot be secured using SSL, that is, the user ID and password are not encrypted. The security of this link, therefore, is dependent on the security offered by the physical configuration.

Remote Gateway daemon on z/OS

In topology 2, where WebSphere Application Server is deployed on one of the distributed platforms, access to CICS is through a Gateway daemon running on z/OS, as shown in Figure 5-17.

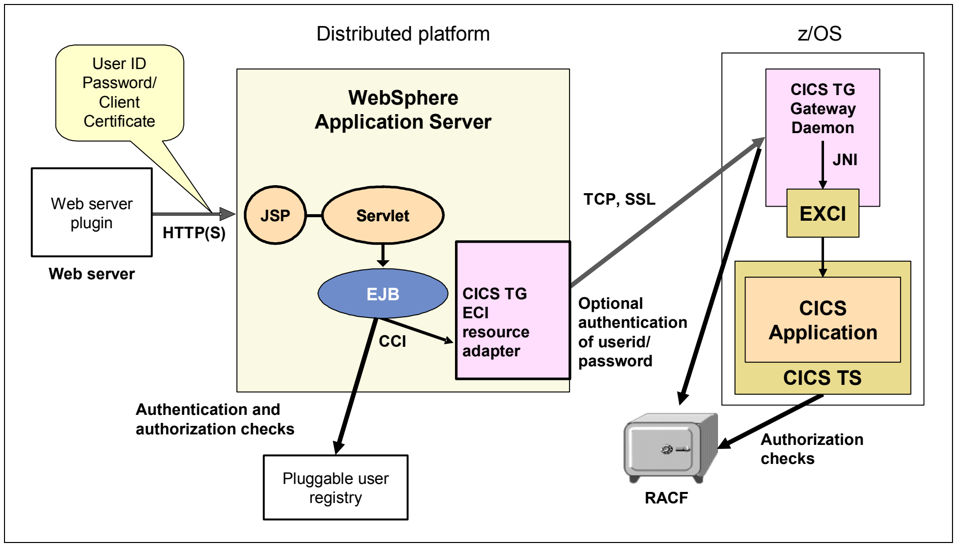


Figure 5-17 Authentication and Authorization mechanisms for CICS TG Topology 2

There are several important security differences between this topology and topology 1 shown in Figure 5-16 on page 145.

Authentication

The same application server authentication options that were discussed for topology 1 apply equally to topology 2. There is, however, an important difference in the authentication processing on z/OS.

Topology 1 requires that a user ID and password are flowed with each ECI request. This can be inconvenient in situations when authentication is being undertaken using a mechanism other than user ID and password authentication, such as client certificate authentication. In these situations, user ID authentication by CICS does not easily fit within the overall security design of the solution. Using topology 2 can help to avoid this problem because CICS TG for z/OS allows a pre-authenticated user ID to be flowed into CICS without a password. The authentication of user ID and password in this topology is optional.

To enable the CICS TG to authenticate each user ID and password flowed on an ECI request, the environment variable AUTH_USERID_PASSWORD must be set in the CICS TG environment variables. If user ID/password checking is not performed, it will probably be necessary to devise a way to establish a trust relationship between the application server and the Gateway daemon so that the application server can be trusted to flow only the user ID on the request through to CICS via the Gateway daemon. Solutions such as SSL client authentication and virtual private networks (VPN) can be used to establish such a trust relationship.

Since no password is flowed to CICS when the CICS TG on z/OS is used, the EXCI CONNECTION definition needs to be defined with `ATTACHSEC=IDENTIFY`. IDENTIFY means that CICS uses the flowed user ID in the EXCI request, but does not expect a password to be flowed with the request, as this is (optionally) checked by the CICS TG itself.

Authorization

In addition to the authorization checks described for topology 1, additional authorization checks can be used when the CICS TG is deployed on z/OS:

- MRO bind security

  MRO bind security can be used to prevent unauthorized attached MRO regions from starting transactions in a CICS region. It is implemented using DFHAPPL profiles in the FACILITY class of RACF; these profiles control logon to DFHIRP. This is used to control whether a particular CICS TG can connect(bind) to a particular CICS.

- Link security

  The link user ID which will be used for authorization checks in CICS is the user ID which is associated with the started task of the CICS TG Gateway daemon. Since this is likely to remain the same after the initial configuration, this can be pre-set on the EXCI SESSIONS definition.

- Surrogate security

  Surrogate security can be used to authorize the user ID which is associated with the CICS TG started task to switch the security context of an EXCI request to the flowed user ID. It is implemented using a profile in the SURROGAT class of RACF.

Data integrity and confidentiality

The CICS TG for z/OS provides SSL support via the Java Secure Sockets Extension (JSSE). Here are some of the features of JSSE on z/OS:

- RACF keyring support

  SSL keystores can now be stored in a RACF database.

- System z hardware cryptographic support

  This provides the ability for the CPU to offload SSL handshakes to hardware. This can substantially reduce the CPU cost of SSL handshakes and SSL data encryption.

- SSL cypher suite selection

  The SSL cypher suite in use can be configured.

When using SSL with this topology, it is particularly important to have an efficient connection-pooling mechanism because otherwise, a significant proportion of the time from making the connection to receiving the result from CICS and closing the connection, can be in the SSL handshaking. The JCA connection-pooling mechanism mitigates this overhead by allowing connections to be pooled by the WebSphere Application Server pool manager so that SSL handshaking for each request is not required.

Note: The link between CICS TG and CICS TS in this topology is MRO (Cross Memory), so cryptography is not necessary.

WebSphere Application Server and CICS TG deployed on System z

In a System z topology, WebSphere Application Server can be deployed on either a z/OS system or on a Linux operating system. The security mechanisms differ significantly between these two topologies.

WebSphere Application Server and CICS TG on z/OS

This topology has significant security advantages because:

- The application server and CICS are able to share the same RACF user registry for authentication and authorization checks (see Figure 5-18).
- The application server and CICS are installed in the same MVS LPAR and, therefore, the connection between the servers is inherently more secure.
- *Thread identity support* is enabled (see “Authorization” on page 152).

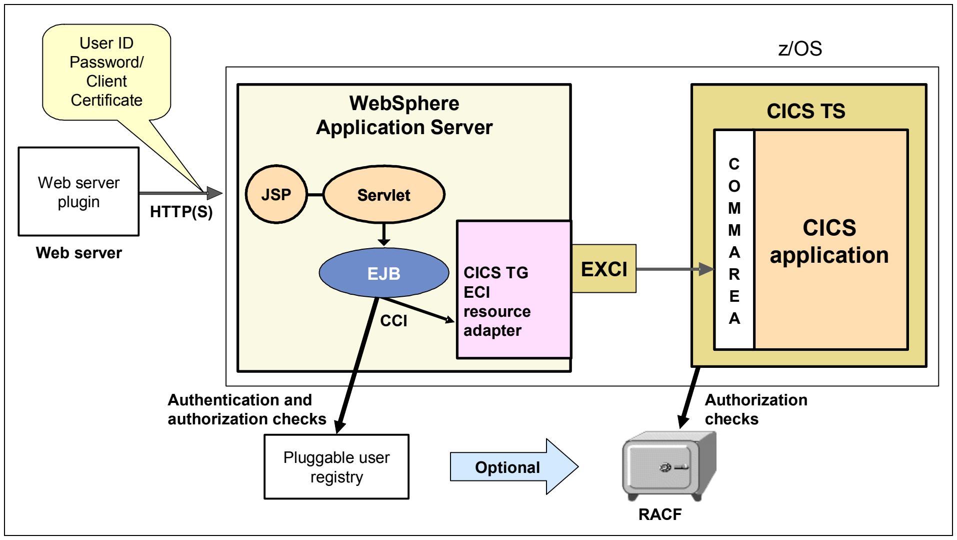


Figure 5-18 Authentication and Authorization mechanisms for CICS TG Topology 3a

Let us look at the authentication, authorization, and data integrity aspects of this topology.

- Authentication

  When WebSphere Application Server is running on z/OS, the same options for a pluggable user registry apply, that is, a local operating system registry, an LDAP registry or a custom user registry. When the application server is configured to use a local operating system registry (such as RACF), the security identity established after authentication in WebSphere Application Server will be a RACF user ID if you use basic authentication (or form-based login). If an SSL client certificate is used to authenticate, you can configure RACF to map that certificate to a RACF user ID. This means that the Java thread in WebSphere Application Server on z/OS will have a security identity that is a RACF user ID.

- Authorization

  When using container-managed sign-on, a z/OS system-specific functionality known as *thread identity support* is provided by WebSphere Application Server for z/OS. This support is unique to WebSphere Application Server for z/OS and allows the application server to automatically pass the user ID of the thread (the caller's identity) to CICS when using the ECI resource adapter.

  Thread identity support is enabled when:

  – WebSphere Global security is enabled and RACF is being used as the local operating system registry.

- A local connection is being used between the application server and CICS.
- Container-managed security is being used (the res-auth deployment descriptor is set to `Container`).
- The connection factory does not specify a JAAS Authentication Alias.

Most z/OS customers will want to use this feature because it enables the application server to behave in a way that traditional z/OS address spaces behave, that is, once you have authenticated, your user ID flows with any work you do within the z/OS system.

The CICS authorization mechanisms apply to this topology as follows:

- MRO bind security

  MRO bind security can be used to establish a trust relationship between the application server and CICS servers. It is implemented using DFHAPPL profiles in the RACF FACILITY class that control logon to DFHIRP.

- Link security

  The link user ID which will be used for authorization checks in CICS is the user ID which is associated with the started task of the WebSphere Application Server J2EE servant region. Since this is likely to remain the same after the initial configuration, this can be pre-set in the EXCI SESSIONS definition.

- Surrogate security

  Surrogate security checks can be enabled to authorize the user ID associated with the J2EE servant region to flow a specific user ID (or one of a generic set of user IDs) to CICS.

WebSphere Application Server and CICS TG on Linux on System z

Topology 3b is where WebSphere Application Server is deployed within Linux on System z. The security options for this topology are almost identical to those described for topology 1. One notable exception is that in this topology HiperSockets can be used to connect from the CICS TG running on Linux to the CICS server. It is unlikely, therefore, that you will need to encrypt the data that is passed on this link.

5.5.4 Security coordination between WebSphere and CICS

In this section, we summarize the key security points you need to understand when using the JCA to connect to CICS from a J2EE application running in WebSphere Application Server:

- You have a choice of using container-managed or component-managed sign-on. Container-managed sign-on is the recommended approach.
- When using the ECI resource adapter, the support provided by the JCA security-management contract is dependent on the CICS TG topology being used.
- Defining a JAAS Authentication Alias is the principal way of specifying the user ID and password to be flowed to CICS when using topologies 1 and 2 (WebSphere Application Server installed on a distributed platform).
- When using the ECI resource adapter with topology 1 (CICS TG installed on a distributed platform) it is a requirement that a user ID and password is flowed with each ECI request. By contrast, you have the option of flowing only the user ID when using topology 2 (CICS TG installed on z/OS).
- When using the ECI resource adapter with topology 3 (WebSphere Application Server installed on z/OS) the application server is capable of automatically passing the caller's authenticated user ID to CICS.
- A number of CICS security mechanisms, including bind, link and surrogate security can be used to provide additional security checks.
- SSL can be used in most configurations if there is a need to encrypt the data which is passed between the different physical tiers in the topology.

5.6 CICS Web support

In this section, we look at the two main security issues that you will face when accessing your CICS applications using CICS Web support; authenticating and authorizing the user, and using SSL support to provide encryption of the data being sent and received.

A summary of where you need to consider security options is shown in Figure 5-19 on page 155.

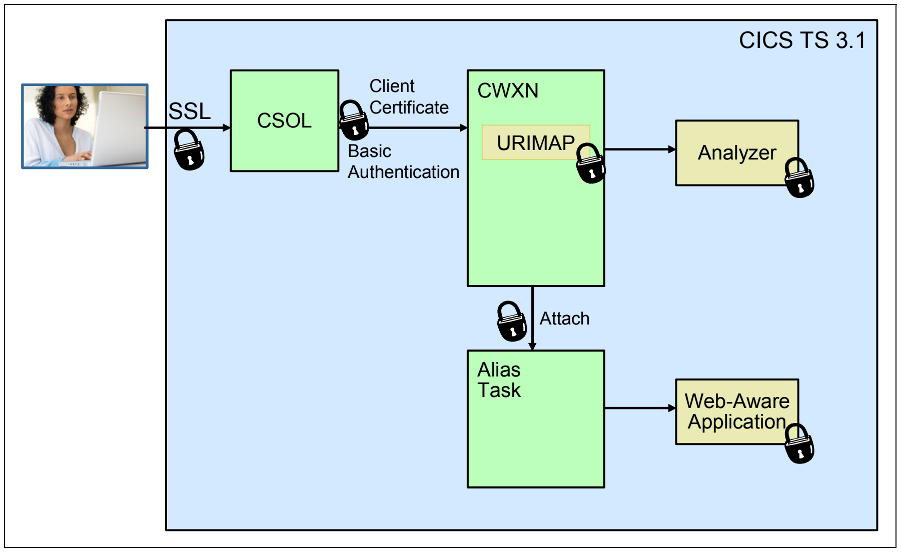


Figure 5-19 Security with CICS Web Support

5.6.1 Authenticating and authorizing the user

When using a direct connection in to CICS Web support the user ID that the alias task, and consequently the required business logic, will run under can be determined in one of the following ways:

- The user ID is associated with an SSL certificate supplied by the client.
- The user ID and password are requested using HTTP Basic Authentication.
- A URIMAP specifies the user ID to use with the request.
- The analyzer program determines the user ID to be used.

Using an SSL client certificate

Using SSL with CICS Web support enables the client to supply a digital certificate in order to identify itself. If you want to use client certificates, specify one of the following values for the AUTHENTICATE parameter on the TCPIPSERVICE definition that was used to define the relevant CICS TCP/IP listener:

- CERTIFICATE

  The SSL client certificate is used to authenticate and identify the client. The client must send a valid certificate which is already registered to the security manager, and associated with a user ID. If a valid certificate is not received, or the certificate is not associated with a user ID, the connection is rejected.

When the end user has been successfully authenticated, the user ID associated with the certificate identifies the client.

- AUTOREGISTER

  The SSL client certificate is used to authenticate the client.

  - If the client sends a valid certificate that is already registered to the security manager, and associated with a user ID, then that user ID identifies the client.
  - If the client sends a valid certificate that is not registered to the security manager, then HTTP Basic Authentication is used to obtain a user ID and password from the client. Provided that the password is valid, CICS registers the certificate with the security manager, and associates it with the user ID. The user ID then identifies the client.

- AUTOMATIC

  This combines the AUTOREGISTER and BASIC functions.

  - If the client sends a certificate that is already registered to the security manager, and associated with a user ID, then that user ID identifies the client.
  - If the client sends a certificate that is not registered to the security manager, then HTTP Basic Authentication is used to obtain a user ID and password from the client. Provided that the password is valid, CICS registers the certificate with the security manager, and associates it with the user ID. The user ID then identifies the client.
  - If the client does not send a certificate, then HTTP Basic Authentication is used to obtain a user ID and password from the user. When the user has been successfully authenticated, the user ID supplied identifies the client.

Using HTTP Basic Authentication

Basic Authentication is an HTTP feature whereby the user ID and password are flowed over the network in a scrambled format that uses the Base64 encoding scheme; it is, however, easily unscrambled. If you want to use HTTP Basic Authentication, specify BASIC as the value of the AUTHENTICATE parameter of the TCPIPSERVICE definition.

Using a URIMAP

The USERID attribute of the URIMAP resource specifies the user ID of the attached alias task. This user ID will apply to all inbound requests that match the SCHEME, HOST and PATH specified in the URIMAP. This would only be used when the specific user does not need to be authenticated, but a user ID other than the CICS default user ID is required to authorize access to the associated resources.

An example of when this might be used is if the CICS default user ID is not authorized to run any Web alias transactions. If the real user is to be authenticated using an HTML forms based dialog then an alias transaction will be required. A special user ID can be setup to allow a specific alias transaction and associated programs to be run before the real user ID is established.

Using the analyzer

In many cases an analyzer is not required. For example, it is not required if a suitable URIMAP definition is defined.

However when an analyzer is required, it can use any information in the incoming HTTP request or obtained using the **`EXEC CICS WEB`** and **`TCPIP API`** to determine what user ID should be used for the alias task.

The analyzer may also determine that the user must supply their user ID and password. This could be done via HTTP Basic Authentication or a HTML forms based dialog. CICS ships sample analyzer programs for both alternatives, but you might like to write your own analyzer, as discussed in Chapter 5 of the redbook *Securing Web Access to CICS*, SG24-5756. Another helpful resource may be the CICS SupportPac CA8D, containing a more elaborate code skeleton. It is available at:

`http://www.ibm.com/software/ts/cics/txppacs`

Note: You should write or customize an analyzer program to authenticate the user only if the other methods of authentication are unsuitable. The analyzer program can perform other functions though, and may still be required even if not used for authentication. For example, it is a good place to produce an audit of Web access to your CICS region.

5.6.2 SSL support

In order for the CICS server to use SSL, you first need to create a server certificate. If your Web application is on an intranet or extranet, it may suffice to create a self-signed certificate. For an Internet application, you should go through an external certification process with a certificate authority (CA).

Next, you need to install the server certificate in a RACF keyring and specify the name of the keyring in the KEYRING system initialization parameter. If you also specify the certificate name in the CERTIFICATE parameter of the TCPIPSERVICE definition, you can use a different certificate in the keyring for each TCPIPSERVICE installed in CICS. If you do not specify the certificate name in the TCPIPSERVICE definition, CICS will use the default certificate in the keyring.

Important: You must specify TCPIP=YES and the KEYRING system initialization parameters for SSL to be available in your CICS region.

To activate SSL support for an *incoming* HTTP request, specify one of the following for the value of the SSL parameter of the TCPIPSERVICE definition:

- YES

  If you set the value to `YES`, CICS will send a server certificate to the client.

- CLIENTAUTH

  If you set the value to `CLIENTAUTH`, CICS will send a server certificate to the client and the client must send a client certificate to CICS.

If you are using the *outbound* HTTP support and the remote server requests a client certificate then the default certificate from the RACF keyring will be sent unless a URIMAP is being used on the EXEC CICS WEB OPEN command. In which case the certificate named in the URIMAP will be sent.

Ciphers

There are many different algorithms that can be used for encrypting data, and for computing the message authentication code.Some provide the highest levels of security, but require a large amount of computation for encryption and decryption; others are less secure, but provide rapid encryption and decryption. The length of the key used for encryption affects the level of security - the longer the key, the more secure the data.

The individual ciphers that can be used by CICS are dependant on the ENCRYPTION system initialization parameter. Table 5-3 shows the maximal set of ciphers available with the different ENCRYPTION settings.

Table 5-3 Available ciphers

ENCRYPTION parameter	**Available ciphers**
ENCRYPTION=WEAK	03060102
ENCRYPTION=MEDIUM	0903060102

ENCRYPTION parameter	Available ciphers
ENCRYPTION=STRONG	0504352F0A0903060201

For inbound requests the ciphers to be used are specified in the TCPIPSERVICE resource definition. For outbound requests the ciphers are specified in the URIMAP resource definition. The list of ciphers is ordered from left to right with each cipher being a 2 hexadecimal digit value. You can specify exactly which ciphers are used and the order of preference by reordering the list or removing ciphers from the list. You cannot add additional ciphers to the list.

When specified in the TCPIPSERVICE definition the final CIPHERS list determines the setting of the PRIVACY attribute. Table 5-4 shows the relationship between the CIPHERS and PRIVACY attributes.

Table 5-4 PRIVACY setting based on CIPHERS value

CIPHERS	PRIVACY
Contains ONLY 01 and 02	NOTSUPPORTED
Does NOT contain 01 or 02	REQUIRED
Any other set of ciphers	SUPPORTED

The available ciphers also depends on the operating system. The operating system must support all the ciphers that you have specified CICS to use. Table 5-1 on page 114 lists the cipher suites that are available with z/OS 1.4.

Important: If cipher suite 01 or 02 gets used then no encryption occurs. Data will be transmitted in the clear. This may not be the desired effect when using SSL support in CICS. Ensure 01 and 02 get removed from the list of ciphers if encryption is required.

5.6.3 Design issues

In summary, these are the key points to consider when designing a secure solution with a CWS direct connection.

- A De-Militarized Zone (DMZ) is a key part of a secure Web-enablement strategy. With CICS Web support you can participate in this architecture by installing a dedicated CICS region, the *listener region,* in a separate Web LPAR. Rather like a terminal-owning region (TOR), the listener region just serves to handle incoming requests from the Web. It also acts as a protocol switch converting TCP/IP requests into the internal CICS protocols.

After the listener region has authenticated the user ID, a CICS user program can be invoked in another CICS region using a distributed program link (DPL) call, as shown in Figure 5-20. This program can possibly be on a different LPAR within the same sysplex if the DPL call utilizes XCF communications.

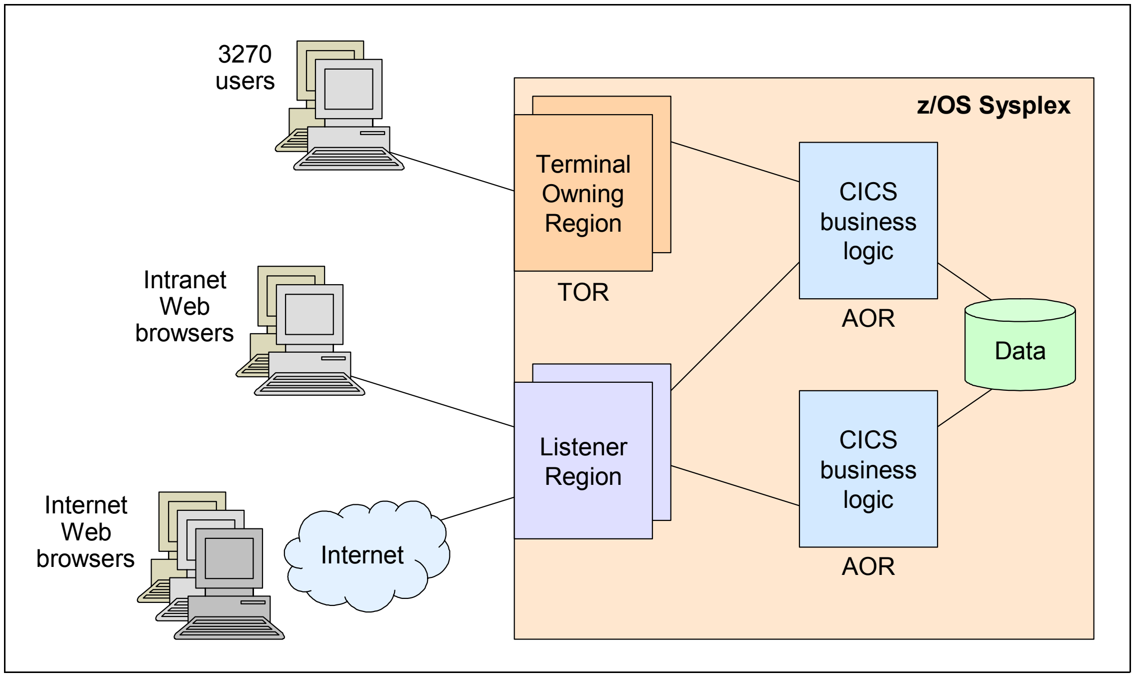


Figure 5-20 Sample CICS Web Support listener region configuration

Note: A separate listener region is only possible if the business logic is a COMMAREA based application. Applications using the `EXEC CICS WEB API` cannot be run remotely.

- Another argument for setting up a listener region may be to have link security between the listener region and the application-owning region (AOR) where the business logic runs. The link user ID will carry the maximum level of authorization for any transaction in the AOR. All authorizations in the AOR are checked against both the link user ID and the user ID flowed with the DPL request, and both authorizations are required for the business logic to be called.

- You should also consider using URIMAPs, or if just using the default analyzer, disabling the program auto-install. This is because the URL format expected by the default analyzer allows any CICS program to be invoked. Obviously, you are unlikely to want just any Web browser to invoke any program that can be found in your CICS load libraries.

6 Transactional scope

CICS is the predominant transaction processing system in use in today's IT systems. Therefore, any service requester which needs to connect to and utilize information from within CICS will most likely need to consider the transactional scope of such calls.

When considering the transactional scope of your new service requester, you will need to think about the following key issues:

- Do any of the programs you invoke within CICS perform work on recoverable CICS resources, such as VSAM files or DB2 tables?
- Do you need to ensure that multiple calls to the same CICS program are handled as a single recoverable unit?
- Do you need to coordinate recoverable work within CICS with work that is performed on other recoverable resources outside of CICS?
- Do you require CICS to control the runtime infrastructure to manage your transactional integrity or will your applications handle it themselves?

First, however, let us take a step back and look at some of the fundamentals of the transactional infrastructure provided by CICS and the SOA connectors before looking at the transactional support provided in each scenario.

6.1 Transactions: What are they?

In the J2EE world, a transaction is a unit of activity within which multiple updates to recoverable resources can be made atomic (that is, an indivisible unit-of-work), such that all or none of the updates are made permanent. However, this is not the same as the traditional usage of the term *transaction* in CICS, so let’s first have a look at CICS transactions and associated terminology.

6.1.1 CICS transactions, tasks and syncpoints

In this chapter, we will refer to a *CICS-transaction* as the work initiated in a CICS region, and which runs as a CICS *task* under a four-character transaction ID (tranid). These tranids are static definitions which specify the initial program to be loaded and the properties of the CICS transaction under which the program will run. They are defined in TRANSACTION definitions within the CICS resource definition online (RDO) database. At task initiation, CICS implicitly starts a unit-of-work (UOW) for all CICS transactions; this is usually the initial boundary of the transactional work to be undertaken. All updates to recoverable resources or requests to other transactional systems are now part of this unit-of-work, until either a synchronization point (sync point) is reached within the CICS program, or the CICS-transaction finishes and the task terminates (Figure 6-1).

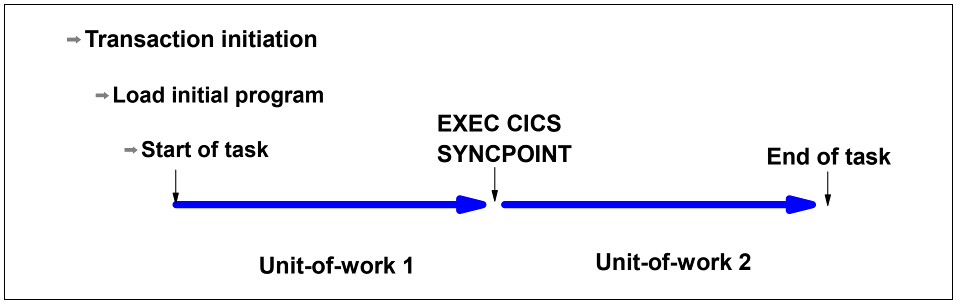


Figure 6-1 CICS synchronization points

In certain circumstances, such as though an inter-system distributed program link (DPL) request is used, the CICS-transaction that is linked to can be co-ordinated by a remote CICS system; in this instance the called mirror task in CICS remains suspended until the end of the transaction. This is referred to as a *long-running mirror task* (Figure 6-2).

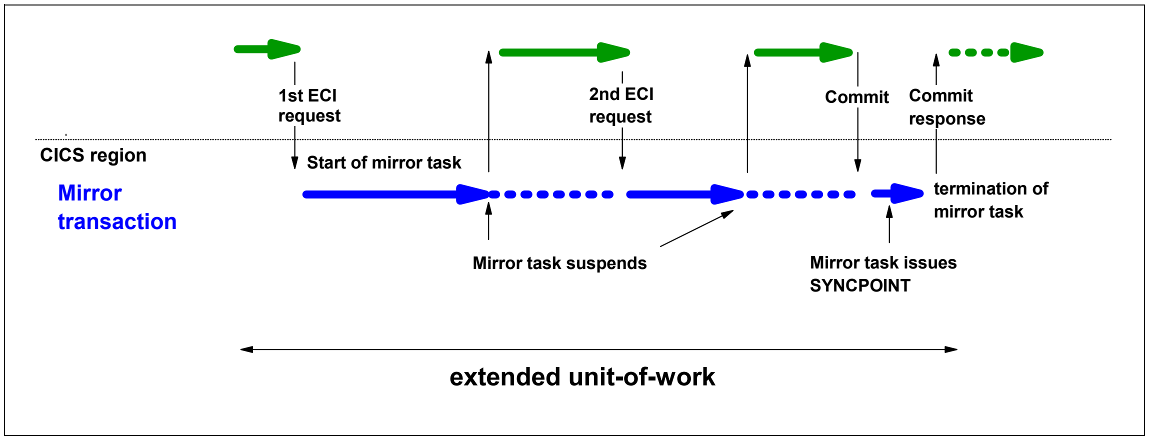


Figure 6-2 Link with long-running mirror task

Additionally, the converse situation is also possible; this is where the invoked CICS transaction runs in a separate transactional context to that of the invoking application. This is referred to as *running with sync-on-return*, which refers to the fact that the controlling mirror transaction in CICS issues a sync point on returning control to the calling application (see Figure 6-3). The use of a sync-on-return type link also allows the called CICS program to issue **`EXEC CICS SYNCPOINT`** commands, since it is not subordinate to another transaction manager.

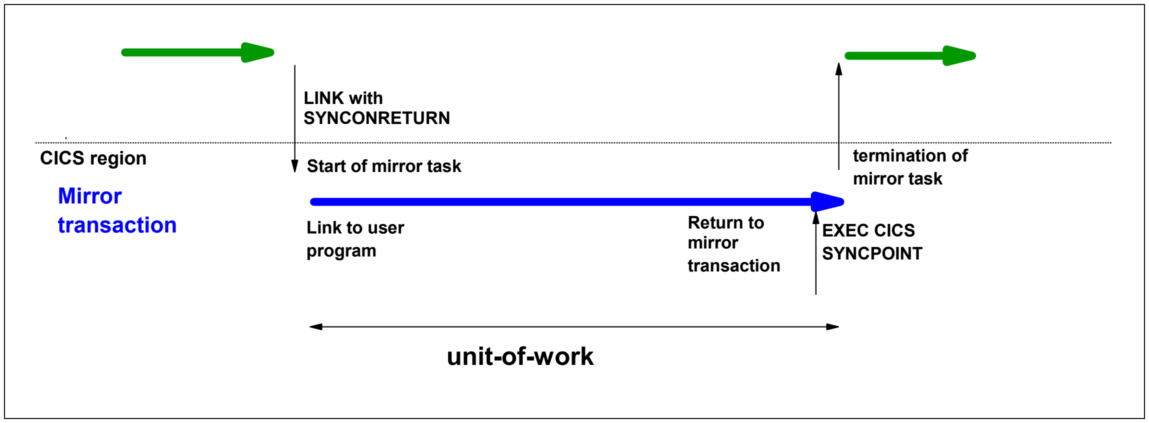


Figure 6-3 Link with SYNCONRETURN

6.2 Transactional building blocks

Within a distributed transactional system, each distributed system is either referred to as a *resource manager* or a *transaction manager*. The *transaction* manager controls the outcome of the transaction and is responsible for the recovery of its resources; it has to implement a recoverable logging mechanism in order to be able to coordinate multiple resource managers. The *resource* managers control access to recoverable resources, and as such have to implement the necessary network flows and logging procedures to provide transactional coordination.

Transactional standards

In order to enable distributed systems to send and receive transactional requests the partners must send and receive the requests using a common standard. It is unlikely that you will have the desire or need to understand the standards in detail but we will give a brief summary here to help you understand upon what standards each of the technologies is built.

CICS CICS itself has its own private transactional intersystem communication (ISC) protocols, which are predominantly based upon the LU6.2 SNA formats and protocols. The ISC protocols allow multiple CICS systems across different MVS systems or different platforms (such as Windows or UNIX systems) to support extended units of work between CICS applications in these disparate systems. For more information, refer to the *CICS Intercommunication Guide*, SC34-6243.

JTA The Java Transaction API (JTA) is a specification of the interfaces between a transaction manager and the other parties involved in a distributed transaction processing system: the application programs, the resource managers, and the application server. The EJB architecture requires that the EJB container support the JTA.

JTS The Java Transaction Service (JTS) API is a Java binding of the CORBA Object Transaction Service (OTS) 1.1 specification. JTS provides transaction interoperability using the standard Internet Inter-ORB Protocol (IIOP) for transaction propagation between servers.

JCA The J2EE Connector Architecture (JCA) is part of the J2EE standard. The JCA specifies the system contracts for connection management, transaction management, and security management, that exist between the application server and enterprise information system (EIS). For further details, refer to “JCA” on page 188.

XA A specification for distributed transaction processing, allowing two-phase commit processing.

Two-phase commit

An essential part of all transactional standards is the two-phase commit process. This is an architected set of flows that transaction managers use to ensure all resource managers in a transaction can be reliably coordinated, irrespective of any failure. It is implemented by all transactional protocols and the fundamental concepts are essentially the same. The following description summarizes the flows according to the XA specification (see Figure 6-4 on page 168); other protocols such as CICS or LU6.2 may use different terminology and variants on the flows. (For further details on the CICS sync point flows refer to Chapter 2 “Recovery and restart in interconnected systems” of the *CICS Intercommunication Guide*, SC34-6243.)

In the first phase (or stage 1), the transaction manager asks all the resource managers to prepare to commit recoverable resources (prepare), each resource manager can vote either positively (*prepared*) or negatively (*rolled-back*). If a resource manager is to reply positively, it records stably the information it needs to do so and replies *prepared*, and is then obliged to follow the eventual outcome of the transaction as determined at the next stage. The resource manager is now described as *in-doubt,* since it has delegated eventual transaction control to the transaction manager.

In stage 2, providing all the resource managers voted positively, the transaction manager replies to each resource manager with a commit flow. Upon receipt of the commit flow, the resource manager finalizes updates to recoverable resources, and releases any locks held on the resources. The resource manager then responds with a final *committed* flow, which indicates to the transaction manager that it is no longer in-doubt. If the final committed flow is not received by the resource manager, the transaction manager must assume the commit was also not received by the resource manager, and must re-transmit the commit.

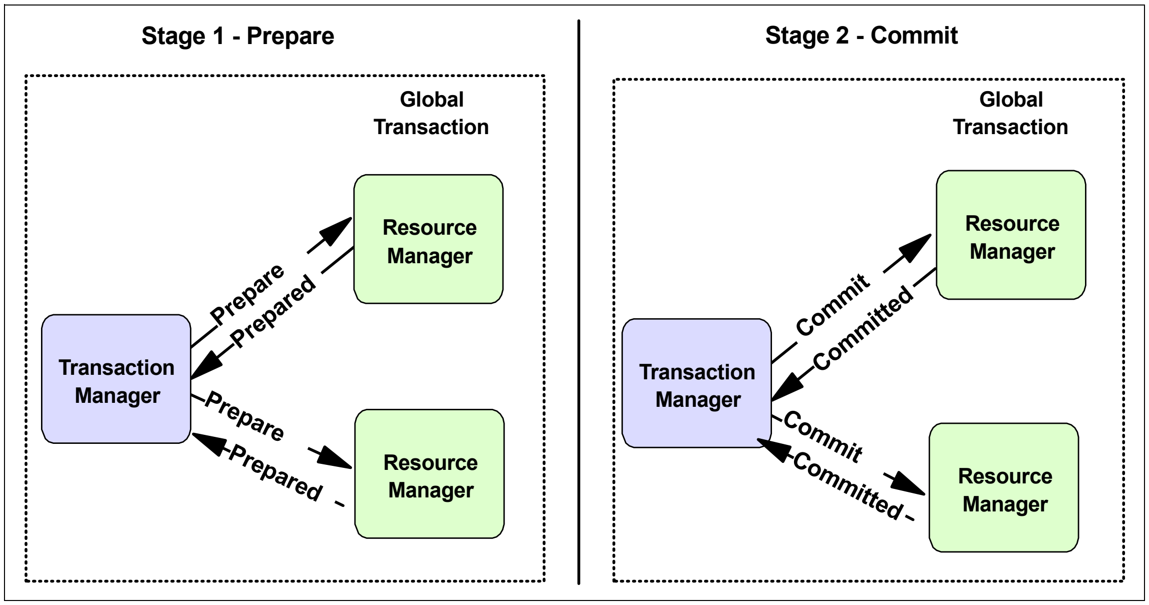


Figure 6-4 Two-phase commit

Although the two-phase commit process is usually a prerequisite to distributed transactional support, there are certain instances where a single-phase commit process can be sufficient. This is referred to as last resource optimization, and is implemented by a variety of transaction managers. It essentially allows the commit decision to be delegated to the one-phase commit resource, allowing the one-phase commit to participate in a global transaction with any number of two-phase commit capable resources (Figure 6-5 on page 169).

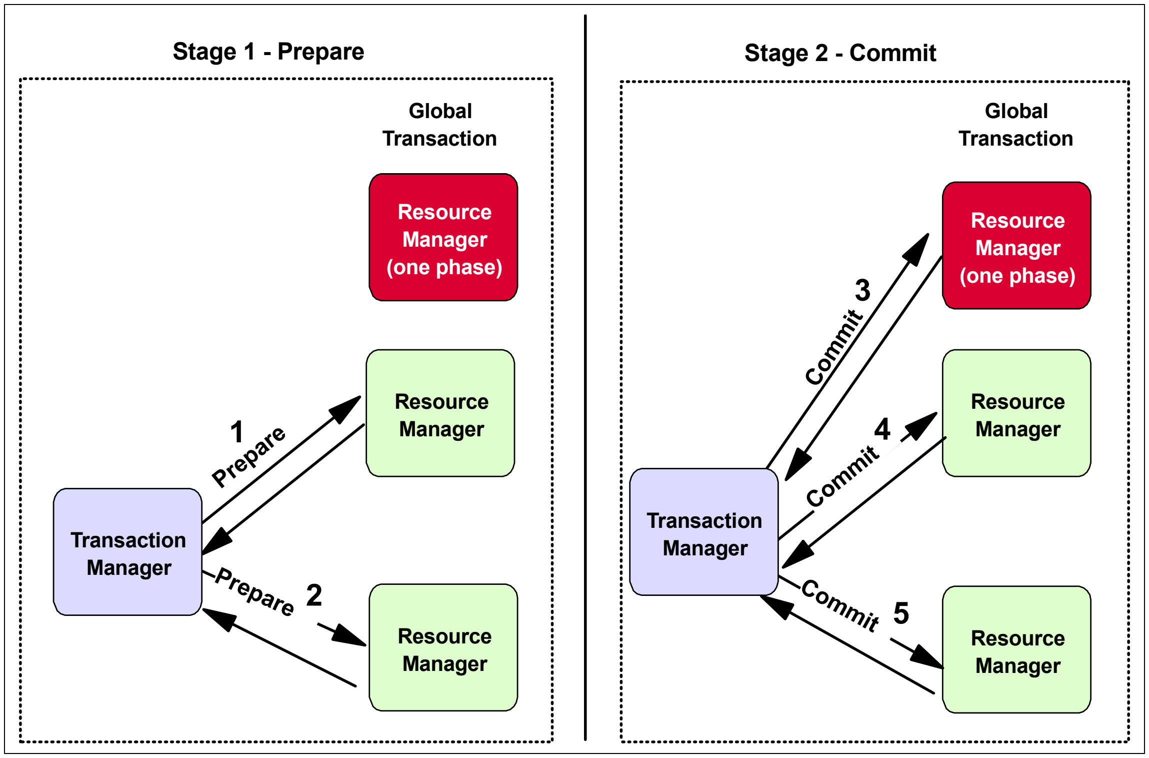


Figure 6-5 Last resource optimization

At transaction commit, the application server first prepares the two-phase commit resource managers and, if this is successful, the one-phase commit-resource is then called to commit. The two-phase commit resources are then committed or rolled back depending on the response of the one-phase commit resource, effectively delegating transaction coordination to the one-phase commit resource.

Unlike a two-phase commit resource, there is no recovery from a communication failure with a one-phase commit resource. Such a communication failure during commit of the one-phase commit resource introduces the risk of a mixed outcome to the transaction. The two-phase commit resources are rolled back, but the outcome of the one-phase commit resource is unknown; it could have committed or rolled back. Applications must therefore be configured to accept the additional risk of such heuristic outcomes.

Last resource optimization is implemented within WebSphere Application Server as *Last Participant Support*, and within CICS and APPC flows as *last agent optimization*. However, applications that exploit last participant support are subject to an increased risk of a mixed outcome in a global transaction if the one-phase commit resource fails during the commit processing.

Compensating transactions

A compensating transaction is a group of operations that undoes the effects of a previously committed transaction. There are many circumstances where compensating transactions may play a role:

- They may be used to restore consistency after an unrecoverable failure that prevented a distributed transaction from normal completion.
- A resource manager may have been left in-doubt because it did not receive a reply in the second stage of the two-phase commit process. If so it may have taken a heuristic decision about the probable outcome of the transaction and so some participants might have committed while others did not.
- When one of the global transaction participants is a non-transactional resource manager. If such a transaction performs a rollback, its non-transactional participant may need to be rolled back via the compensating transaction.
- In certain business transaction scenarios, especially ones that span several systems, maintaining long-lived locks and restricting data access for extended periods of time might not be acceptable options. In these situations, it might not be desirable to map business transactions into global ones, but split them into more manageable units of work (global or local transactions) and provide compensating transactions to perform rollbacks.
- They might be used in some long-lived workflow type transactions (flows) composed of several atomic transactions executing outside of the global unit-of-work, in the workflow controlled sequence. Since these individual transactions commit independently of each other, a failure in one of the downstream transactions may require some compensating transactions to reverse (undo) previously committed ones.

It is important to stress that an application that depends on compensating transactions must have extra logic to deal with failures and the possibility that further updates are made to the resource in between the original committed transaction and the undo transaction. Otherwise, the data may be left in an inconsistent state. For these reasons, their usage should be carefully evaluated.

6.3 CICS Web services

In this section we will look at the support available for Web services to support transactions. We begin by looking at the various specification associated with Web services and transactions then move on to look at the CICS implementation of these.

6.3.1 Specifications

There are several related specifications that go together to make up the support in Web services for transactions. The following section gives details of some of the specifications that are relevant to CICS. Figure 6-6 shows how the specifications relate to each other.

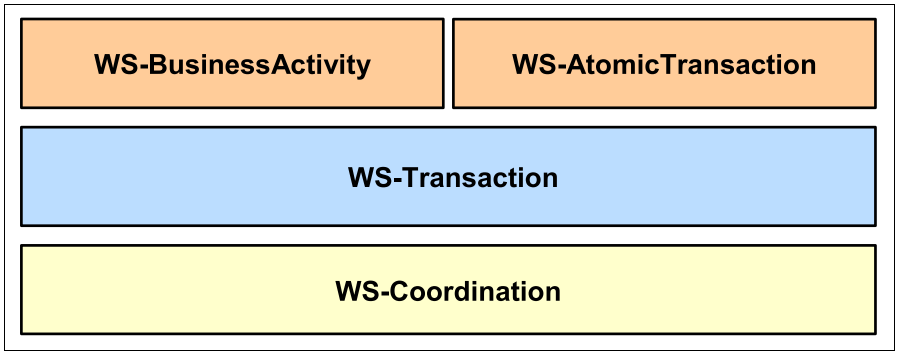


Figure 6-6 Relationship of Web service transaction specifications

WS-Coordination

The WS-Coordination specification describes an extensible framework for providing protocols that coordinate the actions of distributed applications. Such coordination protocols are used to support a number of applications, including those that have to reach consistent agreement on the outcome of distributed activities.

WS-Coordination is a standard developed by IBM, Microsoft, and BEA. A similar standard, WS-CAF, is being developed by OASIS based on WS-Context, WS-Coordination Framework, and WS-Transaction Management specifications published by Arjuna, Fujitsu, Iona, Oracle, and Sun Microsystems.

The specification defines a message exchange that can occur between transactional components within an application. In a typical exchange we might see that:

- An application sends a message to the Activation Service asking for a transactional context
- The Activation Service sends a response containing a transactional context to the application
- A Database-1 and a Database-2 each send a **`Register`** message to the Registration Service and receives a reply

We still need to define what these messages should contain.

Although CICS TS V3.1 was developed and tested when the November 2004, version was the current version, the nature of the differences between the November 2004, version and the August 2005, version is such that it may accurately be said that CICS TS V3.1 also supports the August, 2005, version.

All information items defined by the November, 2004, and August, 2005, versions are identified by the XML namespace URI:

`http://schemas.xmlsoap.org/ws/2004/10/wscoor`

We associate the namespace prefix “wscoor” with this namespace by using the attribute:

```
xmlns:wscoor=”http://schemas.xmlsoap.org/ws/2004/10/wscoor”
```

The specification defines:

- A coordination service
- The following messages:
  – CreateCoordinationContext
  – CreateCoordinationContextResponse
  – Register
  – RegisterResponse

Coordination service

As shown in Figure 6-7 on page 173 a Coordination service (or Coordinator) is an aggregation of the following services:

- Activation service

  When the application sends a `CreateCoordinationContext` element, the Activation service creates a new activity and returns its coordination context in a `CreateCoordinationContextResponse` element.

The Coordination service may, but does not have to, support the Activation Service.

Note: Some products provide this as an external service for others to call. CICS has chosen not to do this, and only supports the creation of coordination contexts internally, for use by the workloads that it manages.

- Registration Service

  The Registration Service defines a `Register` operation that allows a Web service to register to participate in a coordination protocol.

  The Coordination service must support the Registration Service.

- A set of coordination Protocol Services for each supported coordination type.

  These are defined in the specification that defines the coordination type (for example, in the WS-Atomic Transaction specification).

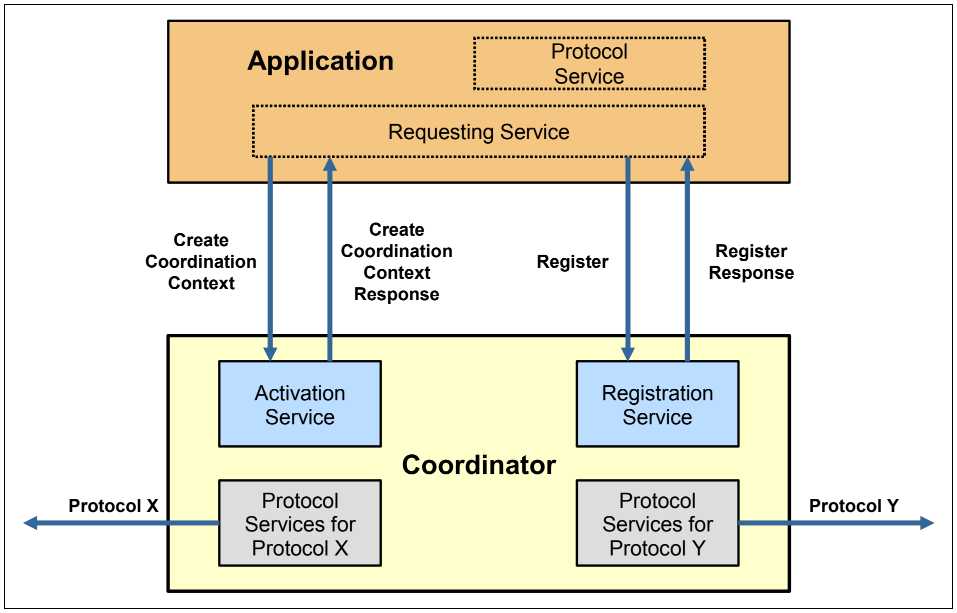


Figure 6-7 A Coordination service (or Coordinator)

Additional information about WS-Coordination can be found at:

http://www.ibm.com/developerworks/library/ws-coor/
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf

WS-Transaction

The WS-Transaction specification describes coordination types that are used with the extensible coordination framework described in the WS-Coordination specification. It defines two coordination types: WS-AtomicTransaction and WS-BusinessActivity. Developers can use either or both of these coordination types when building applications that require consistent agreement on the outcome of distributed activities.

WS-Transaction is a standard developed by IBM, BEA, and Microsoft. More information can be found at:

http://www.ibm.com/developerworks/library/specification/ws-tx/

WS-Atomic Transaction

The WS-Coordination specification defines an extensible framework for defining coordination types. The WS-Atomic Transaction (WS-AT) specification builds on WS-Coordination by providing the definition of the atomic transaction coordination type.

Atomic transactions have an all-or-nothing property. The actions taken prior to commit are only tentative (that is, not persistent and not visible to other activities). When an application finishes, it requests the Coordinator to determine the outcome for the transaction. The Coordinator determines if there were any processing failures by asking the participants to vote. If the participants all vote that they were able to execute successfully, the Coordinator commits all actions taken. If a participant votes that it needs to abort or a participant does not respond at all, the Coordinator aborts all actions taken. Commit makes the tentative actions visible to other transactions. Abort makes the tentative actions appear as though the actions never happened.

IBM, Microsoft, and BEA published the *Web Services- Atomic Transaction (WS - Atomic Transaction)* specification in September of 2003; they updated it in November of 2004. Arjuna Technologies Ltd., Hitachi Ltd., and IONA Technologies joined IBM, Microsoft, and BEA in publishing *Web Services - Atomic Transaction (WS - Atomic Transaction) Version 1.0* in August of 2005. You may find these at:

http://www-128.ibm.com/developerworks/library/specification/ws-tx

Although CICS TS V3.1 was developed and tested when the November,2004, version was the current version, the nature of the differences between the November,2004, version and the August, 2005, version is such that it may accurately be said that CICS TS V3.1 also supports the August, 2005, version.

All information items defined by the November, 2004, and August, 2005, versions are identified by the XML namespace URI:

`http://schemas.xmlsoap.org/ws/2004/10/wsat`

We associate the namespace prefix “wsat” with this namespace by using the attribute:

```
   xmlns:wsat=”http://schemas.xmlsoap.org/ws/2004/10/wsat”
```

The WS-AT specification defines the following protocols for atomic transactions:

- Completion
- Volatile Two-Phase Commit
- Durable Two-Phase Commit

Completion protocol

The **`Completion`** protocol is used by an application to tell the Coordinator to try to either commit or abort an atomic transaction. The **`Completion`** protocol initiates commitment processing. Based on each protocol’s registered participants, the Coordinator begins with Volatile 2PC and then proceeds through Durable 2PC. After the transaction has completed, a status (Committed or Aborted) is returned to the application.

An initiator registers for this protocol by specifying the following URI for the contents of the `ProtocolIdentifier` element in the `Register` element:

`http://schemas.xmlsoap.org/ws/2004/10/wsat/Completion`

Figure 6-8 on page 176 abstractly illustrates the protocol.

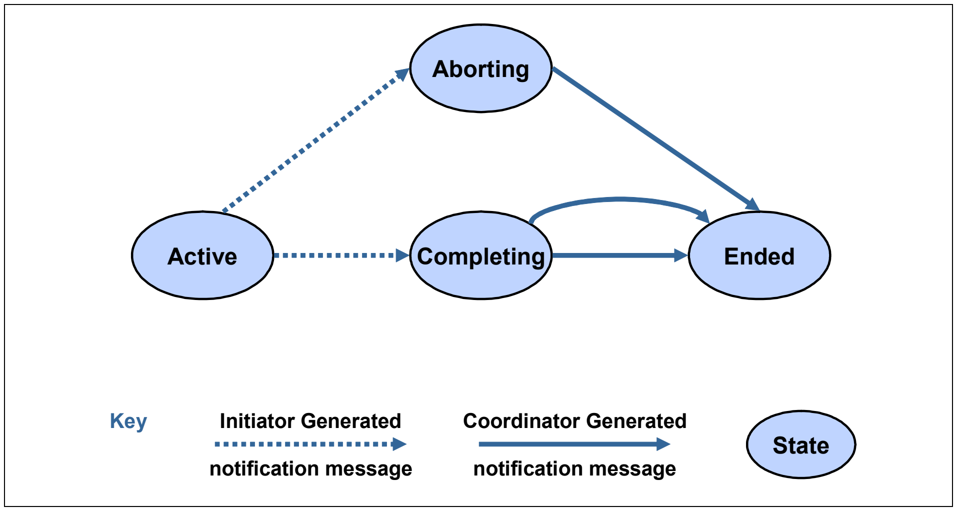


Figure 6-8 Completion protocol

The initiator generates:

- Commit

  Upon receipt of this notification, the Coordinator knows that the initiator has *completed* application processing and that it should attempt to commit the transaction.

- Rollback

  Upon receipt of this notification, the Coordinator knows that the initiator has *terminated* application processing and that it should abort the transaction.

The Coordinator generates:

- Committed

  Upon receipt of this notification, the initiator knows that the Coordinator reached a decision to commit.

- Aborted

  Upon receipt of this notification, the initiator knows that the Coordinator reached a decision to abort.

Two-Phase Commit protocol

The Two-Phase Commit (2PC) protocol defines how multiple registered participants reach agreement on the outcome of an atomic transaction. The 2PC protocol has two variants: Volatile 2PC and Durable 2PC.

Participants managing volatile resources such as a cache should register for this protocol by using the following protocol identifier:

`http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC`

Participants managing durable resources such as a database should register for this protocol by using the following protocol identifier:

`http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC`

Note: When CICS TS V3.1 is a participant in an atomic transaction, it always requests the Durable2PC protocol when it sends a Register request. When CICS TS V3.1 is the coordinator of an atomic transaction, it will tolerate a Register request for Volatile2PC but it will treat it as a Durable2PC request.

After receiving a **`Commit`** notification in the Completion protocol, the root Coordinator begins the Prepare phase of all participants registered for the Volatile 2PC protocol. All participants registered for this protocol must respond before a Prepare is issued to a participant registered for the Durable 2PC protocol. We illustrate this in Figure 6-9 where participants P1 and P3 registered for the Volatile 2PC protocol and participant P2 registered for the Durable 2PC protocol. Both P1 and P3 must respond to the **`Prepare`** notification before the Coordinator can send Prepare to P2.

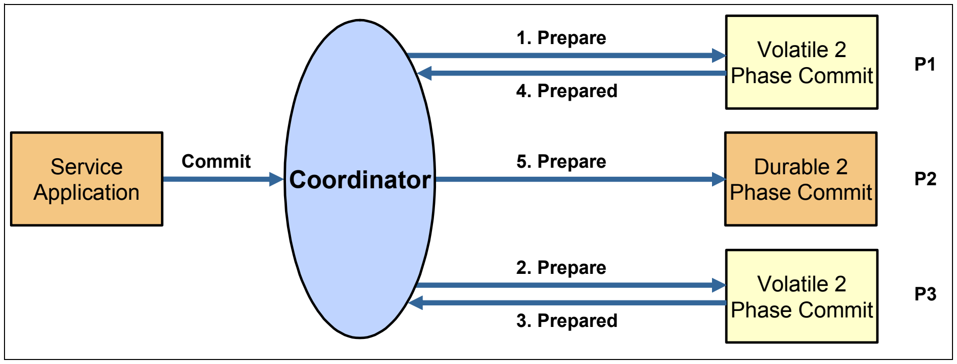


Figure 6-9 Mixture of participants registered for Durable 2PC and Volatile 2PC

Upon successfully completing the prepare phase for Volatile 2PC participants, the root Coordinator begins the Prepare phase for Durable 2PC participants. All participants registered for this protocol must respond Prepared or ReadOnly before a Commit notification is issued to a participant registered for either protocol. A volatile participant is not guaranteed to receive a notification of the transaction's outcome.

Figure 6-10 abstractly illustrates the 2PC protocol.

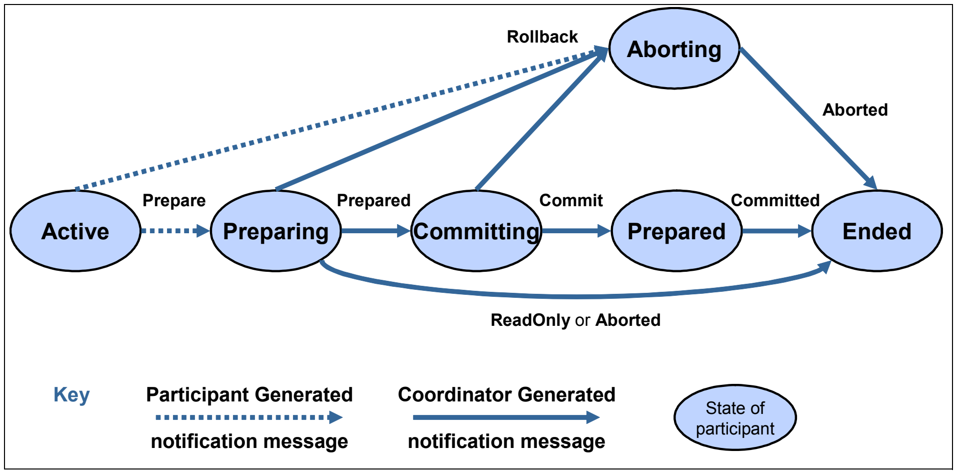


Figure 6-10 Two-Phase Commit protocol

The Coordinator generates:

- **`Prepare`**

  Upon receipt of this notification, the participant should enter phase 1 and vote on the outcome of the transaction.

  – If the participant has already voted, it should resend the same vote

  – If the participant does not know of the transaction, it must vote to abort

- **`Rollback`**

  Upon receipt of this notification, the participant should abort, and forget, the transaction. This notification can be sent in either phase 1 or phase 2. Once sent, the Coordinator may forget all knowledge of this transaction.

- **`Commit`**

  Upon receipt of this notification, the participant should commit the transaction.This notification can only be sent after phase 1 and if the participant voted to commit. If the participant does not know of the transaction, it must send a **`Committed`** notification to the Coordinator.

The participant generates:

- **`Prepared`**

  The participant is prepared and votes to commit the transaction.

- **`ReadOnly`**

  The participant votes to commit the transaction and has forgotten the transaction. The participant does not want to participate in phase two.

Suppose, for example, that the participant received an account number that it could not match to an entry in a database. It might return an error to the requesting application, but, having registered as a participant in the atomic transaction, it would then go on to be coordinated during 2PC processing. When the Coordinator sends **`Prepare`**, the participant replies **`ReadOnly`** and then terminates without waiting for the **`Commit`**. The Coordinator, on receipt of the **`ReadOnly`**, would then delete its own record of the interaction with the participant and would not attempt to send a **`Commit`** to it.

- **`Aborted`**

  The participant has aborted, and forgotten, the transaction.

- **`Committed`**

  The participant has committed the transaction. The Coordinator may safely forget that participant.

- **`Replay`**

The participant has suffered a recoverable failure. The Coordinator should resend the last appropriate protocol notification.

CICS TS V3.1 and resynchronization processing

We have completed our discussion of the WS-AT specification. Unfortunately, the current version of the specification does not completely cover all of the issues surrounding the use of the 2PC protocol. In particular, it does not completely describe the resynchronization processing that should take place following a failure in one of the systems involved in the 2PC protocol or in the network connections that link the systems together. The only thing that the specification mentions relating to resynchronization is the **`Replay`** notification. Therefore, in this section we describe some aspects of how CICS TS V3.1 handles resynchronization processing for transactions which use the 2PC protocol.

Note: For the sake of brevity we do not describe all of the possible issues. For example, we do not describe:

- What happens when resync processing is driven from both sides and a "race" condition results
- What happens when a resync request fails

Network failures can result in messages not being delivered in a timely manner. System failures prevent processing altogether until a restart takes place.

Within the 2PC processing sequence there is a period of time, known as the *in-doubt* window, during which one system is unable to complete processing because it does not know what the other system has done. The distributed UOW is said to be in-doubt when:

- A participant Protocol Service has replied `Prepared` in response to a `Prepare` notification, and
- Has written a log record of its response to signify that it has entered the in-doubt state, and
- Does not yet know the decision of its coordinator (to `Commit` or to `Rollback`).

Barring system or network failures, the UOW remains in-doubt until the coordinator issues either the `Commit` or `Rollback` request as a result of responses received from all UOW participants. If a failure occurs that causes loss of connectivity between a participant and its coordinator, the UOW remains in-doubt until either:

- Recovery from the failure has taken place and synchronization can resume, or
- The in-doubt waiting period is terminated by some built-in control mechanism, and an arbitrary (heuristic) decision is then taken (to commit or back out).

Note that while the UOW remains in-doubt, the recoverable resources that it owns remain locked.

If a system or network failure occurs during the in-doubt window, additional steps must be taken to ensure that the updates are completed in a consistent manner by both systems. This is known as resynchronization processing.

Previous releases of CICS provided a Recovery Manager that dealt with resynchronization processing for distributed workloads that made use of *VTAM®* networks or which used *MRO* connections. These releases dealt with failures during the in-doubt window in one of three ways:

- Automatic heuristic decision

  You could cause CICS to make an automatic heuristic decision by specifying the WAIT, WAITTIME, and ACTION attributes on a TRANSACTION definition.
- Manual heuristic decision

  You could force an in-doubt UOW to complete by issuing a CEMT SET UOW(*uowid*) [COMMIT | BACKOUT] command or its EXEC CICS equivalent.
- Automatic resynchronization

  If you set the WAIT attribute to YES and the WAITTIME attribute to 00.00.00, the transaction would wait until it could communicate with its partner system, after which it could either explicitly request that the message it was waiting for be sent again, or it could resend the last message that it generated.

CICS TS V3.1 extends the Recovery Manager for use by WS-AT workloads. CICS applications that form part of a WS-AT workload can be controlled by any of these mechanisms. However, automatic resynchronization is somewhat different for WS-AT workloads.

The principle difference arises from the fact that WS-AT processing takes place over a TCP/IP network.

- Other forms of distributed transactions make use of communication mechanisms such as VTAM, and resynchronization across a VTAM network can be triggered when the connection between a pair of systems is re-established.
- CICS does not currently support TCP/IP connections in the same way that it does its VTAM connections, and so CICS can only drive resynchronization of WS-AT requests when a region starts.

Important: CICS only drives resynchronization of WS-AT requests during a CICS region restart.

During any type of startup except an initial start, CICS reads the system log to discover any units of work that were in-doubt when the region previously shut down or failed. While reading through the log, CICS may find that it has outstanding units of work that indicate they were involved in an atomic transaction. These log records also indicate whether the UOW was acting as a coordinator or a participant.

- Coordinator

  If it is a coordinator and the log record indicates that the UOW was waiting for a `Committed` or `Aborted` response from a participant when CICS shut down, then the UOW is reactivated (unshunted) and sends out its decision message (`Commit` or `Rollback`) to the participant identified in the log record.

  – If a response is received, then the UOW completes its processing and terminates.

  – If a response is not received before the coordination UOW times out, then CICS shunts the UOW (moves it aside for processing later on). The UOW then persists until another resynchronization attempt takes place or until someone manually forces it to complete. (The coordination UOW times out after 30 seconds, a value set internally by CICS).

- Participant

  If it is a participant and the log record indicates that the UOW had voted in response to a `Prepare` message and was waiting for a `Commit` or `Rollback` decision from its coordinator when communication was lost, then the UOW is reactivated, sends a **`Replay`** message to its coordinator, and once again waits for the decision message to arrive.

- If the decision message is then received, the participant acts on it and sends a **`Committed`** or **`Aborted`** message back to the coordinator before terminating.

If the decision message does not arrive before the participant UOW times out, then CICS shunts the participant UOW. The UOW then persists until another resynchronization attempt takes place or until someone manually forces it to complete.

6.3.2 CICS support for WS-Atomic Transaction

CICS Web services support enables the defining of SOAP processing pipelines. These pipelines can be configured with message handler programs. CICS supplies a message handler for SOAP 1.1 and a handler for SOAP 1.2. These CICS supplied message handlers also have the capability to process SOAP headers. To do this a SOAP header processing program is added that is defined to the pipelines configuration file to be invoked on a match of a particular header.

This section provides an overview of how CICS supplies support for Web service Atomic Transaction through the use of custom message handlers and header processing programs. For simplicity of explanation the example given is of a CICS to CICS Web service. It is understood that most customers would not choose to use WS-AT for workloads distributed entirely within CICS however this allows a single example to show how CICS operates in the role of an atomic transaction coordinator and as a participant in a wider atomic transaction.

Figure 6-11 on page 183 shows two CICS regions: AOR1 and AOR2. A service requester application running in AOR1 invokes a service provider application running in AOR2.

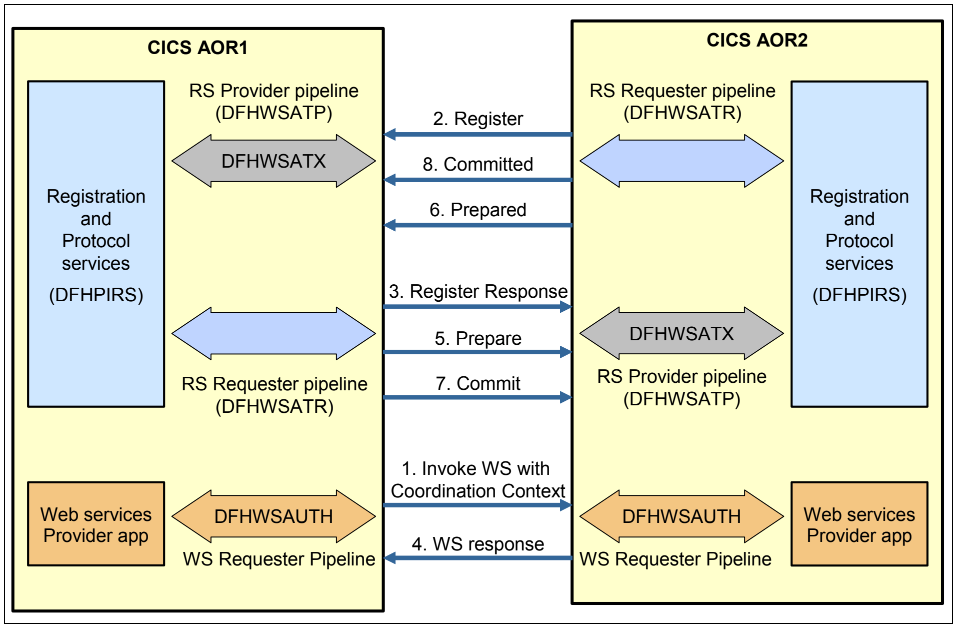

CICS AOR1
CICS AOR2
RS Provider pipeline (DFHWSATP)
DFHWSATX
Registration and Protocol services (DFHPIRS)
RS Requester pipeline (DFHWSATR)
Web services Provider app
DFHWSAUTH
WS Requester Pipeline
2. Register
8. Committed
6. Prepared
3. Register Response
5. Prepare
7. Commit
1. Invoke WS with Coordination Context
4. WS response
RS Requester pipeline (DFHWSATR)
DFHWSATX
RS Provider pipeline (DFHWSATP)
Registration and Protocol services (DFHPIRS)
DFHWSAUTH
Web services Provider app
WS Requester Pipeline

Figure 6-11 CICS as transaction coordinator and participant

In AOR1 the request passes through a pipeline which contains a CICS-provided SOAP message handler module. The SOAP message handler is configured to invoke the CICS-provided header processing program DFHWSATH. DFHWSATH adds a SOAP header containing a `CoordinationContext` to each message that it sends out.

In AOR2 the request passes through a pipeline which supports the Web service that AOR1's application is calling and also invokes the CICS SOAP handler. The handler invokes the header processing program DFHWSATH when it detects a SOAP header which contains a `CoordinationContext` header.

To enable WS-AT support both regions have a requester pipeline named DFHWSATR and a provider pipeline named DFHWSATP for registration and protocol processing. The DFHWSATP pipeline invokes the CICS-supplied message handler DFHWSATX as the last message handler in the pipeline.

For the workload we have shown, AOR1's DFHWSATP pipeline receives registration requests and protocol notifications, while its DFHWSATR pipeline sends registration responses and protocol instructions.

AOR2's DFHWSATP pipeline receives registration responses and protocol instructions, while its DFHWSATR pipeline sends registration requests and protocol notifications.

Note: The DFHWSATP pipeline acts as the registration endpoint for CICS.

Resources required for WS-AT processing

CICS TS V3.1 provides a new resource group DFHWSAT to assist customers with setting up WS-AT support in CICS. The DFHWSAT group contains the resources shown in Table 6-1.

Table 6-1 CICS supplied resource definitions for WS-AT

Resource	Resource Name	Description
Pipeline	DFHWSATP	Registration Services provider PIPELINE
Pipeline	DFHWSATR	Registration Services requester PIPELINE
Urimap	DFHRSURI	URIMAP used by the Registration Services provider
Program	DFHPIRS	Registration and protocol services program
Program	DFHWSATH	SOAP header processing program
Program	DFHWSATR	Registration and coordination services handler program
Program	DFHWSATX	CICS message handler program

Since DFHLIST does not include the DFHWSAT group and you cannot add the DFHWSAT group to DFHLIST, specifying DFHLIST in the system initialization table GRPLIST parameter will not cause CICS to install DFHWSAT automatically during an initial start.

Configuring WS-AT support

As seen above all the notifications for WS-AT are Web service messages. A Web service requires an endpoint for the message to target. The question is how does the participant application know where to register to join the transaction?

The answer is in the pipeline configuration file for the service requester application (running in CICS AOR1) must specify:

- One of the CICS-provided SOAP message handlers (cics_soap_1.1_handler or cics_soap_1.2_handler)
- The mandatory invocation of the DFHWSATH header processing program to add a `CoordinationContext` header to the SOAP request

A `<registration_service_endpoint>` element within the pipeline configuration files `<service_parameter_list>` as shown in Example 6-1.

Example 6-1 Service requester pipeline configuration file which supports WS-AT

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
      xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
      requester.xsd">
  <service>
   <service_handler_list>
     <cics_soap_1.1_handler>
       <headerprogram>
        <program_name>DFHWSATH</program_name>
         <namespace>
          http://schemas.xmlsoap.org/ws/2004/10/wscoor
         </namespace>
        <localname>CoordinationContext</localname>
         <mandatory>true</mandatory>
       </headerprogram>
     </cics_soap_1.1_handler>
   </service_handler_list>
  </service>
  <service_parameter_list>
   <registration_service_endpoint>

http://requester.example.com:3207/cicswsat/RegistrationService
   </registration_service_endpoint>
  </service_parameter_list>
</requester_pipeline>
```

The `<registration_service_endpoint>` element contains the address of the Registration Service endpoint which runs in the requesting CICS region (AOR1). The path component of this address matches the PATH attribute defined in the DFHRSURI URIMAP resource definition of AOR1. Participant Web services should send `Register` requests and `Prepared` and `Committed` (or `Aborted`) notifications to this address.

In the service requester pipeline:

- Since the `<mandatory>` element contains `True`, the pipeline will flow a `CoordinationContext` with the message.
- If you change the `<mandatory>` element to `False` or remove DFHWSATH from the pipeline, the pipeline will not flow a `CoordinationContext` with the message.

The pipeline configuration file for the service provider application must specify:

- One of the CICS-provided SOAP message handlers (`cics_soap_1.1_handler` or `cics_soap_1.2_handler`)
- Invocation of the DFHWSATH header processing program whenever the SOAP message contains a `CoordinationContext` header.
- A `<registration_service_endpoint>` element within a `<service_parameter_list>`

This is shown in Example 6-2.

Example 6-2 Service provider pipeline configuration file which supports WS-AT

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
      xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
      provider.xsd ">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
         <program_name>DFHWSATH</program_name>
          <namespace>
           http://schemas.xmlsoap.org/ws/2004/10/wscoor
          </namespace>
         <localname>CoordinationContext</localname>
         <mandatory>false</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
  <service_parameter_list>
```

```
    <registration_service_endpoint>
   http://provider.example.com:3207/cicswsat/RegistrationService
    </registration_service_endpoint>
  </service_parameter_list>
</provider_pipeline>
```

This time the `<registration_service_endpoint>` element contains the address of the Registration Service endpoint which runs in the provider CICS region. The Coordinator should send `RegisterResponse` messages and `Prepare` and `Commit` (or `Abort`) notifications to this address.

In the service provider pipeline in Example 6-2 on page 186:

- The pipeline will accept flows with a `CoordinationContext`, and such flows will be treated as part of a WS-AT transaction.
- Since the `<mandatory>` element contains `False`, the pipeline will also accept messages *without* a `CoordinationContext` but they will *not* be part of any WS-AT transaction.
- If you change the `<mandatory>` element to `True`, the pipeline will *require* that a `CoordinationContext` flow with the message. A Fault will be raised if a requester attempts to use the service without a `CoordinationContext`.

If you remove DFHWSATH from the pipeline, the pipeline will raise a `mustUnderstand` fault when a `CoordinationContext` arrives with `mustUnderstand` set to `True`.

6.4 CICS Transaction Gateway

This section details transactional considerations when using the JCA resource adapters provided by the CICS TG.

Extended logical units of work

The ECI and EXCI communications protocols used by the CICS TG extend the CICS DPL concept (see 6.1.1, “CICS transactions, tasks and syncpoints” on page 164) by allowing the invoking application to initiate and co-ordinate the CICS unit-of-work. Within the terms of the CICS TG this is referred to as an *extended logical unit-of-work*, and will cause the CICS mirror task to be *long running* so that it does not terminate on return from the initial call. The transactional capabilities of the CICS ECI resource adapter are built upon this ability of the CICS TG to extend the unit-of-work.

An extended logical unit of work is shown in Figure 6-12 on page 188.

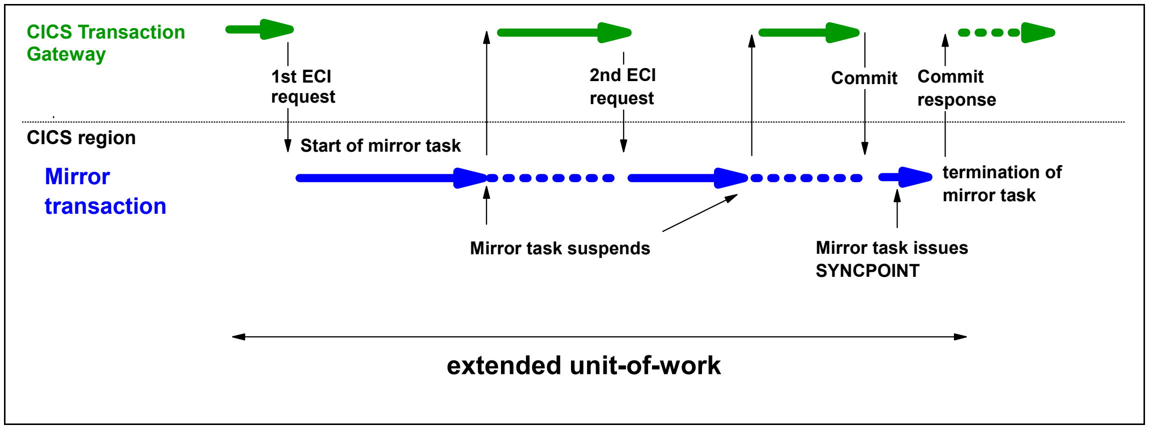


Figure 6-12 Extended logical unit-of-work

JCA

The JCA is part of the J2EE standard and specifies the system contracts to be implemented by a resource adapter. These system contracts define the qualities of service that a resource adapter provides for transaction management, connection management, and security (Figure 6-13).

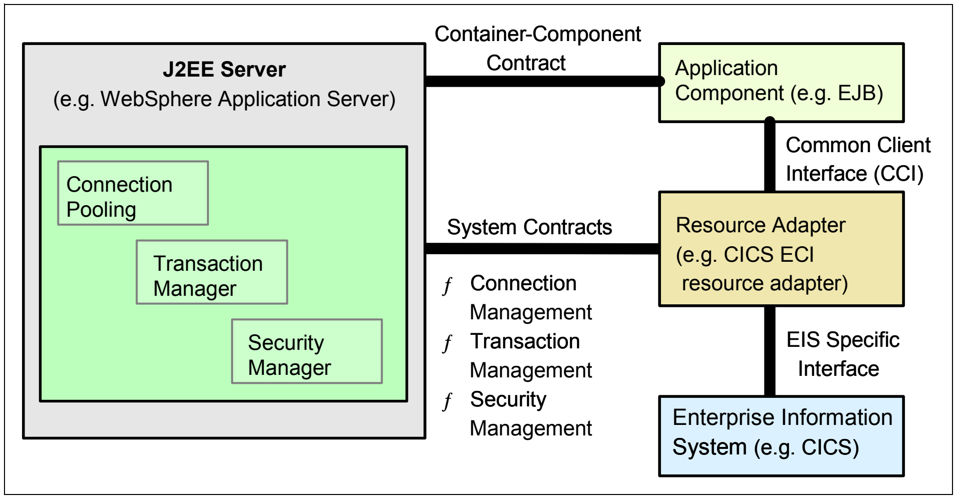

J2EE Server
(e.g. WebSphere Application Server)
Connection Pooling
Transaction Manager
Security Manager
Container-Component Contract
Application Component (e.g. EJB)
Common Client Interface (CCI)
System Contracts
ƒ Connection Management
ƒ Transaction Management
ƒ Security Management
Resource Adapter (e.g. CICS ECI resource adapter)
EIS Specific Interface
Enterprise Information System (e.g. CICS)

Figure 6-13 JCA system contracts

For transaction management, the resource adapter is required to implement one of the following contracts, as defined in the resource adapter's deployment descriptor (ra.xml):

- **XAResource**

  A transaction participant that is called during two-phase commit and which can influence the outcome of the transaction. Typically, an XAResource is implemented by a resource manager and is used to support the external coordination of the resource manager's transaction branch. Enlistment of an XAResource in a transaction is managed by the application server and is not a concern of the application.

- **LocalTransaction**

  A resource adapter that can participate in transactions that are local to the resource manager, but cannot participate in two-phase commit transactions (other than as an only agent or a last participant). For the sake of clarity in this document and other WebSphere-related publications and papers, we use the term *resource manager local transactions* (RMLTs) to refer to a transaction that is local to a single resource manager.

- **NoTransaction**

  A resource adapter with no transactional properties, this can participate in a transactional context but is not influenced by, and has no effect upon the outcome of the transaction.

The WebSphere Application Server transaction support provides coordination, within a transaction, for any number of two-phase capable resource managers. It also enables a single one-phase capable resource manager to be used within a transaction in the absence of any other transactional resource managers. Non-transactional resource managers may also be used within a global transaction but do not participate in the scope of the global transaction.

ECI resource adapter

The CICS ECI resource adapter implements the `LocalTransaction` interface and so has limited support for global transactions. However, when running within WebSphere Application Server for z/OS, the CICS ECI resource adapter supports global transactions, if a local Gateway is used. This signifies the CICS Transaction Gateway runs within the WebSphere Application Server address space and so can use internal functions of the MVS Resource Recovery Services (RRS) available only on z/OS. This provides RRS global transaction support, and allows the CICS ECI resource adapter to participate in a global transaction in WebSphere Application Server for z/OS with any number of other two-phase capable resource managers.

ECI XA resource adapter

The CICS ECI XA resource adapter implements the XAResource interface and so has full support for global transactions. The CICS TG must run on z/OS but the WebSphere Application Server may be on a distributed platform.

6.4.1 WebSphere Application Server transactional support

WebSphere Application Server provides differing qualities of services for the different types of J2EE components, achieved through the use of a set of containers. The four containers are the Web container, EJB container, client container and applet container. JCA support is provided within the Web and EJB containers, both of which provide support for the JCA connection pooling mechanism and propagation of the transaction context from the J2EE component to a JCA interaction.

Transactions in the Web container

The Web container has limited transactional support, as its principal function is for servlet and JSP™ components which are primarily concerned with presentation logic. The Web container does not provide any *container-managed* transaction services. This means that it is not possible to utilize the Web container to make multiple ECI calls to CICS as part of an extended unit-of-work. For example, if the following code (Figure 6-14) is used within a servlet application the two requests 1 and 2 will run as two separate units-of-work within CICS. This is due to the auto-commit behavior of the Web container as defined by the JCA. This specifies that a transactional resource adapter is required to set the auto-commit mode to “on” for all Connection instances outside a transaction.

```
   Context ic = new InitialContext();

   cxnf = (ConnectionFactory) ic.lookup("java:comp/env/eis/ECICICS1");
   Connection cxn = cxnf.getConnection();
   Interaction ixn = cxn.createInteraction();
   ECIInteractionSpec ixnSpec = new
       ECIInteractionSpec(SYNC_SEND_RECEIVE,"CICSPROG");
   JavaStringRecord jsr = new JavaStringRecord();

   jsr.setText("DATA1");
1  ixn.execute(ixnSpec, jsr, jsr);

   jsr.setText("DATA2");
2  ixn.execute(ixnSpec, jsr, jsr);

   ixn.close();
   cxn.close();
```

Figure 6-14 Multiple JCA requests from a servlet

However, servlet applications can *programmatically* utilize either local transactions or global transactions. Local transactions can be controlled by invoking the method `getLocalTransaction()` on the ECI resource adapter `Connection` object; this provides a programmatic transactional context specific to the instance of the JCA connection factory. This can be used to control the scope of the extended unit-of-work created by the CICS ECI resource adapter. Thus in Figure 6-15 requests 1 and 2 will now run as one unit-of-work within CICS.

```
    Context ic = new InitialContext();

    cxnf = (ConnectionFactory) ic.lookup("java:comp/env/eis/ECICICS1");
    Connection cxn = cxnf.getConnection();
    Interaction ixn = cxn.createInteraction();
    ECIInteractionSpec ixnSpec = new
        ECIInteractionSpec(SYNC_SEND_RECEIVE,"CICSPROG");
    JavaStringRecord jsr = new JavaStringRecord();
    LocalTransaction tran = cxn.getLocalTransaction();

    tran.begin();
    jsr.setText("DATA1");
1   ixn.execute(ixnSpec, jsr, jsr);

    jsr.setText("DATA2");
2   ixn.execute(ixnSpec, jsr, jsr);
    tran.commit();

    ixn.close();
    cxn.close();
```

Figure 6-15 Local transaction demarcation in a servlet

Global transactions can also be created within the Web container by using the `javax.transaction.UserTransaction` interface to begin and end a transaction. If the ECI resource adapter is being used in local mode on WebSphere Application Server for z/OS, or the ECI XA resource adapter in conjunction with CICS TG for z/OS, the CICS Transaction Gateway RRS support does allow an ECI request to participate in a bean-managed global transaction. However, as the Web container does not support container management of global transactions, all servlet global transactions must be controlled using the `javax.transaction.UserTransaction` interface (that is, a bean-managed transaction). When the WebSphere Application Server and CICS Transaction Gateway are both on distributed platforms global transaction usage in the Web cannot be used, because of the restriction that the ECI resource adapter only supports the LocalTransaction interface.

Note that it is not possible, or desirable, to extend the life cycle of a transaction over multiple HTTP requests to a servlet, and any global transactions not ended by the end of the HTTP request are rolled-back by WebSphere Application Server.

Transactions in the EJB Container

The EJB container in WebSphere Application Server is ideally suited to the deployment of transactional components, and provides support for both container managed transactions and bean managed transactions. Session beans and message-driven beans may employ either type; entity beans are restricted to contain managed transactions only. Beans using bean-managed transactions are responsible for transaction demarcation and must use the `javax.transaction.UserTransaction` interface to begin and end a transaction. Container-managed transactions are the preferred mechanism as this delegates transactional control to the Application Server, allowing the application developer to concentrate on developing the business logic, while still allowing the transactional properties of the application to be decided upon deployment. The key to transactional control with container managed transactions is the EJB *transaction* attribute.

EJB transaction attribute

The transaction attribute is set in the assembly descriptor section of the EJB deployment descriptor (that is, the file `ejb-jar.xml`). This attribute is used by the EJB container to control under which circumstances a *global* transaction is started when a bean method is invoked.

This transaction attribute appears in the `<container-transaction>` section and is specified with the `<trans-attribute>` tag. For example, the following XML specifies that the `execute()` method on the `CTGTesterCCI` bean has the transaction attribute of `Required`.

```
<container-transaction>
   <method>
      <ejb-name>CTGTesterCCI</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>execute</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
```

The possible values for the transaction attribute are *NotSupported*, *Required*, *RequiresNew*, *Supports*, *Mandatory* and *Never*, and their meanings are described in Table 6-2.

Table 6-2 EJB transaction attribute settings

Transaction Attribute	**Meaning**	**Resulting ECI JCA request**	**CICS mirror task**
NotSupported	Bean method cannot execute within context of an OTS transaction	Non-extended LUW	SYNCONRETURN
Required	Bean method must execute within context of an OTS transaction.	Extended LUW	Long running
RequiresNew	Bean method must execute within context of a new OTS transaction.	Extended LUW	Long running
Supports	Bean method can execute with or without an OTS transaction context.	Non-extended or extended LUW[a]	SYNCONRETURN or long running
Mandatory	Bean method must execute within context of client's OTS transaction.	Extended LUW or Exception thrown [b]	Long running
Never	Bean method must not be invoked in context of an OTS transaction.	Non-extended LUW	SYNCONRETURN

a. For *Supports* the result depends on the existing transactional context; the bean method is executed either under the caller's transaction context, and the ECI call will used an extended logical unit-of-work, or, if there is none present, then it executes under an unspecified transaction context. If under an unspecified transaction context, then the LTC settings control the resulting ECI call type. For further details refer to“Local Transaction Containment (LTC)” on page 194.
b. For *Mandatory*, the bean method is executed under the caller's transactional context. If the caller does not supply a context (in other words there is no global transaction active), then the execute fails with an exception. If there is a global transaction active, the ECI call will used an extended logical unit-of-work

Local Transaction Containment (LTC)

The EJB 2.0 specification does not specify the behavior of the container in the case where a method runs without a global transaction. This can happen for servlets, session beans with a transaction attribute of *Never*, *NotSupported* or *Supports*, and a few other scenarios. In this case the application is said to be executing under an *unspecified transaction* context and WebSphere Application Server implements this unspecified transaction context using a local transaction containment (LTC) policy.

This LTC policy is effectively a scoping device used by the Web and EJB containers to demarcate the beginning and end of work dispatched outside a global transaction. Any access to a resource manager within such a LTC is via a resource manager local transaction (RMLT) that must be resolved by the end of the LTC. There is no *programmatic* control of the LTC possible; instead the following three EJB extended deployment descriptors are used to control the scope of the LTC:

- **Boundary**

  This can have the value *BeanMethod* (default) or *ActivitySession*. It controls the scope of the LTC provided by the EJB container. ActivitySession is an extension to the EJB container available in WebSphere Application Server Enterprise Version 5 onwards. It provides an extended unit-of-work scope beyond the method boundary for local transaction based resource managers. For further information refer to the Redpaper *Transactional services in WebSphere Application Server Enterprise V5*, REDP-3759

- **Resolver**

  This can have the values *ContainerAtBoundary* or *Application* (default). When an ECI request is made from the EJB container outside of a global transaction context (such as with a transaction attribute of Never), then if the Resolver attribute is set to Application, the ECI call type will be non-extended. Conversely, if the Resolver attribute is set to ContainerAtBoundary, then a resource manager local transaction will be started, and the ECI call type will be extended and be resolved by the container at the EJB method boundary.

- **UnresolvedAction**

  This can have the value Commit or Rollback (default). It can be specified for the EJB or Web containers, and signifies how the container should clean up any Connections with an outstanding RMLT at the LTC boundary. This is the only configurable LTC setting for Web components (servlets), and applies to bean managed transactions using the LocalTransaction.begin() method on a Connection. All container managed transactions are auto-committed after the interaction completes, and so this attribute is not used by container managed transactions in the Web container.

LTC support limitations: LTC scope is local to each J2EE component; thus if EJB component A is dispatched under LTC A and then calls EJB component B, component B will be dispatched under a separate LTC. Also the container always establishes an LTC scope if an application executes outside a global transaction, unless Web components or enterprise beans are at a level prior to J2EE 1.3.

WebSphere Application Server for z/OS

The WebSphere Application Server for z/OS transaction manager has the capability to coordinate both XA and RRS compliant connectors in the same transaction automatically on behalf of the application. This means that any number of local CICS ECI connection factories can be coordinated in a global transaction with other two-phase commit connectors such as IMS or JDBC resources. See Figure 6-16 on page 196.

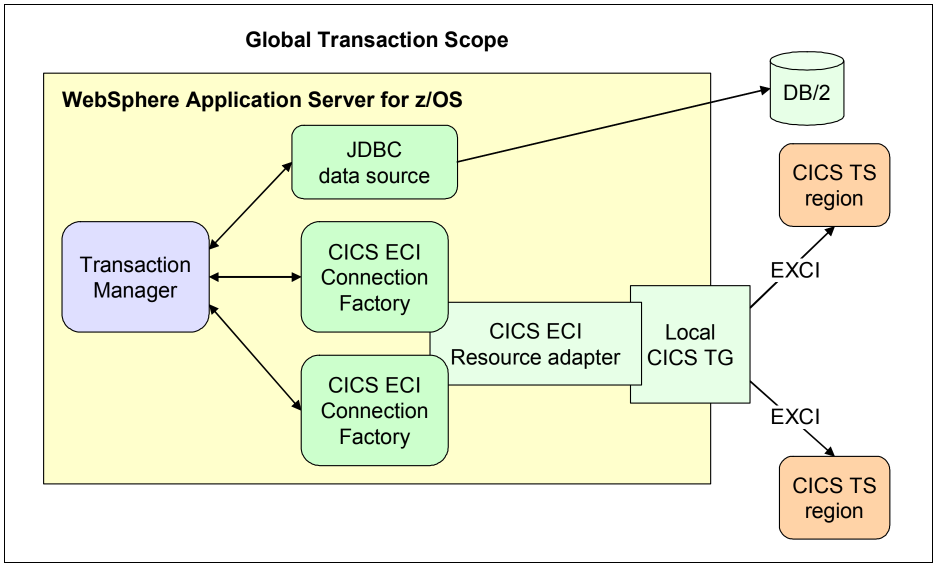

Global Transaction Scope
WebSphere Application Server for z/OS
JDBC data source
DB/2
CICS TS region
Transaction Manager
CICS ECI Connection Factory
CICS ECI Resource adapter
Local CICS TG
EXCI
EXCI
CICS ECI Connection Factory
CICS TS region

Figure 6-16 Global transaction support in WebSphere Application Server on z/OS

The two-phase commit capability that is provided when using the CICS ECI resource adapter with WebSphere Application Server on z/OS requires that a local connection is used between the application server and the CICS TS region. In addition, the CICS TS region must be running on the same MVS LPAR as the application server.

WebSphere Application Server for z/OS also permits the usage of a single one-phase commit capable resource with any RRS-capable resources in the same transaction. If CICS is accessed via a remote connection using the CICS TG TCP protocol, WebSphere Application Server for z/OS includes CICS as the last participant in the global transaction as long as all other resource connectors used by the application (for example, a Java Database Connection (JDBC) to DB2 for z/OS and Java Message Service (JMS) to WebSphere MQ for z/OS) are local and exploit RRS as their transaction manager.

6.4.2 Transactional coordination from WebSphere to CICS

So what are the implications for your transactional JCA applications? In this section, we summarize the key points you need to understand when designing and deploying your transactional JCA applications.

We use the topologies introduced in "Using the CICS ECI resource adapter with different topologies" on page 55.

- In topology 3, the use of the CICS ECI resource adapter with a local CICS TG in WebSphere Application Server for z/OS, provides the ability for any number of CICS regions to participate in a global transaction with two-phase commit coordination with any number of other RRS or XAResource capable resource managers (see Figure 6-16 on page 196).
- In topology 2, the use of the CICS ECI XA resource adapter provides the ability for any number of CICS regions to participate in a global transaction with two-phase commit co-ordination with a number of other XAResource capable resource managers.
- In topology 1 or 2, the WebSphere Application Server transaction manager can provide coordination, within a transaction, for any number of two-phase capable resource managers (such as a JDBC resource). It also enables a single one-phase capable resource manager such as the CICS ECI resource adapter to be used within a global transaction in the absence of any other transactional resource managers.
- In topology 1, to commit a JCA request to a CICS region as part of a global transaction with other two-phase commit resource managers, it is necessary to utilize the Last Participant Support function of WebSphere Application Server. This is also true of topology 2, unless the CICS ECI XA resource adapter is used.
- In topology 1 or 2, to commit multiple JCA requests to the same CICS region as one unit-of-work, you can use the EJB transaction attribute to specify the requests as being part of the same global transaction.
- In topology 1 or 2, to commit multiple JCA requests to different CICS regions as part of the same one-phase commit unit-of-work scope you can use one of two methods:
  - The LTC scoping of the EJB container, by specifying a *Transaction* scope of *None* and a LTC scope of *ContainerAtBoundary*.
  - The `getLocalTransaction()` method (see Figure 6-15 on page 192) to programmatically define the scope of a local transaction containing all the updates. This is supported in the Web or EJB containers.

- It is not possible in topology 1 to commit multiple JCA requests to different CICS regions as part of the same global transaction with other two-phase commit resource managers.
- Non-transactional resource managers may be used within a global transaction. These are not coordinated as part of the global transaction and have no affect on the transaction.

6.5 CICS Web support

CICS Web support has no support for participating in global transactions. All the work is done in standard CICS transactions, though there are several that are used in processing a single request from a client as shown in Figure 6-17.

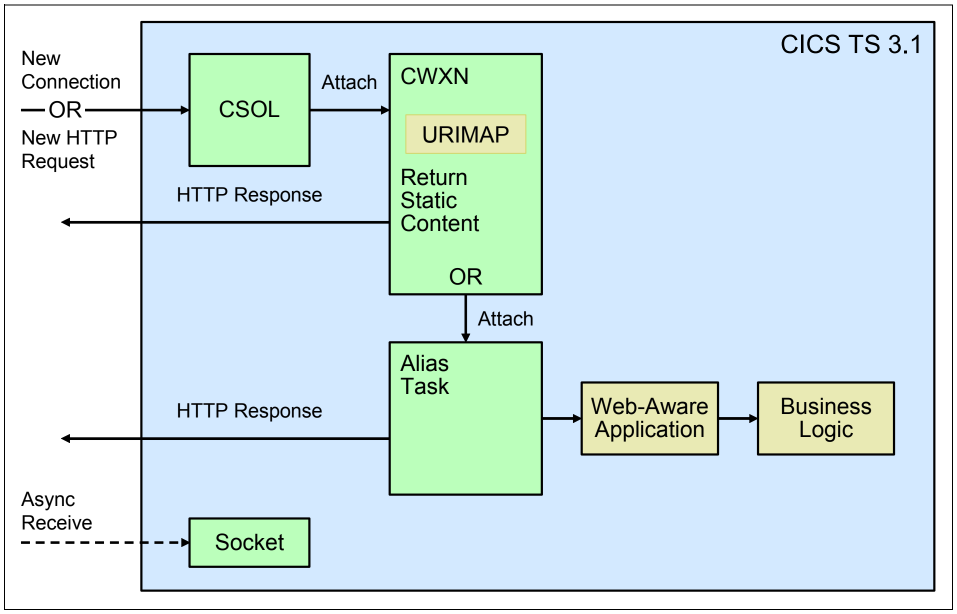


Figure 6-17 Transactions used to process an HTTP request

The sockets listener task, CSOL, is the first transaction involved in processing the request. It is a long running CICS system task that accepts the new connection and starts the CWXN task to receive the data. It will also attach a CWXN task when new HTTP requests arrive on existing connections.

The Web receiver task, CWXN, receives the request from the client. It uses URIMAPs and/or the analyzer program to determine how to handle the request. The URIMAP can indicate that the request is for static content. In which case CWXN will return the data to the client, issue an asynchronous receive to wait for further input, and it will not attach an alias task. Otherwise the alias task is attached. Regardless of whether the alias is attached or not CWXN then terminates.

The alias task is the third CICS transaction involved in processing the request. This is where the actual user application runs. The default tranid for the alias task is CWBA but it can be changed by a URIMAP or the analyzer program to be any tranid currently installed in the CICS region. The alias task must run in the same region as the CWXN task that started it. The user application can run in a remote region as long as it only uses a COMMAREA interface and does not attempt to use the `EXEC CICS WEB API` commands. Any updates to recoverable resources will be committed or rolled back at the end of the alias transaction assuming an explicit sync point request has not been made by the application. At the end of the alias task, if HTTP persistent sessions are being used an asynchronous receive will be issued to wait for further requests from the same client. This receive does not require the alias task to remain in the system so it terminates.

7

Performance and scalability

In this chapter, we start by providing an introduction to the theory of performance and capacity planning as it applies to CICS systems. Then, for each of the CICS access technologies, we discuss the best approach for optimizing the performance and scalability of the desired solution.

The key factors to consider for performance optimization for a particular solution are listed here:

- The application architecture which the client will use
- The performance characteristics of the existing CICS applications
- Network considerations such as message size, SSL encryption and use of persistent connections
- System constraints such as thread or TCB usage
- The workload management techniques which you will use to create a scalable solution

7.1 The theory of performance

CICS has proven to be a high performing and scalable transaction processing system. Good performance implies the efficient use of computing resources and a resulting fast response time. Scalability, on the other hand, implies that the system is capable of running a large number of simultaneous transactions, and that the cost of running a transaction does not increase as the workload increases. A truly scalable solution can be regarded as one that initially performs well at a low rate of usage and whose throughput increases in linear proportion to the number of end users, up to the point at which it saturates the system. This is known as *positive linear scaling*.

The two graphs in Figure 7-1 illustrate how a linearly scalable system should behave. The plot of CPU cost against throughput (transactions per second) shows that the cost per transaction should remain constant as the workload increases. The plot of throughput against number of users shows a linear increase as workload increases, until a resource constraint is reached. The plots are simplistic and for illustration purposes only; the assumption is made that each user is initiating the same amount of work.

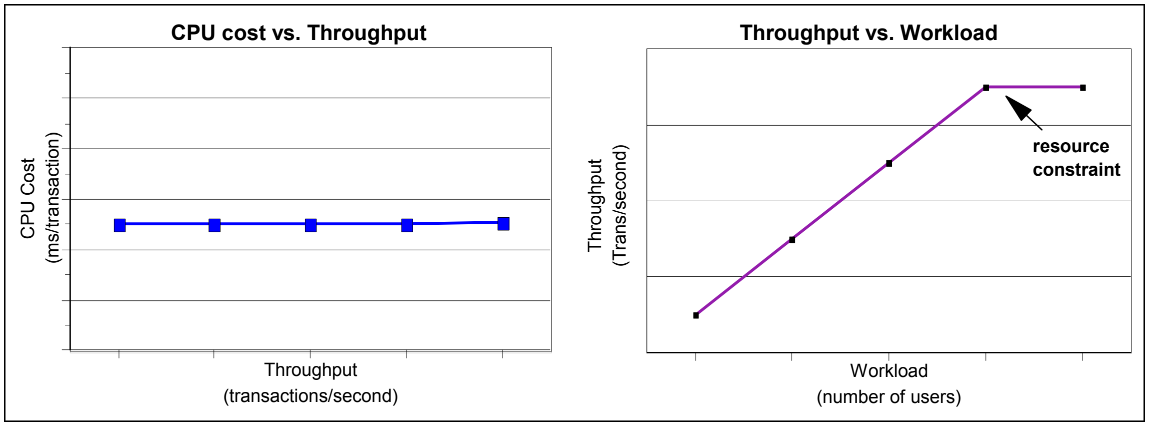


Figure 7-1 Performance of a linearly scalable system

Since computer systems are not infinitely fast, there is always a limit to the system performance level at which the system is saturated; the question is whether or not this limit is acceptable, and what is the limiting factor. The four limiting physical factors are the CPU system, the memory system, the disk I/O system, and the network I/O system.

Given an ever-increasing load, any computer system will eventually be limited by one of these factors. A good architecture will ensure that this limitation does not occur too early. Figure 7-2 outlines a sensible approach for reviewing performance and capacity of a system.

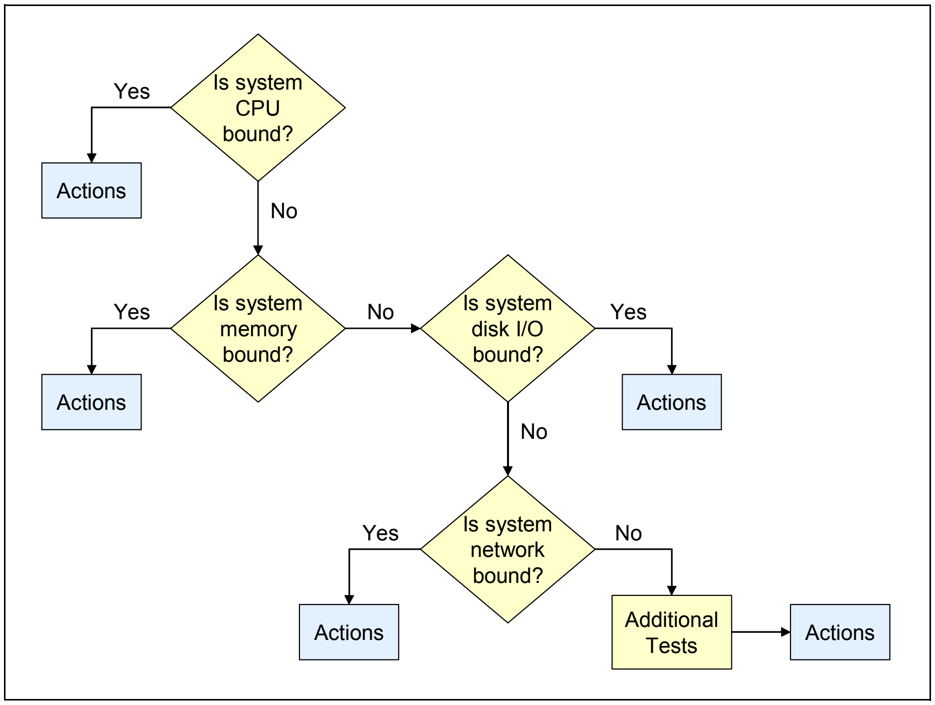


Figure 7-2 Performance flowchart

The CICS access solutions described in this redbook function principally as Internet gateways to enterprise systems. Because of this, they are unlikely to be bound by disk I/O constraints which often affect transaction processing and database systems. Instead, these systems will typically be limited by how quickly and efficiently the components in the system can send the message from the client through the network via the memory structures of the intervening software components.

This is borne out by scaling tests using the CICS Transaction Gateway (CICS TG), which have shown that given sufficient physical memory, its performance is CPU-limited and is inversely proportional to message length. This is illustrated in Figure 7-3 on page 204, with some real-time data for a CICS TG ECI workload.

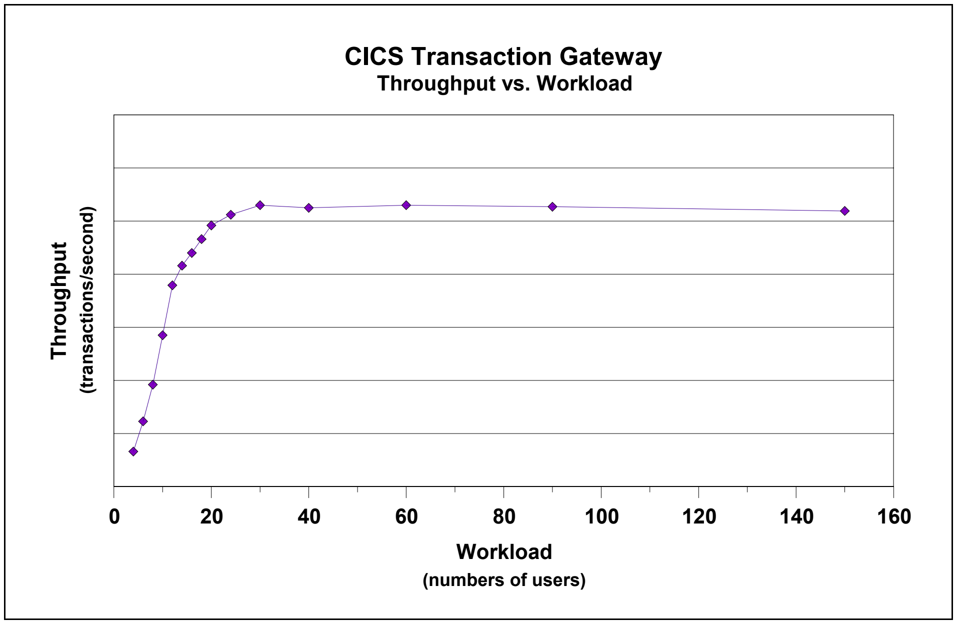


Figure 7-3 Scalability of a CICS TG ECI workload

In this chapter, we analyze the CICS access solutions and contrast how each is likely to affect the performance and scalability of a CICS architecture. Where possible, we highlight practical information. Most of this has been researched using simple CICS applications with very little or no business or data logic within them, unlike typical customer applications.

Thus, a two-fold increase in CPU consumption observed when a particular CICS solution was used does not imply that a real customer application will experience a two-fold decrease in performance. However, whatever architecture is chosen, its performance and scalability should always be verified by a pilot before deployment.

For further details on the performance implications of Web-enabling CICS applications, refer to *A Performance Study of Web Access to CICS*, SG24-5748, which provides actual CPU utilization figures and a detailed discussion of tuning recommendations.

7.2 CICS Web services

CICS TS V3.1 provides fully integrated support for Web services through the use of PIPELINE and WEBSERVICE resources. It provides support for CICS applications to function as both service providers and service requesters. It also provides utilities for creating the artefacts required to marshall and demarshall XML into language structures using the WEBSERVICE resource.

In this section we compare the performance and scalability of the Web service support available in CICS V3.1 using the CICS Web services assistant, CICS V3.1 Web service support using WebSphere Studio Enterprise Developer V5.1 converter programs for XML marshalling, and the SOAP for CICS feature in a CICS TS V3.1 environment. We will also discuss briefly how application design and architecture choices will affect the performance.

The data in this section is based on a the CICS TS v3.1 performance report from the CICS System z Performance team at IBM Hursley. The workload is based on what they considered to be a typical SOAP message request.

Note: These comparisons do not consider WebSphere Developer for System z V6 XML converter programs. Significant performance improvements have been made in this area between WebSphere Studio Enterprise Edition V5.1 and WebSphere Developer for System z v6.

The general observations based on the performance data are:

- Total CPU increases linearly as the transaction rate increases.
- CPU/transaction increases linearly as the return message size increases.
- CPU/transaction increases linearly as the number of elements in the XML message increases.
- CICS V3.1 Web service support significantly out-performs the SOAP for CICS feature.
- Persistent connections improve SOAP performance.

Tip: Significant performance improvements have been delivered through service on the CICS TS V3.1 code. Always apply the latest service available for the Web service support to take advantage of this.

7.2.1 Throughput

In this section we compare the relative throughput performance of the Web service enablement options available including:

- Choice of XML marshalling technology
- Choice of transport technology

CICS as a Web service provider

Figure 7-4 shows the relative CPU requirements to process a typical SOAP message. The comparison is between the Web services support of CICS TS V3.1 using a custom written XML parser, a parser generated using WebSphere Studio Enterprise Developer V5.1 (shown as WSED) and marshalling performed by the CICS Web Services Assistant (shown as CWSA). Note the performance improvement achieved with service applied to the CICS Web Services Assistant.

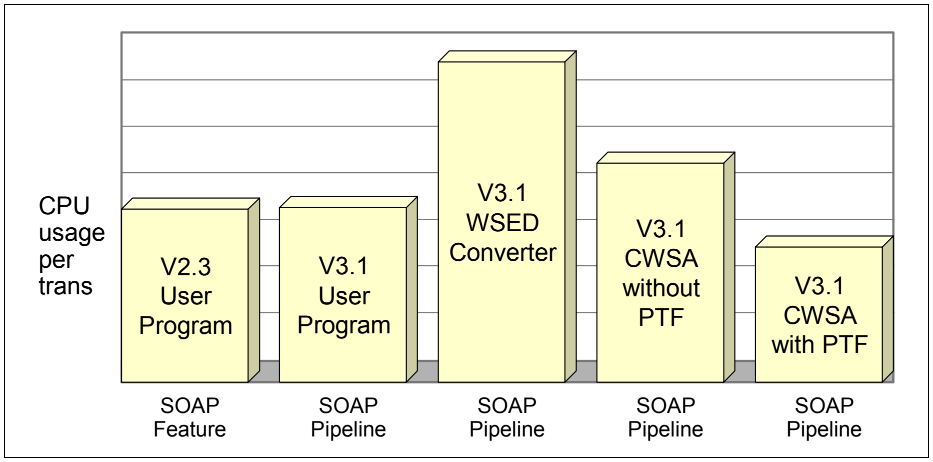


Figure 7-4 Comparison of CPU usage of Web service support

Note: Significant performance improvements have been made in the generated XML converters between WebSphere Studio Enterprise Edition V5.1 and WebSphere Developer for System z V6.

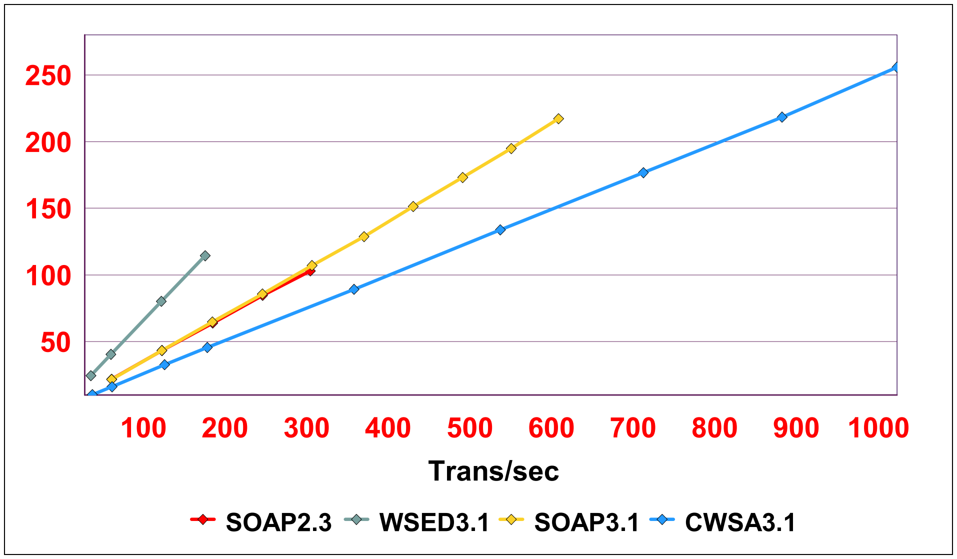


Figure 7-5 CPU Utilization against transaction rates

Figure 7-5 shows a linear increase in CPU usage as the transaction rate increases. We can see here that once again the CICS Web Services Assistant support provides the best performance in this particular test.

CICS as a Web service requester

The SOAP for CICS feature for CICS TS V2.3 introduced the capability for CICS to be a Web service requester making outbound calls to other Web service providers. The Web services support in CICS TS V3.1 also has this support. Figure 7-6 on page 208 shows the relative performances of these two implementations.

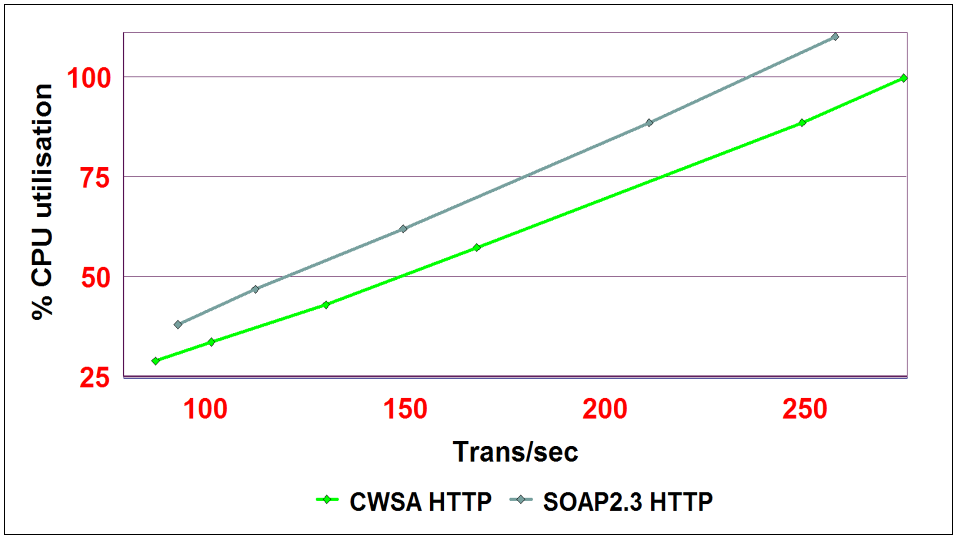


Figure 7-6 Web service requester: CICS Web services support verses SOAP feature

7.2.2 Transport considerations

There are two transports supported by the CICS Web services support:

- HTTP
- WebSphere MQ

When HTTP is used as the transport, persistent connections outperform non-persistent connections. The reason for this is that, when using persistent connections, the client will reuse both the connection and the CWXN task in CICS on subsequent invocations of a service provider on the same CICS system. If the connection is not persistent then the client will have to establish a new connection and also cause a new CWXN transaction to be created for every invocation of a service provider.

When WebSphere MQ is used as the transport, WebSphere MQ storage requirements, address space storage, data storage, library storage, system LX usage, logging, and backup and recovery environments will have an impact on the performance. For more information, see *WebSphere MQ for z/OS Concepts and Planning Guide,* GC34-6051.

Figure 7-7 on page 209 shows the relative performance of a Web service provider with HTTP and MQ. Both transports appear to scale linearly. Note again the importance of applying service to the Web service support.

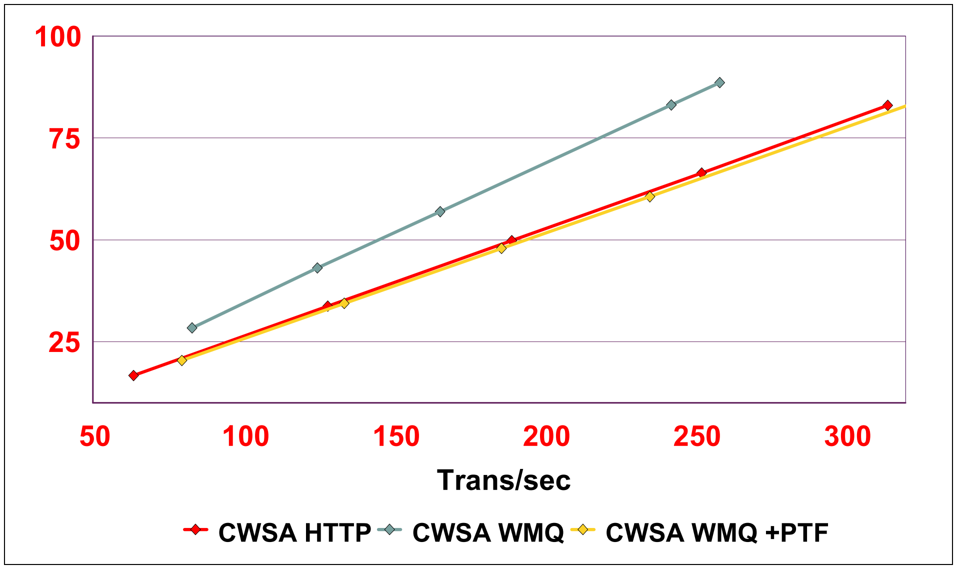


Figure 7-7 CPU utilization against transaction rates for HTTP versus WebSphere MQ

7.2.3 Design and architecture considerations

Designing and architecting a solution that meets the performance requirements has to be considered from the start, not afterwards.

The architecture should guarantee the behavior of the application and operate within a range of acceptable measurements; it must also ensure that the overall behavior of the solution is predictable. Here we will briefly look at the performance impact of message size, message structure, and WebSphere tooling.

Keep message size small

The obvious rule is to keep your payload small and simple. However, in the real world, you do not always have the luxury of adhering to this rule. Larger messages result in longer parsing times. It increases linearly as shown in Figure 7-8 on page 210. You should be aware of these impacts and try to minimize the size of the XML message.

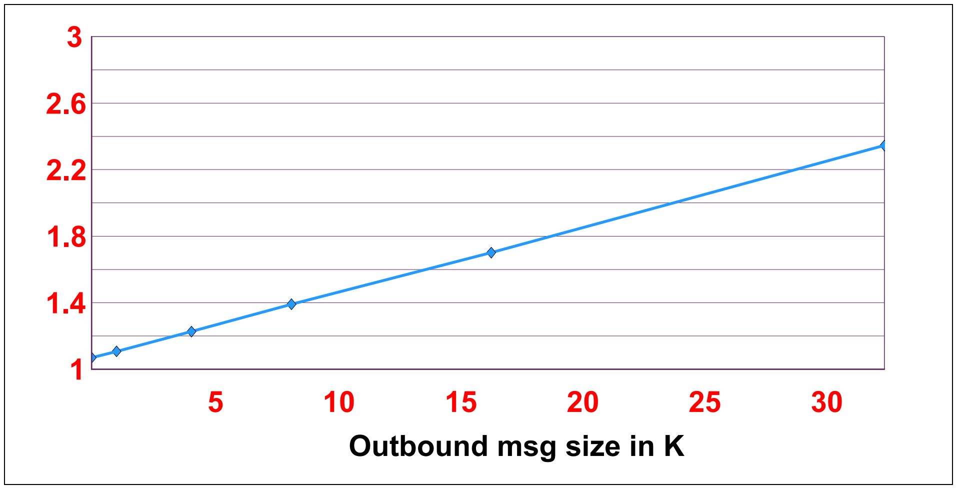


Figure 7-8 CPU versus message size

Keep message structure simple

Complex XML structures with nested elements result in longer times for marshalling and un-marshalling the XML elements. XML elements are extensible and they have relationships as shown in Example 7-1. The book is the root element. Title, prod, and chapter are child elements of book. Book is the parent element of title, prod, and chapter. Title, prod, and chapter are siblings because they have the same parent.

Example 7-1 An XML example showing XML elements

```
<book>
   <title>All you wanted to know about CICS Web services</title>
   <prod id="SG241234" media="paper"></prod>
   <chapter>Overview of Web services
      <para>what is a Web services</para>
      <para>How do Web services relate to SOA</para>
   </chapter>

   <chapter>Expose a CICS application as a Web service
      <para>The CICS Web Services Assistant</para>
      <para>WebSphere Developer for System z V6</para>
   </chapter>

</book>
```

There is a linear increase in CPU/transaction as the number of elements increases as shown in Figure 7-9 on page 211.

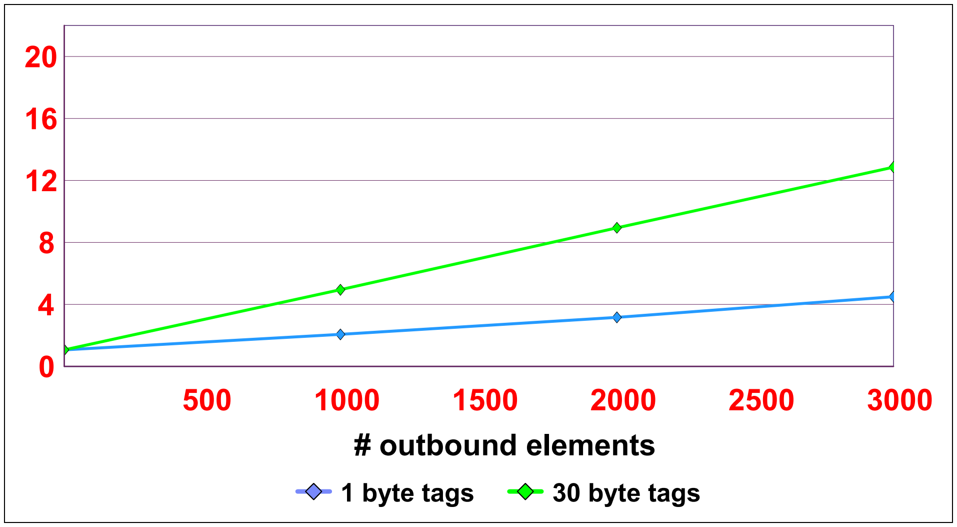


Figure 7-9 CPU/Transaction versus number of elements

7.2.4 SSL considerations

As mentioned earlier, security comes with a performance cost. How much security is enough depends on the business requirements. The security between network nodes over the Internet is traditionally provided using SSL over HTTP (HTTPS). With HTTPS, you can perform mutual authentication of both the sender and receiver of messages and ensure message confidentiality.

If security is needed end-to-end through the application stack, or if security must be independent of the networking protocol, then other means must be considered. WS-Security specifies authentication and message integrity through XML Digital Signatures and message confidentiality through XML encryption using X.509 certificates in both instances. For details, see Chapter 5, “Security” on page 101. However, there are performance trades-offs that must be understood and evaluated against the overall set of requirements.

A common practice is to combine the two approaches by using SSL for encryption and then using XML Digital Signatures to authenticate the application end points and to ensure message integrity. Keep in mind that SSL will also involve at least one authentication of the server to whom a message is being sent, thus some redundancy is occurring.

Also different SSL key size and the encryption algorithms have different performance impacts.

7.2.5 Workload balancing

As the transaction rate increases it will become necessary to balance the workload across multiple CICS regions. For HTTP this can be achieved by using the Sysplex Distributor to route the incoming requests to different CICS regions. For WebSphere MQ a queue sharing group (described in *WebSphere MQ for z/OS Concepts and Planning Guide V5.3.1,* GC34-6051) can be setup to allow multiple CICS regions to service messages on the same queue.

Once within CICS the existing business logic application that is linked to from the message adapter can be on an AOR and workload managed, for example, by CICSPlex SM. This is shown in Figure 7-10.

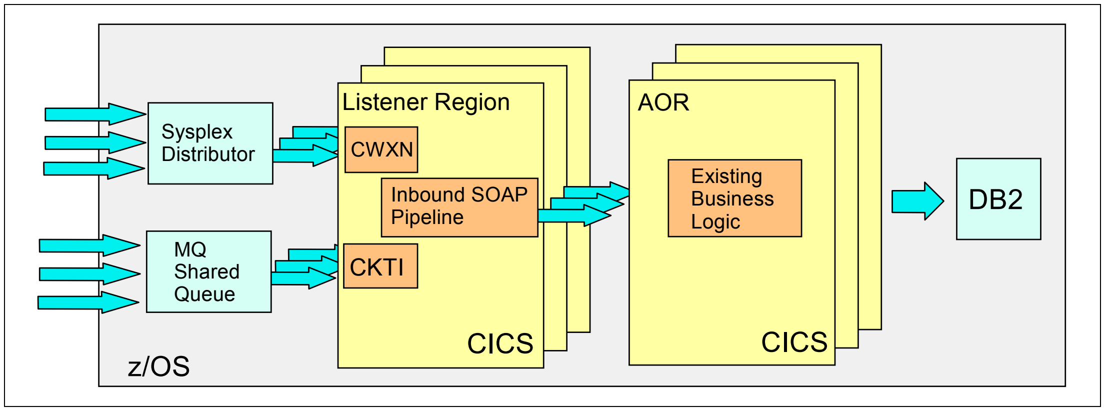


Figure 7-10 Workload can be balanced across multiple listener regions and AORs

7.2.6 Scalability and availability considerations

After you have successfully configured and tested your CICS Web service configuration, you should consider how you can clone the CICS regions in order to improve scalability and availability.

The principal areas for consideration are:

- How to load balance TCP/IP requests across multiple CICS listener regions
- How to load balance Web service requests dynamically across multiple CICS AORs

TCP/IP load balancing

CICS is designed to work with Sysplex Distributor. Sysplex Distributor is an integral part of z/OS Communications Manager, which offers the ability to load balance incoming socket open requests across different address spaces running on different IP stacks (usually on different LPARs). The routing decision is based on real-time socket status and z/OS Quality of Service (QoS) criteria. This provides the benefit of balancing work across different MVS images, providing enhanced scalability and failover in a z/OS Parallel Sysplex.

High availability configuration

Figure 7-11 shows the recommended high availability configuration. CICSPlex SM provides a dynamic routing program which supports the dynamic routing of transactions. This provides the ability for applications invoked by Web service requests to be dynamically routed across a CICSplex.

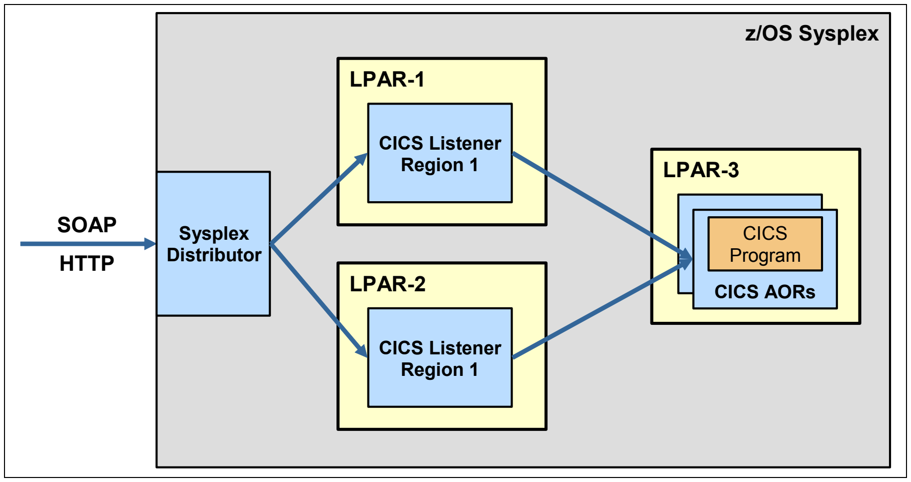


Figure 7-11 High scalability and availability configuration

Routing inbound Web service requests

Inbound Web service requests can be routed to a different CICS region than the one that receives the request using one of two routing models:

- Distributed routing
- Dynamic program routing

The distributed routing model

The transaction that runs the target application program is eligible for routing when one of the following is true:

- The content of the DFHWS-USERID container has been changed by a program in the pipeline
- The content of the DFHWS-TRANID container has been changed by a program in the pipeline
- The transaction is defined as DYNAMIC or with REMOTESYSTEM(sysid)

Figure 7-12 shows how the distributed routing model can be used to route requests for the ORDR transaction. The routing can be controlled by the routing program specified in the DSRTPGM system initialization parameter. CICSPlex SM can be used to balance the routing requests across multiple AORs.

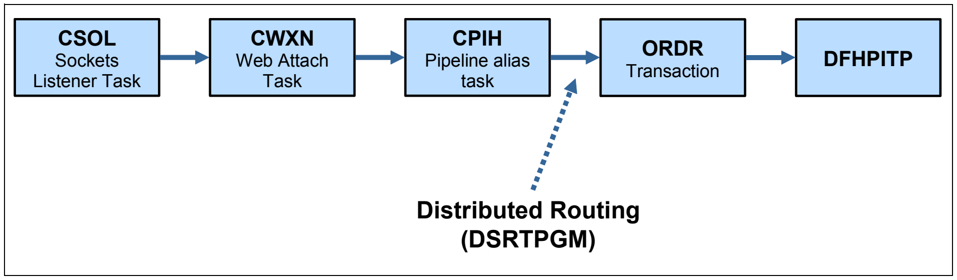


Figure 7-12 Web service provider - distributed routing

Special considerations have to be made when configuring a pipeline to be used in a distributed routing environment. Table 7-1 shows the resource definition requirements for both the listener region and AOR, and whether each resource definition can be shared between the regions.

Table 7-1 Pipeline resource definitions in dynamic routing configuration

Resource	Listener region	AOR
TCPIPSERVICE	required	not required
PIPELINE	required, shared	required, shared
WEBSERVICE	automatically installed from PIPELINE, shared	required, automatically installed from PIPELINE, shared
Pipeline configuration file	required, shared	required, shared
TRANSACTION definition	DYNAMIC(YES)	DYNAMIC(NO)

The dynamic routing model

An alternative way to dynamically route a Web service request, is at the point where CICS links to the user program, in our case DFH0XCMN. At this point (Figure 7-13 on page 215) the request is routed using the dynamic routing model. In this scenario, the routing can be controlled by the program specified in the DTRPGM system initialization parameter. CICSPlex SM can be used to balance the program link requests across multiple AORs.

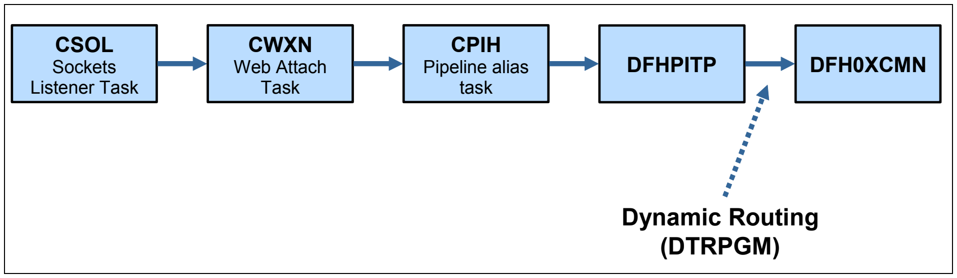


Figure 7-13 Web service provider - dynamic routing

7.2.7 Other performance considerations

The majority of other runtime performance issues for a CICS Web services solution will be no different than other CICS applications. As previously mentioned, consider these issues:

- Hardware, capacity settings
- Operating system parameter settings
- Memory management
- LE runtime
- Transaction logs
- Server logs

Once the solution is operational, then an iterative process is applied to fine-tune a solution by capturing measurements from simulated loads, making adjustments and measuring again to understand their influence. You can iterate as many times as you need until the solution meets the business requirements.

7.3 CICS Transaction Gateway

This section discusses performances considerations when using the CICS Transaction Gateway (CICS TG). It contains the following topics:

- Comparison of JCA versus CICS TG base classes
- Application architectures
- Topology decisions

7.3.1 Comparison of JCA versus CICS TG base classes

The CICS TG offers a variety of Java-based APIs to access existing CICS applications. These include:

- The JCA and the CCI-based interfaces provided with the ECI and EPI resource adapters
- The lower level classes including the `ECIRequest`, `EPIRequest`, and `JavaGateway` classes
- The EPI support classes

The JCA offers the distinct advantage that if the application is deployed into the managed environment of WebSphere Application Server, the application will benefit from the connection pooling, transaction and security management provided by the application server. These qualities of service are controlled by system contracts between the application server and the resource adapter, and can be configured and modified at the time of deployment. This frees the application programmer to concentrate on developing the business logic within the application.

The lower level classes, on the other hand, offer a simple, more procedural style of application development and can be utilized from any Java runtime environment. Any transactional, security or connection usage considerations will need to be carefully designed into the existing application.

7.3.2 Application architectures

When using the CICS TG to provide SOA access to your CICS applications, you will need to decide which is the most appropriate application architecture to use. The recommended and most common architecture is to use the ECI JCA approach which is what will be discussed here. There are further considerations are as follows:

- Whether to use tool generated code or write your own code by hand.
- How to minimize the amount of data which is transmitted between different tiers of the physical configuration.

Comparison of tool generated code versus hand coding

The CCI provided by the JCA is targeted primarily toward application development tools and enterprise application integration frameworks. Rational Application Developer provides a development environment that contains a range of JCA resource adapters, including the CICS ECI adapters.

Rational Application Developer is targeted to development projects requiring integration with back-end systems and provides support for a variety of client types, including Web services clients.

Several factors should be considered when choosing between a hand-coded development process and a tooled solution like Rational Application Developer. From a strictly performance point of view, the performance of hand-coded CCI applications for accessing simple CICS applications is likely to be better than tool-generated applications because of the additional abstraction and runtime requirements of the tool-generated applications. The performance differential for more complex applications is less significant and hand-coded applications are more expensive to develop and maintain.

Tip: Performance tests have shown that a significant performance benefit can be obtained when using the CCI by caching the result of the connection factory instance lookup so that the lookup is not repeated each time the application is invoked. Rational Application Developer has the option of caching the connection factory when using the wizards to build the service proxy or EJB which is used to access the generated CICS ECI service.

Reduction of data transmitted

The performance of the CICS TG has been shown to be inversely proportional to length of data transmitted, suggesting that the limiting factor is the path length of the communications stack. Figure 7-14 plots throughput against COMMAREA size, for an ECI based Java client calling a CICS program. Throughput substantially decreases when message ECI COMMAREA length increases.

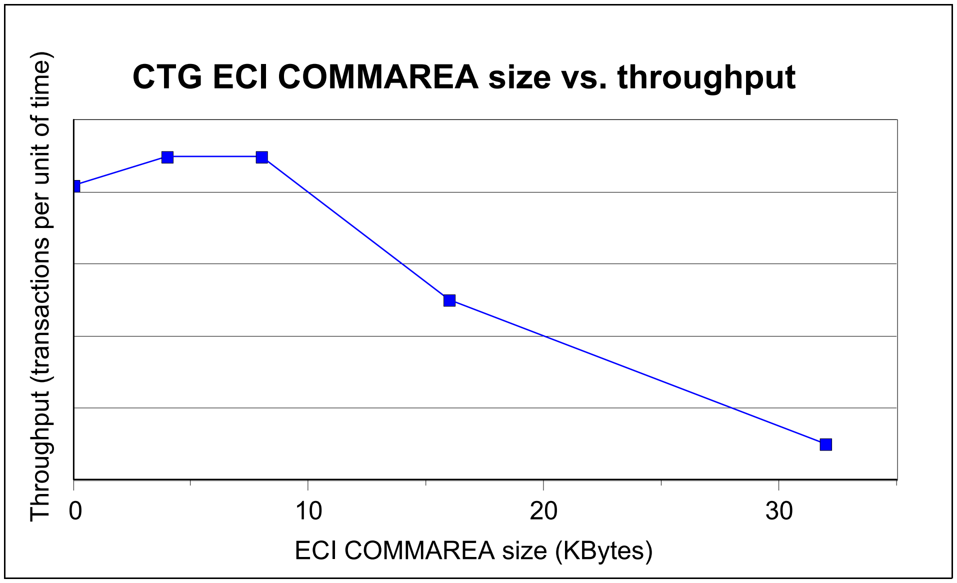


Figure 7-14 Throughput versus COMMAREA size for a CICS TG ECI workload

There are several techniques for reducing data flowed through the CICS TG, but broadly speaking there are three ways of addressing the problem.

- The application should be designed so that it has the minimum number of data flows from the service requester through to the CICS server. Options may be limited by the existing interface offered by the CICS application, and the ability to re-engineer these interfaces.
- The application should be designed to transmit only the data essential to the CICS TG, that is, only the data directly required by the business logic.
- The data flowed across the network can be compressed or truncated.

Data *truncation* facilities are built into the CICS TG and CICS client/server flows and can be invoked as follows:

- The CICS Transaction Gateway in combination with the CICS servers, provide built-in truncation for ECI flows. Any trailing nulls are not physically passed across the network between client and server. This truncation is automatic and not configurable. Thus you should design your CICS COMMAREAs to be padded with trailing nulls and to store data efficiently in the beginning of the COMMAREA.
- Outbound data from the JCA ECI resource adapter is automatically truncated. This truncation is based on the length of the input data in the CCI record.

- Inbound data to the JCA ECI resource adapter can be restricted using the `setReplyLength()` method on the `ECIIntersactionSpec`. This controls the amount of data returned from the CICS LINK and should only be used if the size of the data required is known in advance. It is not normally required if the COMMAREA data is null truncated, as in this case the CICS null truncation routines will be automatically invoked.

Data *compression* is applicable when the system is network I/O bound and yet still has spare CPU cycles. This is likely to happen if many clients are trying to transfer large amounts of data via the CICS TG, for instance, multiple ECI applications transferring 32 KB COMMAREAs to the CICS server over a low bandwidth network. When using a remote Java application it is possible to compress data as it flows from the Java client to the Gateway daemon and as it flows via SNA from the Gateway daemon to the CICS server.

- The CICS TG security exits can be used to compress data instead of, or as well as, encrypting data. The data is compressed as it leaves the Java client application, and is uncompressed as it enters the Gateway daemon. Examples of how to use these exits for data compression are given in the `ClientCompression.java` and `ServerCompression.java` samples in the `samples\Java\com\ibm\ctg\security` directory, and a working example is given in *Revealed! CICS Transaction Gateway with More CICS Clients Unmasked*, SG24-5277.
- The IBM Communications Servers provide compression on LU6.2 RU packets using the Run Length Encoding (RLE) or Lempel_Ziv (LZ) algorithms. For more details on which algorithms are supported on each platform, refer to the platform-specific IBM Communications Server documentation.

It is best to compress the data as early as possible in its life cycle to reduce the flows through the various components. Data encryption, where used, should be performed after any compression routines for reasons of efficiency.

7.3.3 Topology decisions

In this section we will review the three different topologies which can be used with the CICS TG and analyze the network considerations, system limits, and the workload management possibilities for each solution.

WebSphere Application Server and CICS TG deployed on distributed platforms

Figure 7-15 illustrates the use of WebSphere Application Server and the CICS TG on a distributed platform (topology 1).

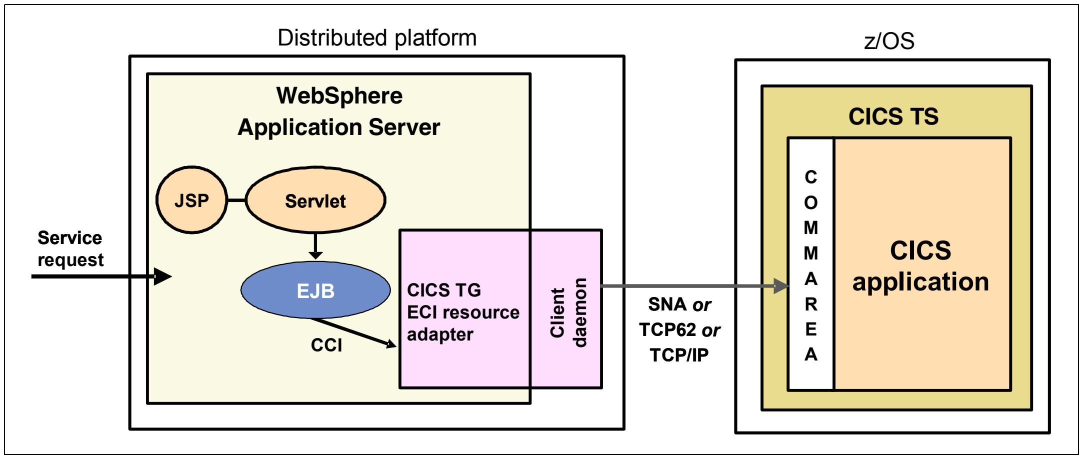


Figure 7-15 CICS TG topology 1

Network considerations

When the CICS TG is deployed on a distributed platform the network connectivity from the distributed platform to CICS TS is provided by the Client daemon component of the CICS TG. The Client daemon flows the ECI request to the CICS server using a SNA connection, TCP62 connection or TCP/IP directly.

As a general principle, native methods (in this instance LU6.2 over SNA or native TCP/IP) should perform better than non-native methods (TCP62), due to the overhead incurred in translation and encapsulation when performing protocol switching. Also Communications Server for z/OS currently provides the ability for native TCP/IP communications to exploit the QDIO mode of OSA-Express adapters, providing higher bandwidth usage with gigabit ethernet networks. For further details refer to the IBM Redbook *OSA-Express Implementation Guide*, SG24-5948. In addition LU6.2 over SNA also utilizes more complex pacing algorithms within the communications stack, including DLC pacing, session level pacing and APPN high performance routing, which are not provided by TCP62.

Tip: Note that fundamental differences between SNA and IP, as well as the different approaches that are required for tuning, may in practice give different results for the same workload running over the same physical network. For a comprehensive summary of LU6.2 tuning, and comparative information about LU6.2 via SNA and IP protocols, see the *Communications Server for AIX Planning and Performance Guide,* SC31-8220*.*

Connection pooling

When the application is written using the JCA and deployed into WebSphere Application Server, pooling of connections is provided seamlessly by the pool manager component of WebSphere Application Server. This allows the reuse of connections to the resource adapter between multiple J2EE components, such as enterprise beans or servlets. The pool parameters are configured through the connection pool settings for the connection factory. Because the CICS TG and WebSphere Application Server both reside on the same host in this configuration, these pooled connections are actually a set of connection objects, each representing a local connection between WebSphere Application Server and the CICS TG. As a result, the connection pooling provides a benefit in terms of reduced memory and CPU utilization. The network connections from the CICS TG into CICS TS are managed and pooled independently by the CICS TG Client daemon component.

System limits

The principal system limits that are applicable to this topology are the threading limits within the WebSphere Application Server JVM™ environment, and the limits on concurrent requests within the CICS TG Client daemon.

WebSphere Application Server thread usage

The number of threads in the WebSphere Application server thread pool for running Java servlet or EJB components can be pre-configured within WebSphere Application Server. The trade-off between a thread bottleneck, and over-commitment of resources if too many threads are used, should be found by performance benchmarking. Generally speaking, low numbers of threads will give the best throughput when the calls to the CICS server execute quickly. Conversely, higher numbers of threads may give better throughput when using a multi-CPU architecture, or when CICS applications take longer to respond due to network delays or more complex CICS business logic. In WebSphere Application Server on distributed platforms, the thread settings are specific to the Web or EJB containers and are specified as follows:

EJB container	ORB Service -> Thread Pool -> Maximum Size
Web container	Web container -> Thread Pool -> Maximum Size

CICS TG concurrent requests

The maximum number of concurrent requests from the CICS TG to all connected CICS servers is controlled by the `MaxRequests` parameter in the CTG.INI file. This defaults to 256 and must be increased if more parallel requests need to be supported. In addition on the CICS server, the maximum number of active tasks is controlled by the MXT CICS SIT parameter. If your CICS transactions are short-lived, this is unlikely to be limiting. However, if your ECI applications use extended units-of-work or your 3270 CICS applications use conversational transactions, this could become a bottleneck.

Workload management

Workload Management is the process of spreading multiple requests for work over the resources that can do the work. It optimizes the distribution of processing tasks thus improving performance, scalability, and reliability of an application. It also provides failover when servers or systems are not available.

The following workload management techniques can be used with topology1:

- WebSphere Application Server workload management
- MVS TCP/IP port sharing
- Sysplex distributor
- CICSPlex SM
- CICS TG Workload Manager

WebSphere Application Server workload management

WebSphere Application Server Network Deployment implements workload management using *clustering.* A cluster is a set of application servers that are managed together and participate in workload management. Application servers participating in a cluster can be on the same *node* or on different nodes.

Cluster members are required to have identical application components, but can be sized differently in terms of weight, heap size, and other environmental factors. This allows large enterprise machines to belong to a cluster that also contains smaller machines.

Since Web applications often maintain state between different legs of a user dialog, you will need to consider the management of state data when designing a workload management solution. The CICS TG itself does not store application state, although state data may be stored within the application server or within CICS.

For further details on the workload management capabilities of WebSphere Application Server on the distributed platforms, including the techniques for state management, refer to *IBM WebSphere V5.1 Performance, Scalability, and High Availability WebSphere Handbook Series*, SG24-6198. This IBM Redbook discusses various options for scaling applications based on WebSphere Application Server Network Deployment V5.1.

MVS TCP/IP port sharing

TCP/IP port sharing is a feature provided by z/OS Communications Server which allows multiple address spaces (such as CICS regions) to listen for IP requests on the same port. It functions by allowing incoming socket open requests to be distributed among the listening address spaces according to both the socket backlog and current usage. Because the CICS TCP/IP sockets listener only utilizes a single socket for each connection from a CICS TG, TCP/IP port sharing can only be used to provide for failover of CICS regions in this topology and not for load balancing across regions.

Sysplex distributor

Sysplex Distributor offers the ability to load balance incoming socket open requests across different address spaces running on different IP stacks (usually on different LPARs). Because the CICS TCP/IP sockets listener only utilizes a single socket for each connection from the CICS TG Client daemon, Sysplex Distributor can only be used to provide for failover of CICS regions in this topology and not for load balancing across CICS regions.

CICSPlex SM

The CICS dynamic routing program allows for the dynamic routing of LINK commands or 3270 transaction requests. This provides the ability for applications to be dynamically routed within the CICSplex.

CICS TG Workload Manager

The CICS TG on Windows incorporates its own Workload Manager. This allows simple round robin or a biased distribution of non-extended ECI requests across configured CICS servers. Workload management is configured on a per program basis. Both biasing and round-robin methods are able to provide failover. Should requests to a region unexpectedly fail, requests are directed to another region. The unavailable region is bypassed until a region time-out value is reached, at which point, if it is available, the region is returned to the pool of available regions. If the CICS TG Workload Manager is used, we recommend the use of the round-robin method, as it is a simpler scheme to configure, and it is not easy to calculate the correct relative weights to use in a biased scheme.

Remote Gateway daemon on z/OS

Figure 7-16 illustrates the use of WebSphere Application Server on a distributed platform and the CICS TG daemon on z/OS (topology 2).

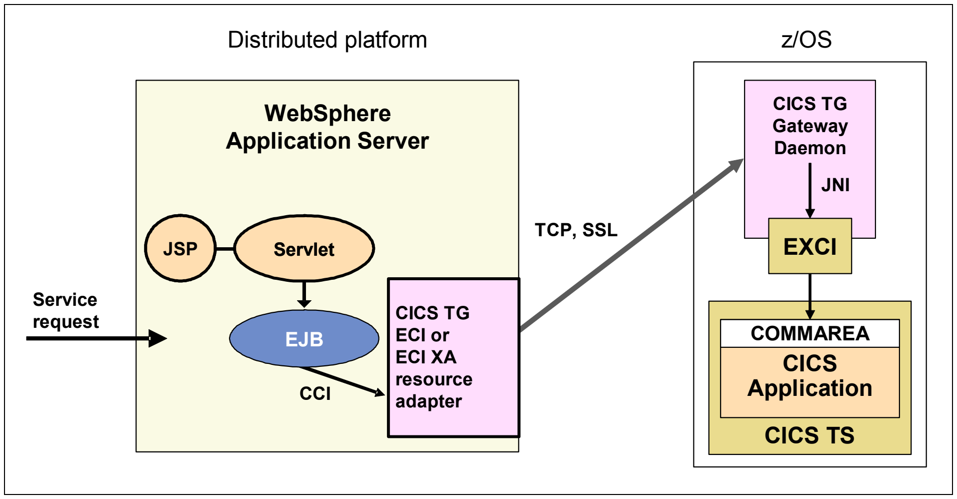


Figure 7-16 CICS TG topology 2

Network considerations

When the CICS TG is deployed on the z/OS platform and the CICS ECI resource adapter is used in a distributed WebSphere Application Server, network connectivity from the distributed platform to System z is provided by CICS TG Gateway daemon on z/OS. This supports two protocols: TCP and SSL variants.

Connection pooling

The connection pool represents physical network connections between WebSphere Application Server and the Gateway daemon on z/OS. In this configuration it is essential to have an efficient connection-pooling mechanism otherwise, a significant proportion of the time from making the connection to receiving the result from CICS and closing the connection can be in the creation and destruction of the connection itself. The JCA connection pooling mechanism mitigates this overhead by allowing connections to be pooled by the WebSphere Application Server pool manager.

SSL considerations

When using the CICS TG on z/OS the secure SSL network protocol can be used to secure network flows into System z. The CPU cost of SSL is predominantly in the initial handshake phase, as this uses public key/private key encryption, which is more expensive than the secret key encryption used for SSL data transmission. There are several means of reducing the cost of SSL encryption with the CICS TG; these are summarized here:

- Persistent connections

  When you use a persistent connection, a subsequent connection re-uses a previously opened TCP/IP socket. This ensures that after the initial SSL handshake, no other handshake is performed until the persistent socket connection is closed. If using an SSL connection from a JCA application in WebSphere Application Server the Pool Manager in WebSphere Application Server will automatically provide this function.

- Encryption algorithms

  Different SSL key sizes (1024 or 512 bit) used in the SSL handshake, and the different SSL data encryption algorithms (DES,TDES, RC4, and so forth), have considerably different performance characteristics.

Gateway daemon thread usage

The Gateway daemon is a sophisticated multi-threaded Java application. It can handle multiple requests simultaneously and uses a set of properties configured in the `CTG.INI` configuration file. Two pools of threads can be configured, the *ConnectionManager* threads and the *worker* threads. For each connected client, one ConnectionManager thread is used in the CICS TG, and is held until the Java client issues a disconnect. In order for an ECI call to be performed via an allocated ConnectionManager thread, a thread must be allocated from the worker thread pool for the duration of the request. This relationship is summarized in Figure 7-17 on page 226.

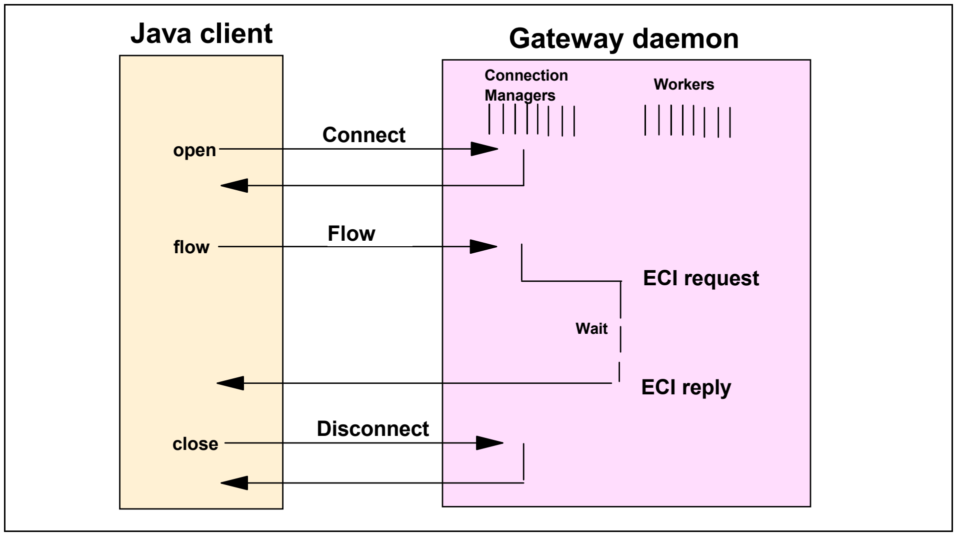


Figure 7-17 CICS TG threading model

The ConnectionManager threads limit the maximum number of connected Java clients, while the worker threads limit the number of concurrent ECI calls that can be issued by these attached clients. The initial and maximum numbers of these ConnectionManager and worker threads are set in the `CTG.INI` file. Requests can be timed out if a ConnectionManager or worker thread does not become available within a specified time, or if the Gateway daemon detects that a client is idle or is not responding.

When using the CICS ECI JCA resource adapter the connection pool is controlled by the pool manager component of WebSphere Application Server. The WebSphere Administrative Console provides an administrative interface to control the number of managed connections in the pool.

Since the Gateway daemon on z/OS runs as a Unix System Services (USS) application the USS system limits will affect the amount of storage, sockets and threads the Gateway daemon JVM process can utilize. Refer to the manual *UNIX System Services Planning*, GA22-7800, for more details on these limits.

Support for large numbers of threads within a JVM is limited by storage requirements per thread and the practical limitations on multi-threading with a given process. On most 32-bit JVMs (or the 31-bit z/OS JVM) the Gateway daemon will be limited to a practical maximum of around 5,000 threads in total.

When you configure a connection to a remote Gateway daemon, we advise you to use the following general rules to configure the thread pools.

- The maximum number of Gateway daemon ConnectionManager threads (*maxconnect*) should be the same as the maximum number of managed connections in the JCA connection pool (*max connection*)
- Gateway daemon *minconnect* is equal to pool manager *min connection*
- The maximum number of Gateway daemon worker threads (*maxworker*) should be set to less than or equal to the number of connection manager threads (*maxconnect*)
- The longer the response time of your CICS applications the more worker threads you should define in the Gateway daemon. This will help to prevent the Java client applications being suspended while trying to obtain a managed connection.

In addition both the pool manager and the Gateway daemon have a set of timeout parameters to control how long connections are maintained. The pool manager settings are as follows:

Aged timeout	The total age of a managed connection from initial creation to the current time.
Reap time	The interval at which the reaper (timer) thread runs.
Unused timeout	The amount of time that a given managed connection can be inactive in the free pool before being closed.

The Gateway daemon settings are as follows:

closetimeout	How long a connection manager thread waits for work to finish on a disconnected client.
connecttimeout	How long the protocol handler waits to obtain a connection manager thread from the connection manager thread pool.
idletimeout	How long a connection is allowed to remain dormant before it is closed.
pingfrequency	How often the Gateway daemon will ping a remote Java client.
sotimeout	How frequently the protocol handler wakes from accepting inbound connections.
solinger	The TCP/IP SO_LINGER value for any socket used by this connection
workertimeout	How long a connection manager thread will wait to obtain a worker thread from the worker thread pool

We recommend that the timeout settings are specified so that timeouts are triggered as close as possible to the component that will receive the error, that is the J2EE component in WebSphere Application Server. Bearing this in mind we recommend the following settings:

- Pool manager *reap time* < *unused timeout*
- Pool manager *unused timeout* < Gateway daemon *idletimeout*
- Pool manager *aged timeout* can be disabled, unless you require that connections are closed down after a certain age
- Pool manager *connect timeout* < Gateway daemon *connecttimeout*, so that a timeout obtaining a connection to the CICS TG will be triggered from the pool manager.

EXCI pipe usage

On z/OS the CICS TG uses the facilities of the External CICS Interface (EXCI) to flow ECI requests into CICS. When using the EXCI, CICS currently limits each client address space to 250 simultaneous sessions (pipes) to all attached CICS regions.

Note: The EXCI pipe limit can be set between 100 and 250 using the LOGONLIM parameter within SYS1.PARMLIB. For CICS TS V2.3 and CICS TS V2.2 an APAR must first be supplied. Prior to CICS TS V2.2 the value is fixed at 100.

An attempt to use more than 250 pipes per address space will be rejected by the EXCI. The 250 pipe limit is usually sufficient to provide the required throughput although the following points should be considered when designing a scalable CICS TG solution:

- The maximum number of worker threads (*maxconnect* in the CTG.INI) should be set to less than or equal to 250, and less than or equal to the number of sessions defined on the EXCI connection.
- For reasons of performance, once a call has been made to a CICS region from a given thread the CICS TG will keep the pipe allocated to that particular CICS region for the lifetime of the thread, or until the CICS region terminates or closes IRC.
- If a pipe shortage situation is encountered then multiple Gateway daemons can be used and work shared across them using z/OS IP workload balancing solutions such as TCP/IP port sharing or Sysplex Distributor. For more details refer to“Workload management” on page 229.

Workload management

The CICS TG on z/OS is designed for use with the z/OS IP load balancing topologies, due to its usage of multiple socket connections. The following z/OS IP load balancing technologies can be utilized:

- MVS TCP/IP port sharing
- Sysplex distributor
- CICSPlex SM

Workload management with a CICS TG on z/OS is illustrated in Figure 7-18.

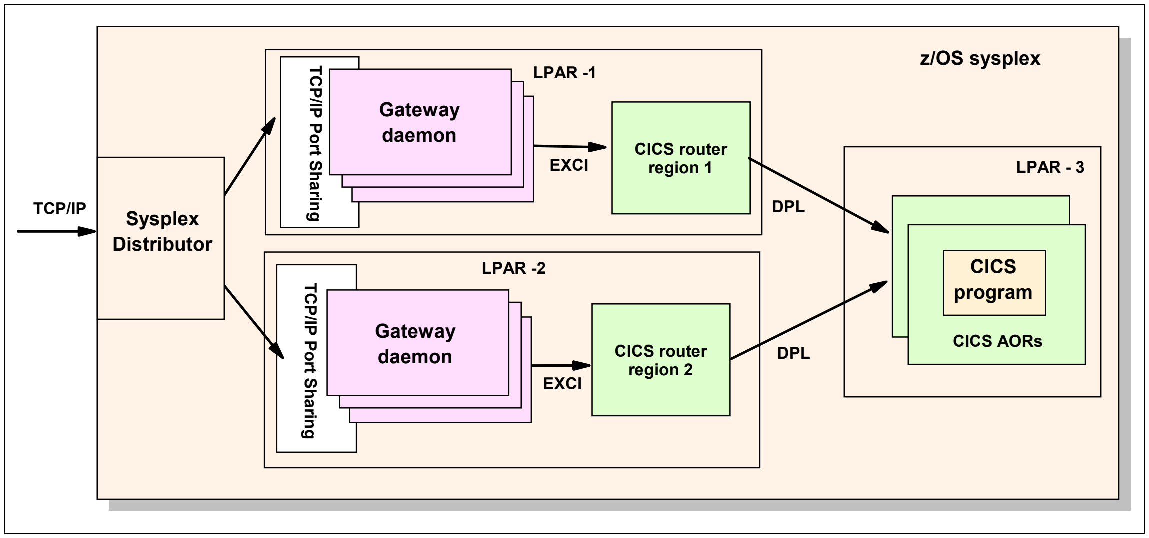


Figure 7-18 Workload management with a CICS TG on z/OS

MVS TCP/IP port sharing

TCP/IP port sharing is a feature provided by z/OS Communications Server which allows multiple address spaces (such as Gateway regions) to listen for IP requests on the same port. It functions by allowing incoming socket open requests to be distributed among the listening address spaces according to the socket backlog and current usage. It is a very simple feature to implement, and provides effective load balancing across multiple Gateway regions listening on the same TCP/IP stack in the same LPAR.

Sysplex distributor

Sysplex Distributor offers the ability to load balance incoming socket open requests across different address spaces running on different IP stacks (usually on different LPARs). The routing decision is based on real-time socket status, and MVS Quality of Service (QoS) criteria. This provides the benefit of balancing work across different MVS images, providing enhanced scalability and failover in a z/OS Parallel Sysplex.

The use of both of these workload management techniques is shown in Figure 7-18 on page 229.

When using these techniques, it is important to be aware of a potential affinity-related problem which might be created. For example, in Figure 7-18 on page 229, if a request for CICS router region1 is routed by Sysplex Distributor to LPAR-1 then a cross-memory (XM) EXCI connection is used between the Gateway daemon and the CICS router. However, if the request is instead routed to LPAR-2 then a cross-system coupling facility (XCF) EXCI connection is used. The performance and transactional characteristics of these types of connection are different, and certain restrictions apply to XCF group membership; therefore, it is preferable to use MRO XM connections.

It is possible to use the EXCI user exit DFHXCURM to resolve this affinity problem. DFHXCURM is able to dynamically change the APPLID of the CICS router such that all requests which are routed to the Gateway daemons running on LPAR-1 are passed to CICS router 1 and all requests which are routed to the Gateway daemons on LPAR-2 are passed to CICS router 2.

CICSPlex SM

The CICS dynamic routing program allows for the dynamic routing of LINK commands; this provides the ability for applications invoked by ECI requests to be dynamically routed within the CICSplex. Note that due to transactional limitations, the EXCI connection from the Gateway daemon must be to a CICS region in the same LPAR, although further XCF links to CICS regions across the CICSPlex are supported (Figure 7-18 on page 229).

WebSphere Application Server and CICS TG deployed on System z

Figure 7-19 on page 231 illustrates the use of WebSphere Application Server and the CICS TG deployed on z/OS (topology 3a).

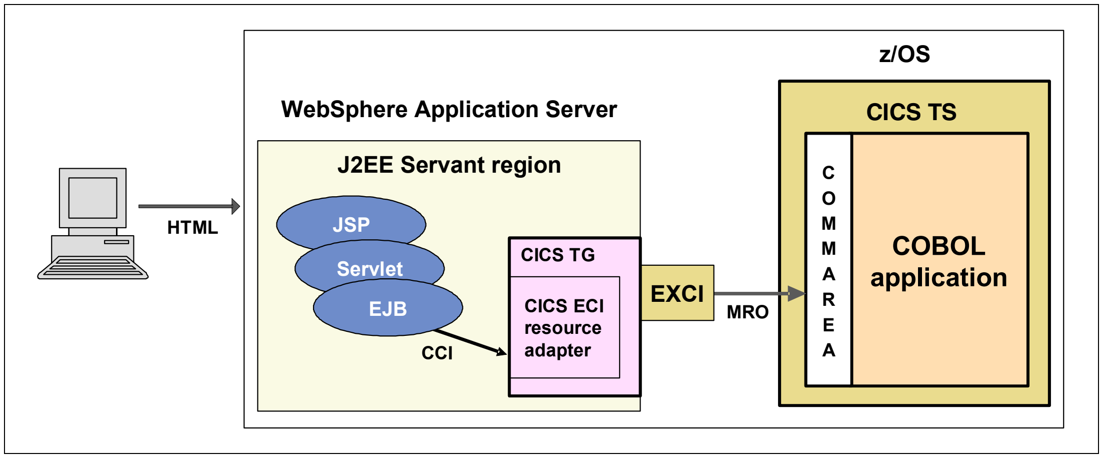


Figure 7-19 CICS TG topology 3a

Network considerations

As in topology 1, the most common WebSphere Application Server for z/OS configuration makes use of a local Gateway. On z/OS, this results in a direct cross-memory EXCI connection between the application server and CICS. Note that due to transactional limitations, the EXCI connection from the Gateway daemon must be to a CICS region in the same LPAR, although further XCF links from the initial linked-to CICS region to other CICS regions across the CICSPlex are supported (Figure 7-20 on page 233).

WebSphere Application Server thread usage

The number of threads in a J2EE servant region can be pre-configured within WebSphere Application Server. (See “Workload management” on page 233 for an explanation of what a servant region is). The trade-off between a thread bottleneck, and over-commitment of resources if too many threads are used, should be found by performance benchmarking. Generally speaking, low numbers of threads will give the best throughput when the calls to the CICS server execute quickly. Conversely, higher numbers of threads may give better throughput when using a multi-CPU architecture, or when CICS applications take longer to respond due to network delays or more complex CICS business logic. In WebSphere Application Server for z/OS, the number of threads in a J2EE servant regions is controlled by the server workload profile. This can take the following values:

IOBOUND	3 x the number of CPUs (max of 3x32=96)
CPUBOUND	The number of CPUs online (max 24)
ISOLATE	1 thread
LONGWAIT	40 threads

EXCI pipe usage

Within WebSphere Application Server for z/OS the CICS TG uses the facilities of the external CICS interface (EXCI) to flow ECI requests into CICS. When using the EXCI, CICS currently limits each client address space to a limit of 250 simultaneous sessions (pipes) time to all attached CICS regions.

An attempt to use more than 250 pipes per address space will be rejected by the EXCI. The 250 pipe limit is usually sufficient to provide the required throughput although the following points should be considered when designing a scalable CICS TG based solution:

- The maximum number of JVM threads within a J2EE servant region is defined in the server workload profile, and has an upper maximum of either 40 or 96, depending on the setting of the application server workload profile. For reasons of performance, once a call has been made to a CICS region from a given thread the CICS TG will keep the pipe allocated to that particular CICS region for the lifetime of the thread, or until the CICS region terminates or closes IRC.
- The JCA connection pool managed by WebSphere Application Server is a set of local connection objects which map on the CICS TG JavaGateway connections. These connection objects do *not* map onto the EXCI pipes allocated by the CICS TG. Instead EXCI pipes are allocated directly by the JVM threads within the J2EE servant region.
- An EXCI pipe shortage situation will only occur if multiple JCA connection factories are used naming different CICS regions. This has the potential to cause pipes to become allocated to *multiple* CICS regions from a single thread. In this instance you should configure the J2EE servant region to have fewer threads, and then you use multiple servant regions to obtain the desired throughput. Alternatively you can utilize the EXCI user exit DFHXCURM to cause all EXCI calls to go to the same local CICS routing region, before being forwarded onto the CICS AOR, using CICSPlex SM or another routing technology (Figure 7-20).

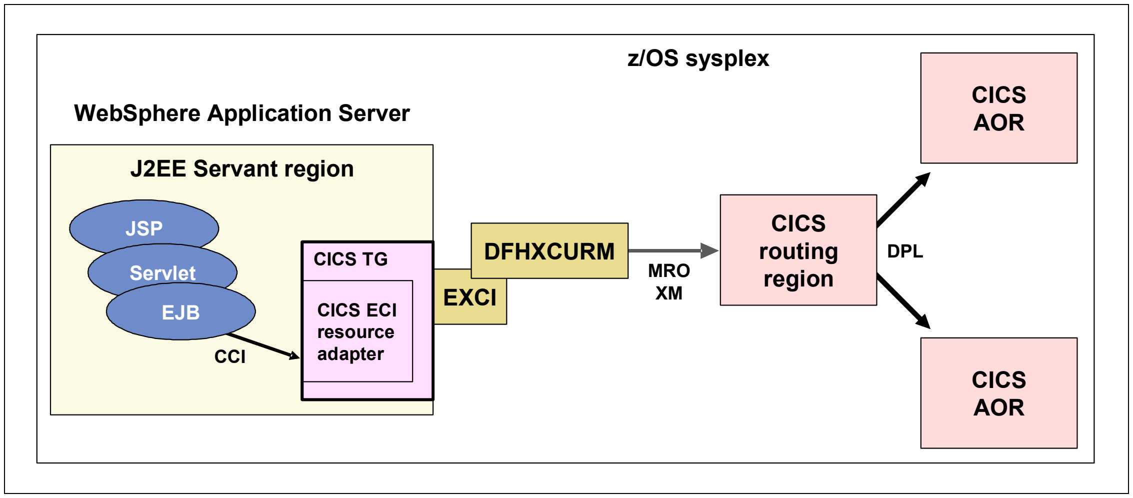


Figure 7-20 DFHXCURM usage with WebSphere Application Server on z/OS

Workload management

WebSphere Application Server for z/OS takes full advantage of the workload management and high availability features provided by z/OS Parallel Sysplex and System z.

The following workload management techniques can be used with topology 3a:

- WebSphere Application Server workload management
- MVS TCP/IP port sharing
- Sysplex distributor
- CICSPlex SM

WebSphere Application Server workload management

The application server on z/OS (also known as a *J2EE server*) is made up of two different address spaces:

Servant Region The servant region runs the Web and EJB containers.

Controller Region The controller region controls and routes work into the servant regions.

A J2EE server is made up of one controller region and one or more related servant regions (see Figure 7-21). The controller region controls and routes work into the servant regions based on the performance goals defined in MVS Workload Manager (WLM). These goals may be based on a number of criteria, including the caller's user ID or the type of caller, for example, an intranet user or an Internet user. The control region can dynamically start and stop servant regions based on the needs of the workload.

The J2EE server lives within the boundaries of a z/OS image. Since the work runs in multiple address spaces and on multiple physical processors, MVS WLM can differentiate and prioritize work based on Service Level Agreements allowing MVS WLM to manage the workload.

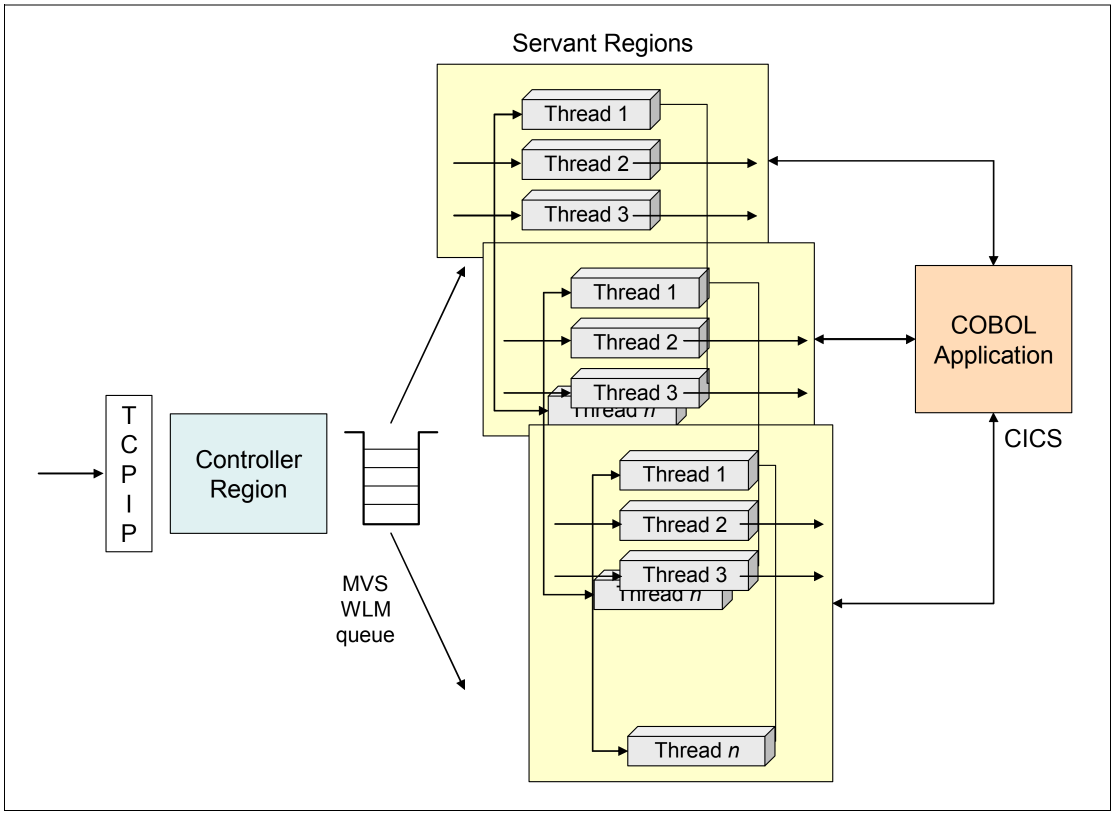


Figure 7-21 WebSphere Application Server for z/OS and MVS WLM

As on the distributed platforms, WebSphere Application Server for z/OS also support *clustering.* A cluster is a logical grouping of replicated J2EE server instances, allowing you to partition your workload and still see it as one entity. The server instances belonging to a cluster can run on the same z/OS image, or can span more images in a sysplex.

From the client point of view, the cluster is an entity, and the client should not know which server instance will perform the given request. The cluster may span multiple z/OS images in a sysplex, therefore providing a horizontal scaling capability.

The clustering capabilities of a WebSphere Application Server Network Deployment also allow for nondisruptive changes to software components and enable continuous application availability in the event of unexpected failures.

MVS TCP/IP port sharing

TCP/IP port sharing is not normally required with this topology because the workload management features of WebSphere Application Server for z/OS can be used instead.

Sysplex distributor

Sysplex Distributor offers the ability to load balance incoming socket open requests across different J2EE servers running on different LPARs. The routing decision is based on real-time socket status, and MVS quality of service criteria. This provides the benefit of balancing work across different MVS images, providing enhanced scalability and failover in a z/OS Parallel Sysplex.

CICSPlex SM

This topology supports the use of CICS dynamic program routing which provides the ability for CICS applications invoked by ECI requests to be dynamically routed within the CICSplex. Note that due to transactional limitations, the EXCI connection from the WebSphere servant region must be to a CICS region in the same LPAR, although further XCF links to CICS regions across the CICSplex are supported

The use of these workload management techniques in combination is shown in Figure 7-22 on page 235.

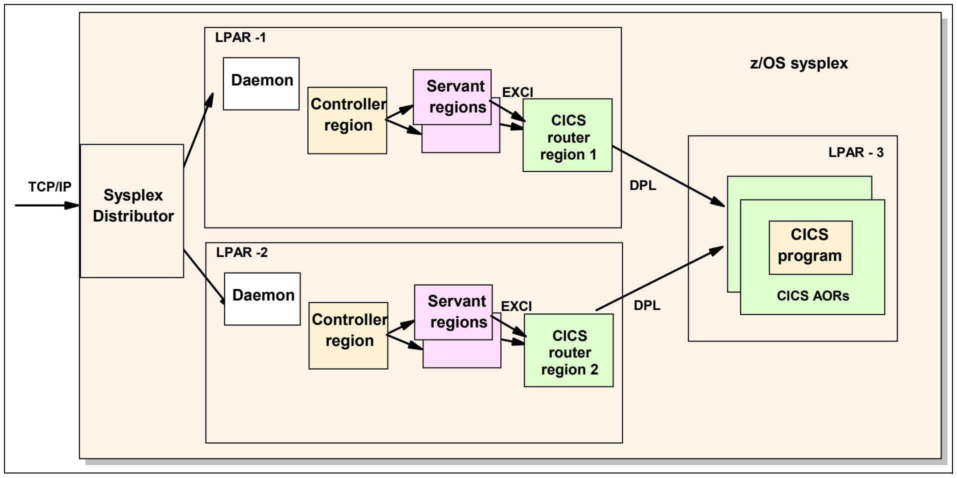


Figure 7-22 Workload management with WebSphere Application Server for z/OS

For further details on the workload management capabilities of WebSphere Application Server for z/OS, including the techniques for state management, refer to *Architecting High Availability Using WebSphere V6 on z/OS*, SG24-6850.

7.4 CICS Web support

CICS Web support refers to the ability of a CICS region to process HTTP requests from Web browsers. This section reviews the major design choices which will have to be made when architecting the CICS application, and how they are likely to affect the performance of the solution. This includes looking at SSL and non-SSL communication.

7.4.1 Transactions and TCBs

There are up to three tasks involved in processing an HTTP request.

- CICS TCP/IP Listener task (CSOL)
- CICS Web attach task (CWXN)
- CICS Web Alias task (CWBA or a user defined transid)

The CICS TCP/IP listener task (CSOL) runs as a CICS system task listening for socket requests. You can configure the listener task to listen on multiple ports by configuring multiple TCPIPSERVICE definitions. When CSOL receives an HTTP request, it attaches the Web attach transaction (CWXN) to continue processing the HTTP request. CWXN analyzes the HTTP request and then starts an alias transaction (CWBA) to continue processing the HTTP request and return the HTTP response to the Web browser. If the HTTP request matched a URIMAP and was for static content then CWXN will return the HTTP response without attaching an alias task.

To support this architecture, CICS uses some additional Task Control Blocks (TCBs). The SL TCB is used by CSOL to process new connections and the notification of data arriving to satisfy an outstanding asynchronous receive. The SO TCB is used by CWXN and the alias task to send and receive data over the socket connection. There is also a pool of S8 TCBs used for SSL processing. They get used by CWXN or the alias task for the SSL handshake and to encrypt and decrypt data being sent and received. The S8 TCBs are used for the duration of the function they are performing and returned to the pool afterwards. This differs from previous CICS releases where the S8 TCB was owned by the associated task until the task terminated.

The use of multiple TCBs means that the additional TCP/IP socket processing incurred by using the CICS TCP/IP listener does not run within the main CICS TCB, and so will not affect the processing of requests within the Quasi

Re-entrant TCB (QR TCB), on which the majority of CICS business logic is processed.

Since the HTTP requests are processed directly within the CICS address space, it is not surprising that performance tests have shown this to be a very efficient and scalable means of accessing CICS applications from the Web. (For further details, refer to Chapter 5 in the redbook *A Performance Study of Web Access to CICS*, SG24-5748). However, you need to consider that processing HTTP requests does add an extra load to your CICS region, and you may want to separate the work into dedicated *Web-owning* regions.

7.4.2 COMMAREA interface or 3270 interface?

When designing your CICS Web support architecture, you first need to know if you have separate presentation and business logic within your CICS applications. A call to the business logic is considerably more efficient than accessing a CICS application using the services of the 3270 Web bridge.

Simple Web workloads have demonstrated that using a Web-aware application design will be at least twice as efficient in terms of z/OS CPU usage as the 3270 bridge route. This approach also offers the advantage that you will no longer be tied to the original 3270 application design and can directly access the business logic in your CICS application.

7.4.3 Designing an efficient Web application

URIMAPs should be used to map HTTP requests to CICS resources. Avoid using an analyzer program if at all possible. If an analyzer program is required then try to avoid performing any processing that is likely to invoke delays, such as allocating storage, ENQs, or disk I/O requests. If you use a converter program program instead of a normal Web-aware application program then the efficiency of later processing can also be improved if the Decode method of the converter sets the `decode_input_data_len` to the exact length of the data to be passed to the business logic in the COMMAREA.

7.4.4 Improving HTTP performance

The overhead of the HTTP protocol can be significantly reduced by enabling HTTP KeepAlive (persistent connections); this provides for long-lived HTTP connections. These long-lived connections allow multiple HTTP requests to be sent over the same TCP socket connection, instead of the usual stateless HTTP model whereby every HTTP request opens a new TCP/IP socket. With HTTP/1.1, persistent connections are the default. As such, in CICS TS 3.1 it is recommended to set SOCKETCLOSE on the TCPIPSERVICE definition to NO to take full advantage of this. Unlike previous releases of CICS, an active CWXN task is not required to wait for the client to send in another request. Asynchronous receives are used in the same manner as us are done for IIOP and ECI over TCPIP.

The following charts show CPU and storage usage for HTTP based transactions. The data is for inbound HTTP transactions. It does not cover the outbound HTTP support. Comparison is also shown between CICS TS 3.1 which uses HTTP/1.1 persistent connections and CICS TS 2.3 with and without using HTTP/1.0 persistent connections.

Figure 7-23 on page 238 shows the amount of storage used in CICS to maintain an HTTP connection. This is CICS internal storage in the sockets and Web domains and does not include user application storage. The amount is approximately 4.8K per connection.

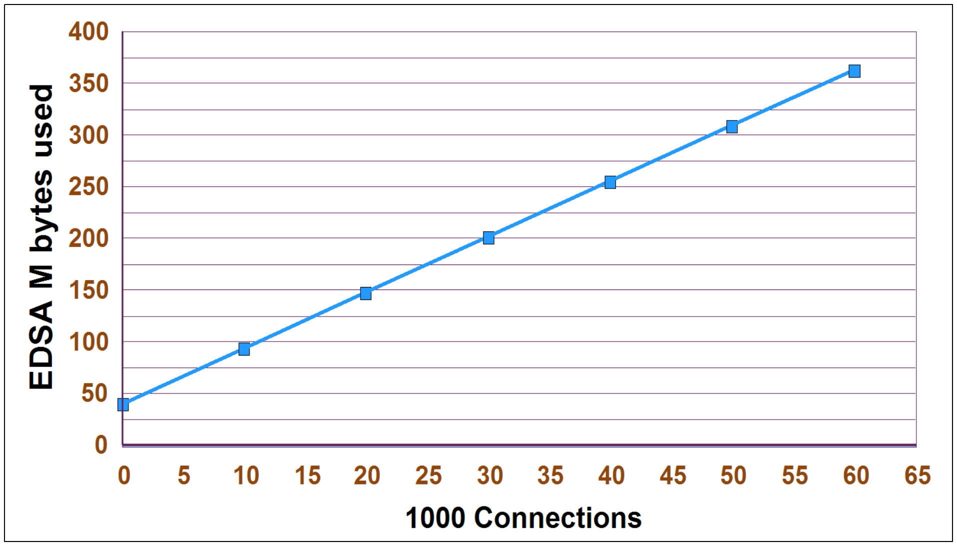


Figure 7-23 Storage usage per HTTP connection

Figure 7-24 shows a comparison in the CPU cost between CIC TS 3.1 and CICS TS 2.3. Two measurements were made for CICS TS 2.3; one with persistent sessions and one without. The values relate to applications processing the HTTP request and response in the COMMAREA.

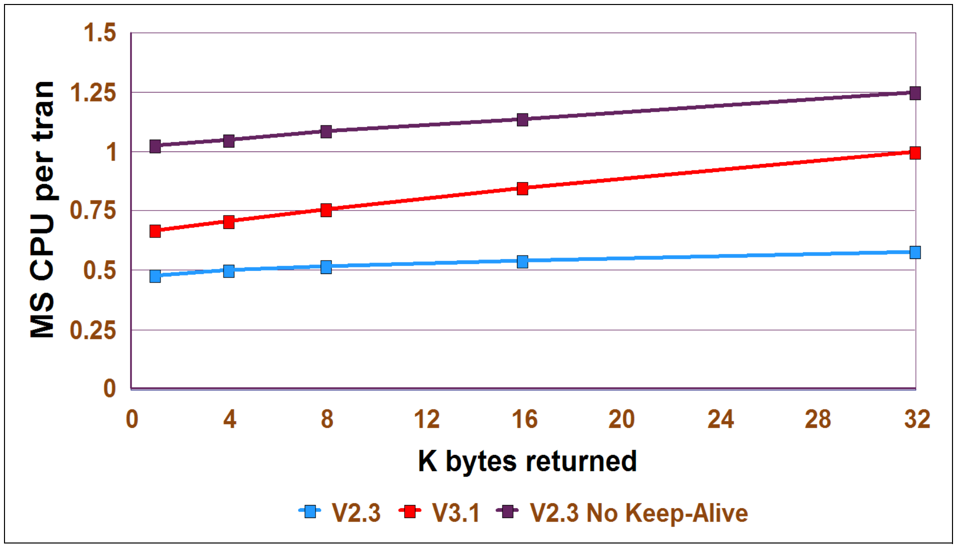


Figure 7-24 CPU per transaction (COMMAREA based)

Figure 7-25 shows a comparison in the CPU cost between CIC TS 3.1 and CICS TS 2.3. Two measurements were made for CICS TS 2.3; one with persistent sessions and one without. The values relate to applications processing the HTTP request and response using the **`EXEC CICS WEB`** and **`DOCUMENT API`** commands.

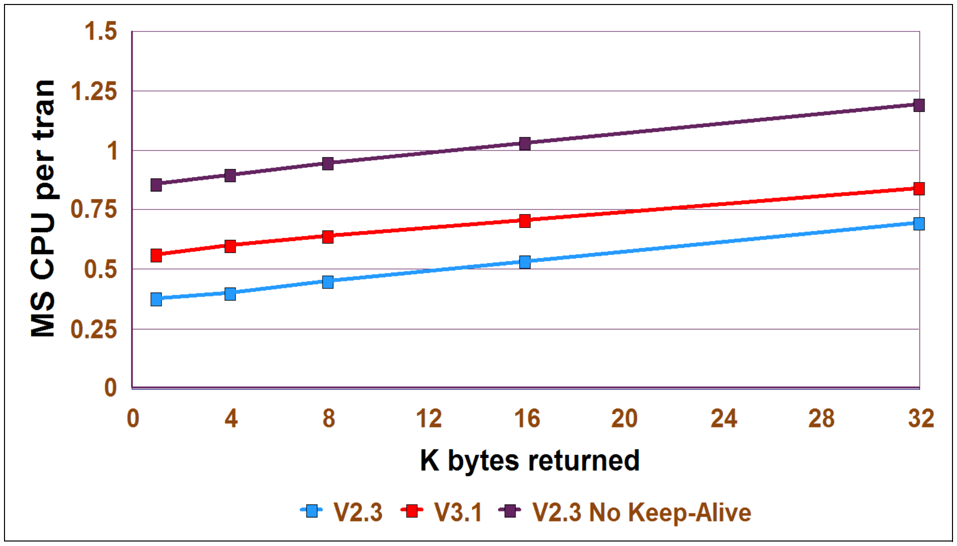


Figure 7-25 CPU per transaction (WEB API less than 32K)

In both Figure 7-24 on page 239 and Figure 7-25, CICS TS 3.1 is shown to be using more CPU per transaction than an equivalent transaction run with persistent connections in CICS TS 2.3. The increase is about 3 to 4% on a 2.5 ms transaction. However, CICS TS 3.1 allows up to 65,000 concurrent connections, where as CICS TS 2.3 required an active task for each connection and so could only support 999 at most with a realistic usable value being much lower than that.

Figure 7-26 on page 240 shows a comparison of the amount of CPU time used when sending very large HTTP responses. It can be seen that for very large HTTP responses CICS TS 3.1 actually uses less CPU time than an equivalent transaction running on CICS TS 2.3.

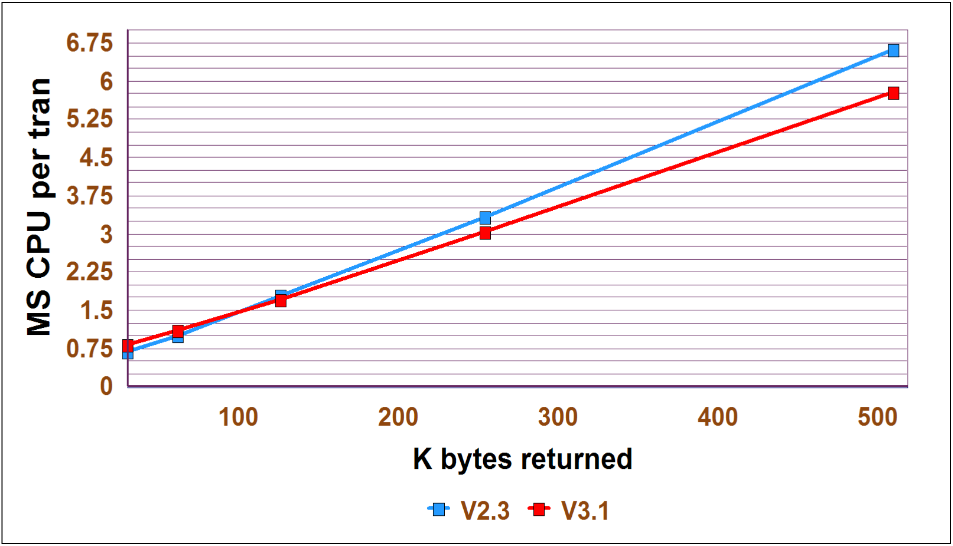


Figure 7-26   CPU per transaction (WEB API over 32K)

If you are using HTTPS (HTTP via SSL) to secure the Web connection to CICS Web support, then the cost of SSL encryption can be reduced by the following means:

- **Persistent HTTP connections**

  Use of a persistent HTTP connection, whereby a subsequent HTTP connection re-uses a previously opened persistent TCP/IP socket connection, ensures that after the initial SSL handshake, no other handshake is performed until the persistent HTTP connection is broken, which will usually only occur when the HTTP connection is timed out by the server.

- **SSL session ID reuse**

  Session ID reuse allows the Web client and server to communicate with a shortened SSL handshake by allowing the client to reuse an SSL session ID without re-negotiating encryption keys with the server. When using the HTTP Server for z/OS, session ID re-use is controlled by the `SSLV3Timeout` and `SSLV2Timeout` directives. When using the CICS TCP/IP listener, session ID re-use is enabled using the SSLDELAY SIT parameter.

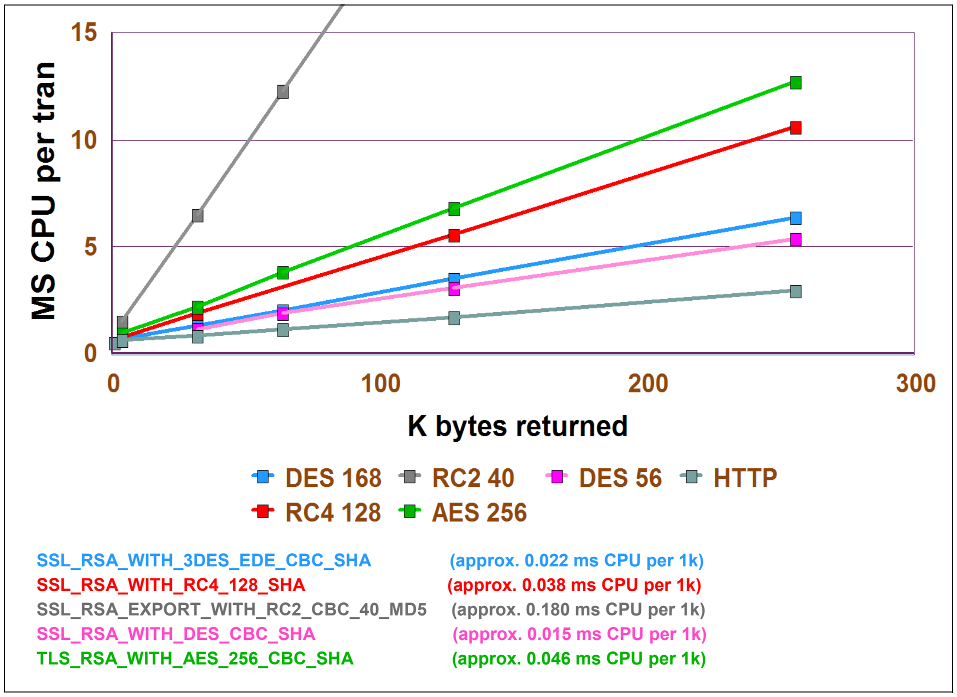
- **Cryptographic hardware**

  The Cryptographic Coprocessor feature is a hardware feature available on S/390® and zSeries processors. It consists of dual cryptographic module chips protected by tamper-detection circuitry and a cryptographic battery unit. It can be used to off-load CPU processing from the main CEC processors when performing cryptographic operations, and as such, can provide a significant reduction in CPU usage. The Cryptographic Coprocessor Feature can be used to reduce the CPU costs of SSL data transmission when using the DES or triple DES ciphers, and SSL handshaking when using the RSA PKCS#1 cipher. In order to use this hardware feature, the z/OS Integrated Cryptographic Service Facility (ICSF) has to be installed and operational.

- **Encryption algorithms**

  Different SSL key sizes (1024 or 512 bit) used in the SSL handshake, and the different SSL data encryption algorithms (DES, Triple DES, RC4, and so on) have considerably different performance characteristics. For full detail of the CPU costs when used with CICS Web support, refer to Chapter 6 “SSL with CWS” in the redbook *A Performance Study of Web Access to CICS*, SG24-5748.

Figure 7-27 shows the effect of different encryption algorithms on performance. Specifically CPU usage related to size of data.



SSL_RSA_WITH_3DES_EDE_CBC_SHA (approx. 0.022 ms CPU per 1k)
SSL_RSA_WITH_RC4_128_SHA (approx. 0.038 ms CPU per 1k)
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (approx. 0.180 ms CPU per 1k)
SSL_RSA_WITH_DES_CBC_SHA (approx. 0.015 ms CPU per 1k)
TLS_RSA_WITH_AES_256_CBC_SHA (approx. 0.046 ms CPU per 1k)

Figure 7-27 SSL encryption/decryption costs

7.4.5 Workload balancing

Utilizing CICS Web support to Web-enable your CICS application is quite likely to mean that the workload within your CICS regions will increase. CICS Web support makes use of multiple TCBs (SL, SO, S8, and QR), and thus it is able to concurrently utilize multiple processors in a multi-CPU LPAR. With the expansion in OTE and threadsafe support in CICS TS v3.1 it is now also possible to have significant business logic run on open TCBs

Work can be distributed across multiple CICS regions using workload balancing techniques. There are several different ways of balancing work across multiple CICS regions when using CICS Web support; we will provide a summary of them in the following sections. However, for a full analysis of the different options available, refer to the redbook *Workload Management for Web Access to CICS*, SG24-6118.

Dynamic routing of LINK commands

CICS allows for the dynamic routing of LINK commands. This includes the LINK to the COMMAREA driven business logic made from either the CICS Business Logic Interface or a Web-aware application program. This allows the user application running in an AOR to be accessed from a remote listener CICS region using a distributed program link (DPL) request; therefore enabling a CICS Web support workload to be workload-balanced across a z/OS sysplex in conjunction with the services of CICSPlex SM.

Note: If the target of the LINK is DFHWBTTA, DFHWBTTB, or DFHWBTTC (the 3270 Web Bridge) or the user program is using the `EXEC CICS WEB` or `TCPIP API` commands, then the LINKed to program cannot be in a remote region.

TCP/IP port sharing

TCP/IP port sharing provides a simple way of workload balancing HTTP requests across a group of cloned address spaces (including CICS regions or Web servers) running in the same z/OS image. As incoming client connections arrive for this port, TCP/IP will distribute them across the address spaces that are listening on the shared port. TCP/IP will select the address space with the least number of connections (both active and in the backlog) at the time that the incoming client connection request is received.

This allows you to workload-balance incoming HTTP requests across several cloned CICS regions or standalone HTTP server address spaces. The load balancing is based entirely on the number of IP connections, and so does not take into account the individual health or capacity of any given CICS region. However, it does provide a very simple means of providing failover and workload balancing within an LPAR.

Sysplex Distributor

Sysplex Distributor is implemented in OS/390 V2R10 and offers major enhancements to TCP/IP workload management in a sysplex. It has four main functions:

- Balancing of IP packets across multiple z/OS LPARs
- Close integration with MVS WLM
- Failover of the cluster IP address
- Re-routing of IP packets away from one node

As such, it offers a very efficient means of workload managing IP requests from within the heart of an z/OS Parallel Sysplex, in tight conjunction with MVS WLM and z/OS virtual IP addressing (VIPA). A more detailed discussion of Sysplex Distributor may be found in *TCP/IP in a Sysplex*, SG24-5235.

8

Application development

In this chapter, we look at each of the CICS enablement options from an application development point of view, and give you an overall description of the steps you need to follow to implement each option.

Before you embark on your chosen application development strategy, these are some of the questions you will need to consider:

- Do your developers have CICS or COBOL skills, or Java or Web skills?
- Do you want a hand-coded or tooled application development solution?
- Does the CICS application use a 3270 interface or does it have other interfaces, such as a communication area (COMMAREA) interface, or an APPC or socket interface?
- Is the CICS application based on a two-tier model with separate presentation and business logic, or are they intertwined?
- Does your application currently use a state control design that introduces inter-transaction affinities?
- Are you willing to invest in modifying and enhancing the CICS application?
- Do you want to develop a pure Java solution?
- Does your application need to be accessed by your external partners?

8.1 Development products overview

IBM provides high quality application development tools that are based around the Eclipse platform.

Eclipse is an open source, Java-based, extensible development project aimed at providing an integrated tool framework. At the heart of Eclipse is an extensive tool framework offering a set of core capabilities that supports extension through a plug-in architecture.

The following products are available, and are relevant to this chapter:

- WebSphere Developer for System z
- Rational Application Developer
- WebSphere Integration Developer

8.1.1 WebSphere Developer for System z

WebSphere Developer for System z consists of a common workbench and an integrated set of tools that support end-to-end, model-based application development, runtime testing, and rapid deployment of on demand applications. WebSphere Developer for System z V6 is based on the Rational Software Development Platform and facilitates the development of both Java- and z/OS-based applications. It includes capabilities that make traditional z/OS mainframe development, Web development, and integrated composite development faster and more efficient.

Upgraded XML and Web services support, enabling SOA access to CICS Transaction Server V3.1 and IMS V9 are now included.

More information about WebSphere Developer for System z can be found at:

`http://www.ibm.com/software/awdtools/devzseries/`

8.1.2 Rational Application Developer

Rational Application Developer is a development environment for creating J2EE enterprise applications targeted for WebSphere Application Server. It includes a test environment for WebSphere Application Server, and a full set of development wizards.

Rational Application Developer contains wizards to create connections to Enterprise Information Systems using J2EE Connector Architecture resource adapters. This includes support for using the CICS ECI resource adapter to connect to CICS TS.

More information about Rational Application Developer can be found at:

http://www.ibm.com/software/awdtools/studioappdev/

8.1.3 WebSphere Integration Developer

WebSphere Integration Developer is the development environment for building integrated business applications that are targeted for WebSphere Enterprise Service Bus and WebSphere Process Server. One of the primary purposes of WebSphere Integration Developer is to provide the appropriate tools to build and test Service Component Architecture (SCA) based solutions easily.

WebSphere Integration Developer provides the Enterprise Service Discovery for creating SCA imports and exports to connect with Enterprise Information Systems such as CICS TS.

More information about WebSphere Integration Developer can be found at:

http://www.ibm.com/software/integration/wid/

8.2 CICS Web services

In this section we introduce the tooling options available for transforming a CICS asset into a Web service endpoint. The main choice available is in the development tool utilized. The two options are:

- CICS Web Services Assistant
- WebSphere Developer for System z

The following section provides a high-level overview of how to transform an existing CICS asset into a Web service endpoint as either a provider or requester. We give a high level overview of the tooling available in the CICS Web Services Assistant and the XML Services for the Enterprise (XSE) capability of WebSphere Developer for System z.

8.2.1 CICS Web services assistant

The CICS Web services assistant is a set of batch utilities which can help you to transform existing CICS applications into Web services and to enable CICS applications to use Web services provided by external providers. The assistant supports rapid deployment of CICS applications for use in service providers and service requesters, with the minimum of programming effort.

When you use the Web services assistant for CICS, you do not have to write your own code for parsing inbound messages and for constructing outbound messages; CICS maps data between the body of a SOAP message and the application program's data structure.

The most important output from the CICS Web Services Assistant is the WSBind file. This file contains the metadata required to marshall and demarshall the XML of the SOAP request to and from the language structure of the application program. The metadata contained in the WSBind file is installed to an in-memory form when the corresponding WEBSERVICE resource is installed. This allows for very efficient marshalling and demarshalling of the SOAP messages from within core CICS code.

The assistant can create a WSDL document and WSBind file from a simple language structure, or a language structure from an existing WSDL document, and supports COBOL, C/C++, and PL/I. It also generates information used to enable automatic runtime conversion of the SOAP messages to containers and COMMAREAs, and vice versa.

However, the assistant cannot deal with every possibility, and there are times when you will need to take a different approach. For example:

- You do not want to use SOAP messages

  If you prefer to use a non-SOAP protocol for your messages, you can do so. However, your application programs will be responsible for parsing inbound messages, and constructing outbound messages.

- You want to use SOAP messages, but do not want CICS to parse them

  For an inbound message, the assistant maps the SOAP body to an application data structure. In some applications, you may want to parse the SOAP body yourself.

- The CICS Web services assistant does not support your application's data structure

  Although the CICS Web services assistant supports the most common data types and structures, there are some which are not supported or do not have simple obvious equivalents in XML, for example, OCCURS DEPENDING ON and REDEFINES on data description entries are not currently supported. For full details on the data types and structures supported by the CICS Web Services assistant, see the *CICS Web Services Guide*, SC34-6458.

  In this situation, you should consider one of the following alternatives:

  – Provide a wrapper program that maps your application's data to a format that the assistant can support.

  – Use WebSphere Developer for System z

Utility programs

The CICS Web Services Assistant provides two utilities:

- DFHLS2WS

  Create Web service artefacts from a program language structure. Will produce only provider Web services.

- DFHWS2LS

  Creates Web service artefacts from a WSDL Web service description. Will produce provider or requester Web services based on the presence or absence of the PGMNAME parameter.

DFHLS2WS

This program generates a Web services description and Web services binding file from a language structure. Example 8-1 shows sample JCL for running DFHLS2WS.

Example 8-1 DFHLS2WS JCL sample

```
//LS2WS JOB 'accounting information',name,MSGCLASS=A
//JAVAPROG EXEC DFHLS2WS
//INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.CICS.SDFHSAMP
REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
LANG=COBOL
PGMNAME=DFH0XCMN
URI=exampleApp/inquireSingle
PGMINT=COMMAREA
WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
WSDL=/u/exampleapp/wsdl/inquireSingle.wsdl
/*
```

The main input parameters are as follows:

- PDSLIB

  Specifies the name of the partitioned data set that contains the high level language data structures to be processed.

- REQMEM

  Specifies the name of the partitioned data set member which contains the high level language structure for the Web service request. This is the data structure that the target application will take as input.

- RESPMEM

  Specifies the name of the partitioned data set member which contains the high level language structure for the Web service response. This is the data structure that the program will return as output.

- LANG

  Specifies the language of the language structure to be parsed.

- PGMNAME

  Specifies the name of the target CICS application program that is being exposed as a Web service.

- URI

  Specifies the relative URI that a client will to use to access the Web service. CICS uses the value specified when it generates a URIMAP resource from the Web service binding file created by DFHLS2WS: the parameter specifies the path component of the URI to which the URIMAP definition applies.

- PGMINT

  Specifies how CICS passes data to the target application program (using a COMMAREA or a channel).

- WSBIND

  Specifies the location on HFS of the Web service binding file.

- WSDL

  Specifies the name on HFS of the Web service description file.

DFHWS2LS

This program generates a language structure and Web services binding file from a Web services description. Example 8-2 shows sample JCL for running DFHWS2LS.

Example 8-2 DFHWS2LS JCL sample

```
//WS2LS JOB 'accounting information',name,MSGCLASS=A
//JAVAPROG EXEC DFHWS2LS
//INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.CICS.SDFHSAMP
REQMEM=CPYBK1
RESPMEM=CPYBK2
LANG=COBOL
PGMNAME=DFH0XCMN
URI=exampleApp/inquireSingle
PGMINT=COMMAREA
```

```
WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
WSDL=/u/exampleapp/wsdl/inquireSingle.wsdl
BINDING=exampleAppInquireSingleHTTPSoapBinding
/*
```

The main differences to the parameters for DFHLS2WS are:

- REQMEM

  Specifies the name of the partitioned data set member into which a language structure will be generated that corresponds to the Web service request

  - For a service provider, the Web service request is the input to the application program.
  - For a service requester, the Web service request is the output from the application program.

- RESPMEM

  Specifies the name of the partitioned data set member into which a language structure will be generated that corresponds to the Web service response.

  - For a service provider, the Web service response is the output from the application program.
  - For a service requester, the Web service response is the input to the application program.

- LANG

  Specifies the language of the language structure to be generated.

- PGMNAME

  Used to switch between provider and requester Web services. If no PGMNAME is specified the Web service will be generated as a requester Web service. If specified, contains the name of the target CICS application program that is being exposed as a provider Web service.

- BINDING

  A WSDL document may contain more than one Web service description. The BINDING parameter is required to identify a unique Web service and collection of operations within the WSDL file.

Supported datatypes

As mentioned above there are some datatypes that at the time of writing cannot currently be supported by the CICS Web Services Assistant. Some of the most significant of these are listed in Table 8-1.

Note: The CICS Web Services Assistant is constantly being upgraded. For the current list of supported data types, refer to the CICS InfoCenter.

Table 8-1 CICS Web Services Assistant unsupported data types

Language	Data type
COBOL	OCCURS DEPENDING ON REDEFINES COMPUTATIONAL COMPUTATIONAL-1 COMPUTATIONAL-2 BINARY
C / C++	decimal float double long double
PL I	decimal float double long double PICTURE strings

8.2.2 WebSphere Developer for System z

WebSphere Developer for System z V6 is based on the IBM Rational Software Development Platform and facilitates the development of both Java and z/OS-based applications. It includes capabilities that make traditional z/OS mainframe development, Web development, and integrated composite development faster and more efficient.

In particular, WebSphere Developer contains tools that support the development of Web services and the XML enablement of existing CICS COBOL applications.

The XML Services for the Enterprise (XSE) capability of WebSphere Developer provides tools that let you adapt COBOL-based applications so that they can consume and produce XML messages. The XSE tool will perform much the same functionality as the CICS Web Services Assistant in that it will take a language structures in the form of COBOL copybooks or Web service descriptions as WSDL documents and produce wsbind files that can be installed into a CICS region. As the tool is driven through an interactive wizard a greater degree of flexibility is available to the developer though the option of optimizing the generation is obviously limited. At the time of writing the functionality of the XSE tool is limited to the COBOL language only.

Invoking the XSE tooling can be as simple as importing a COBOL copybook into your workspace, right-clicking it and selecting as shown in Figure 8-1.

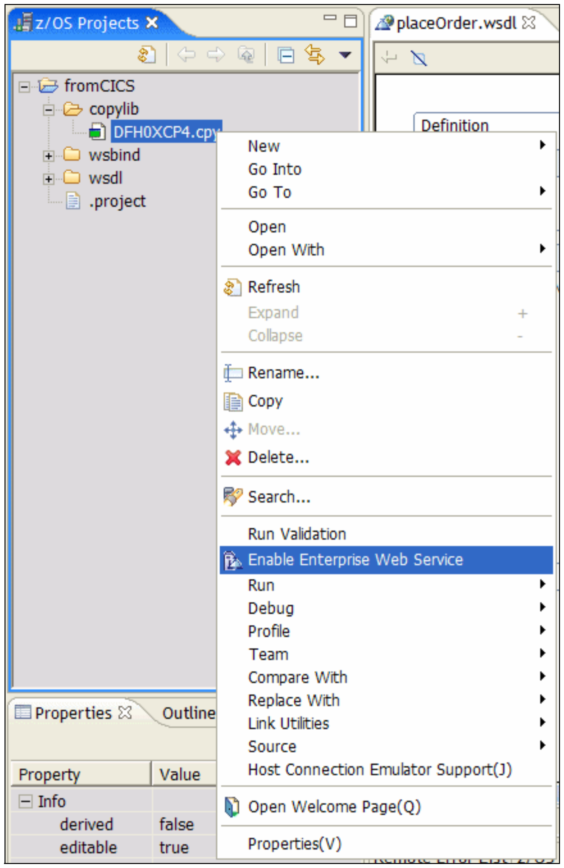
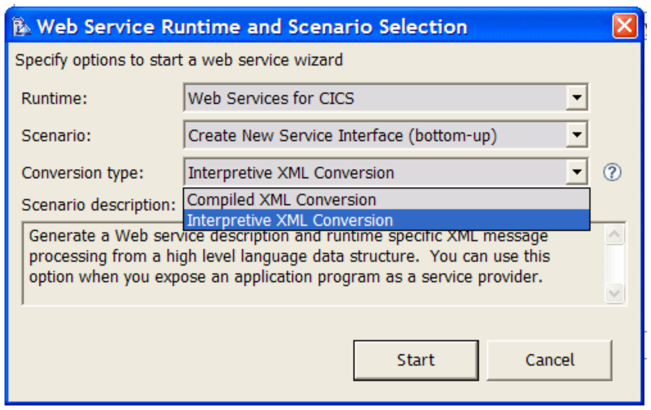


Figure 8-1 Invoking the XSE tool

When using the XSE tool there a two main options available to you as. These are selected on the first panel of the XSE tool as shown in. They are:

- Compiled style
- Interpretive style

Figure 8-2 shows the first panel of the XSE tool wizard. It is here that you select the style of the WSBind file produced.



Web Service Runtime and Scenario Selection

Specify options to start a web service wizard

Runtime: Web Services for CICS

Scenario: Create New Service Interface (bottom-up)

Conversion type: Interpretive XML Conversion

Compiled XML Conversion

Interpretive XML Conversion

Scenario description:

Generate a Web service description and runtime specific XML message processing from a high level language data structure. You can use this option when you expose an application program as a service provider.

Figure 8-2 XSE tool generation styles

Compiled style

The compiled style of program generation is the traditional tooled style and is an update to the tooling that was available in WebSphere Studio Enterprise Edition. In this style XSE supports the creation of *driver programs* that work with existing CICS applications. A vendor style wsbind file is created that delegates the XML marshalling and demarshalling to the externally generated conversion programs.

The Web Services Enablement wizard is the XSE tool that supports the bottom-up approach for creating Web services based on existing CICS COBOL programs. It takes as input the COMMAREA copybook. The XML structure and data types are then derived from the COBOL data declarations. Based on these, the Web Services Enablement wizard generates the set of artifacts shown in Figure 8-3 on page 255.

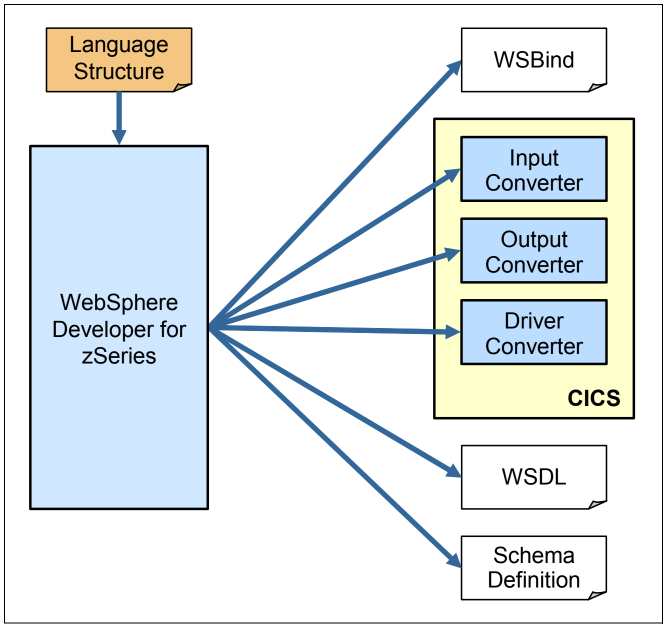


Figure 8-3 WebSphere Developer for System z

The artifacts generated by the Web Services Enablement wizard are:

- WSBind

  Web service binding file used by CICS to define the WEBSERVICE resource.
- WSDL

  Web service description file.
- Input converter

  A COBOL program that takes an incoming XML document and maps it into the corresponding COBOL data structure that the existing CICS application expects. Optionally this module can be inlined into the driver module.
- Output converter

  COBOL program that takes the COBOL data results returned from the CICS application and maps them to an XML document. Optionally this module can be inlined into the driver module.
- Converter driver

  COBOL program that shows how the input and output converters can be used to interact with the existing CICS application. Optionally this module can also perform the conversion without the need for separate driver programs.

- Input document XML schema definition (XSD)

  XML schema that describes the incoming XML document.
- Output document XML schema definition (XSD)

  XML schema that describes the outgoing XML document

Interpretive style

The XSE tool within WebSphere Developer for System z v6.0.1 now also includes the capability to create CICS Web Service Assistant metadata driven or interpretive style wsbind files. When selecting this option for generation you are actually invoking the same generation code as used by the CICS Web Services Assistant. In this case the only artefacts generated are:

- WSBind

  Web service binding file used by CICS to define the WEBSERVICE resource
- WSDL

  Web service description file

8.2.3 Development tool comparison

This section compares the use of the CICS Web Service Assistant with WebSphere Developer for System z.

Development style

The most obvious difference between the two development styles is the interface exposed to the end user. The CICS Web Services Assistant has a traditional JCL batch tool bases interface. This interface should be well known (and loved) by traditional CICS application developers and systems programmers. It also allows for the option of automating the creation of Web services artefacts.

The XSE tool in WebSphere Developer for System z provides a wizard based interface to the user. The tooling is aimed primarily at application developers and provides interactive panels through the GUI development workbench. The advantage of this interface is that it opens up CICS tooling to a new development community who are comfortable with the modern style of a development workbench. Also being interactive allows for more complex data structure to be interpreted as the user can select the appropriate sections of the data structure to be processes.

Architecture and capabilities

In terms of the experience of the application programs invoking the Web services generated by either the Web Services Assistant or the XSE tooling of

WebSphere Developer for System z there is little to separate them. The main difference is number of artefacts that must be managed and maintained by the CICS systems programmer. it is important to keep the corresponding WSDL documents and WSBind files together as the data conversion in the WSBind file is represented by the schema in the WSDL document.

In the case of the Web Services Assistant there is just the WSDL document and the WSBind file that must be handled. All the information required to marshall and demarshall the XML is contained within the metadata of the WSBind file. This is also true when selecting the *Interpretive XML Conversion* style in the XSE tooling.

When using the *Compiled XML Conversion* style in the XSE tool along with the WSDL document and WSBind file one or more COBOL source programs will be produced to perform the mashalling and demarshalling of the XML. As well as keeping the WSDL document and WSBind files together, these source programs must also be kept with their corresponding WSBind files and deployed into CICS with program names matching the file names as generated. This allows the CICS Web service runtime to know what program to call out to perform the conversion based on the information stored in the WSBind file.

If using COBOL, conversion capabilities of the two methods are fairly similar. As noted before there are several COBOL data types that are currently not supported with the CICS Web Services Assistant or interpretive XML conversion style that are available in the Compiled XML conversion style of the XSE tool. The XSE tool currently only supports COBOL data structures.

We recommend that where possible the CICS Web Services or Interpretive XML conversion styles be used. This has the advantages of less artefacts to manage and service/performance improvements to the CICS runtime will be automatically exploited without the need for code to be regenerated.

8.3 CICS Transaction Gateway

The development environments for the CICS Transaction Gateway (CICS TG) offer a solution in tune with the object-oriented application development environments that are often used for distributed SOA applications. Development of such a CICS TG architecture will involve Java programming, and working with a COMMAREA call interface.

This section introduces three ways to develop applications that use the CICS Transaction Gateway:

- Develop Common Client Interface (CCI) applications with CICS ECI resource adapter by hand

- Use the Rational Application Developer tool
- Use the WebSphere Integration Developer tool

8.3.1 CCI application development with the CICS ECI resource adapter

The CCI is the Java interface for all J2EE Connector Architecture resource adapters. You can create Java classes (in tools such as Rational Application Developer, or using the Java SDK) which make use of these resources adapters.

The CICS ECI resource adapter, which is used to make ECI calls the CICS TG, can be programmed in this way by hand.

The following list shows the basic outline for using the CCI with the CICS ECI resource adapter and sample code for programming CCI with the CICS ECI resource adapter (Example 8-3):

1. Look up a ConnectionFactory for the ECI resource adapter.
2. Create a Connection object using this ConnectionFactory. A ConnectionFactory Connection is a handle to the underlying network connection to the EIS. Specific connection properties, such as a user name and password, can be passed using an ECIConnectionSpec object.
3. Create an Interaction from the Connection. Specific interaction properties can be passed using an ECIInteractionSpec object. The call to the EIS is initiated by invoking the execute() method on the Interaction, passing data as input and output records.
4. After the required interactions have been processed, the Interaction and Connection should be closed.

Example 8-3 Sample code for programming CCI with CICS ECI resource adapter

```
1
//lookup an ECI Connection Factory
javax.naming.Context ic = new InitialContext();
cf = (ConnectionFactory) ic.lookup("java:comp/env/ECI");

2
//Create a Connection object using this ConnectionFactory
eciConn = (Connection) cf.getConnection();

3
//Create an Interaction from the Connection
eciInt = (Interaction) eciConn.createInteraction();

// setup the interactionSpec.
```

```
eSpec.setFunctionName(funcName);
eSpec.setCommareaLength(commareaLength);

// create a record for use
jsr = new JavaStringRecord(encoding);
// set input data if we have any
if (commareaLength > 0){
                jsr.setText(commarea);
}

// make the call
try{
                eciInt.execute(eSpec, jsr, jsr);
}

4
//close the Interaction and Connection and set our references to null.
eciInt.close();
eciConn.close();
eciInt = null;
eciConn = null;
```

CCI accessing CICS via the ECI resource adapter can be either used in an EJB, exposed as a Web service or simply used in a servlet. We will not cover details here; for more complete examples, refer to the Redbooks *Java Connectors for CICS*, SG24-6401, and *Rational Application Developer V6 Programming Guide*, SG24-6449.

Table 8-2 and Table 8-3 on page 260 provide a brief description of the main generic CCI classes (as they apply to CICS) and the specific ECI resource adapter classes.

Table 8-2 Generic classes

ConnectionFactory	Creates a Connection object from supplied connection settings. The connection settings are defined using the WebSphere Administrative Console, which are then looked up by the application in order to create the ConnectionFactory.
Connection	A handle to a connection managed by the J2EE application server.
Interaction	A specific interaction with CICS that occurs over an established connection.
Record	A generic wrapper for the data passed within the CICS COMMAREA.

Table 8-3 Specific ECI resource adapter classes

ECIConnectionSpec	CICS-specific connection information, such as user ID and password, which can be used to override values set in the ConnectionFactory.
ECIInteractionSpec	CICS-specific interaction information, such as the mirror transaction identifier and program name.

8.3.2 Rational Application Developer J2C Java Bean wizard

Rational Application Developer contains a wizard called the *J2C Java Bean wizard*, which can be used to create Java resources that use the CICS ECI resource adapter. These Java resources can then be deployed to WebSphere Application Server.

Note: Rational Application Developer uses the acronym J2C to refer to the J2EE Connector Architecture.

Using this tool overcomes many of the problems encountered when hand-coding CCI code to use the CICS ECI resource adapter. In particular the data type conversions (for example from a COMMAREA described in COBOL, to the Java data types used by CCI).

The J2C Java Bean wizard produces two artifacts:

- A J2C Java Bean that uses the CICS ECI resource adapter to make a call to a CICS program. This Java Bean can be wrappered as an EJB session bean or as a Web service and deployed to WebSphere Application Server.
- Data handling classes, for converting data types to and from the COMMAREA structure that is passed to the CICS program. These data handling classes are used by the J2C Java Bean to send messages to and from CICS applications. Additionally, these classes can also be used independently with other applications, as they are themselves stand-alone Java Beans.

Using the J2C Java Bean wizard

The J2C Java Bean wizard in Rational Application Developer can be used to generate a Java bean that makes calls to the CICS ECI resource adapter to invoke CICS programs.

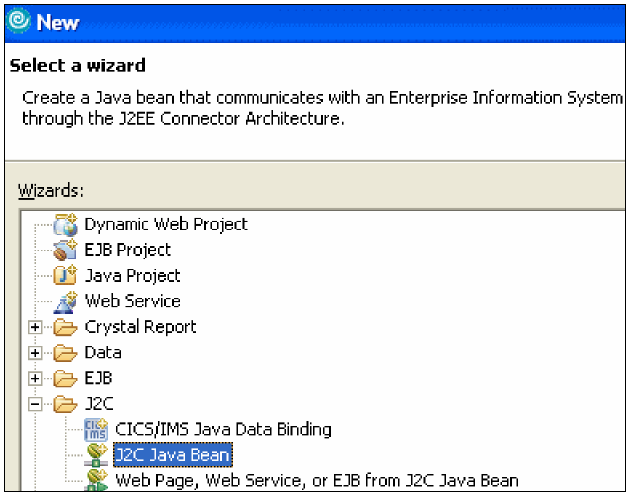
In this section we provide an overview of using the J2C Java Bean wizard. For more detailed information and a step-by-step guide refer to Chapter 6 of the redbook *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064.

The high level steps for using the J2C Java Bean wizard are:

- Starting the wizard
- CICS ECI resource adapter installation
- Java bean settings
- Message mapping settings
- Connection properties

Starting the wizard

In Rational Application Developer, to start the process click **File** → **New** → **Other**, then select the J2C Java Bean option as shown in Figure 8-4.



New

Select a wizard

Create a Java bean that communicates with an Enterprise Information System through the J2EE Connector Architecture.

Wizards:

- Dynamic Web Project
- EJB Project
- Java Project
- Web Service
- Crystal Report
- Data
- EJB
- J2C
  - CICS/IMS Java Data Binding
  - J2C Java Bean
  - Web Page, Web Service, or EJB from J2C Java Bean

Figure 8-4   Wizard to create J2C Java Bean

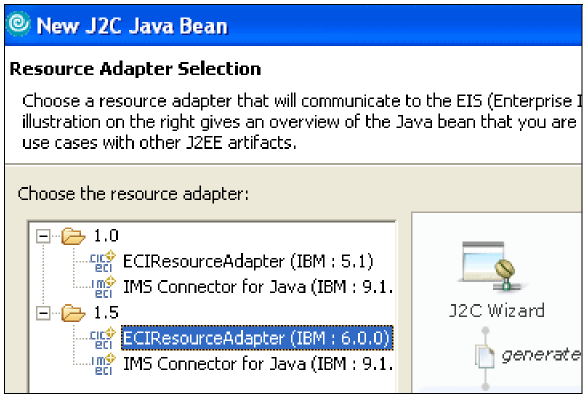
Note: To be able to perform this task, ensure that the J2C connectivity feature is installed in Rational Application Developer.

This will open the J2C Java Bean wizard.

CICS ECI resource adapter installation

Rational Application Developer requires the use of the relevant resource adapter to use this wizard. Therefore you will need to add the CICS ECI resource adapter to the workbench.

When the CICS ECI resource adapter has been made available to Rational Application Developer, select it in the wizard to indicate that this J2C Java bean should use the CICS ECI resource adapter to connect to CICS (Figure 8-5).


New J2C Java Bean

Resource Adapter Selection

Choose a resource adapter that will communicate to the EIS (Enterprise I
illustration on the right gives an overview of the Java bean that you are
use cases with other J2EE artifacts.

Choose the resource adapter:

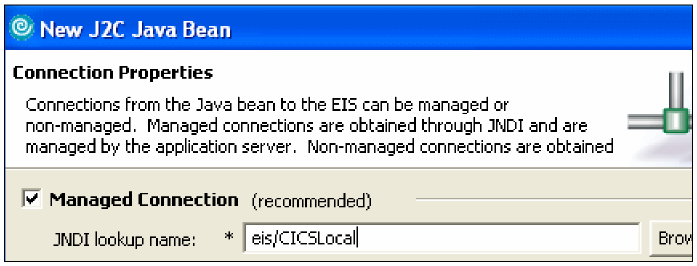
- 1.0
  - ECIResourceAdapter (IBM : 5.1)
  - IMS Connector for Java (IBM : 9.1.
- 1.5
  - ECIResourceAdapter (IBM : 6.0.0)
  - IMS Connector for Java (IBM : 9.1.

Figure 8-5   Selecting the V6 CICS Resource Adapter

Java bean settings

The next stage of the J2C Java Bean wizard requires you to specify the following information:

- ► If the connection to the CICS ECI resource adapter is to be managed by WebSphere Application Server, the JNDI name of where the resource adapter can be found (Figure 8-6).


New J2C Java Bean

Connection Properties

Connections from the Java bean to the EIS can be managed or
non-managed. Managed connections are obtained through JNDI and are
managed by the application server. Non-managed connections are obtained

☑ Managed Connection (recommended)

JNDI lookup name: * eis/CICSLocal

Figure 8-6   Setting JNDI Name

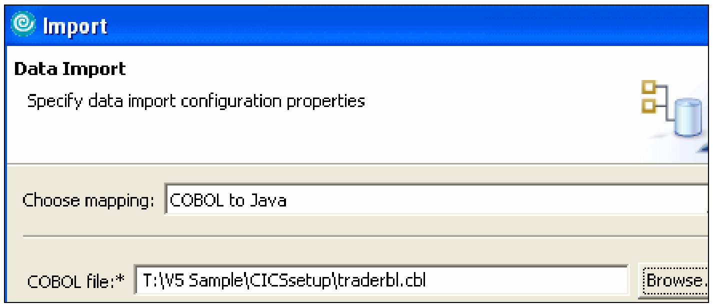
- ► The Java class name to use for the generated J2C Java bean, and the project where this Java class should be created.

Message mapping settings

In this stage we must create a new Java method that correspond to a call to a CICS program. We need to specify the following information:

- ► The name of the new Java method to create in the J2C Java bean.

- The message to send as input when invoking the CICS program. For example, you must provide the wizard with a copy of the COBOL COMMAREA structure to use (Figure 8-7).


Import

Data Import

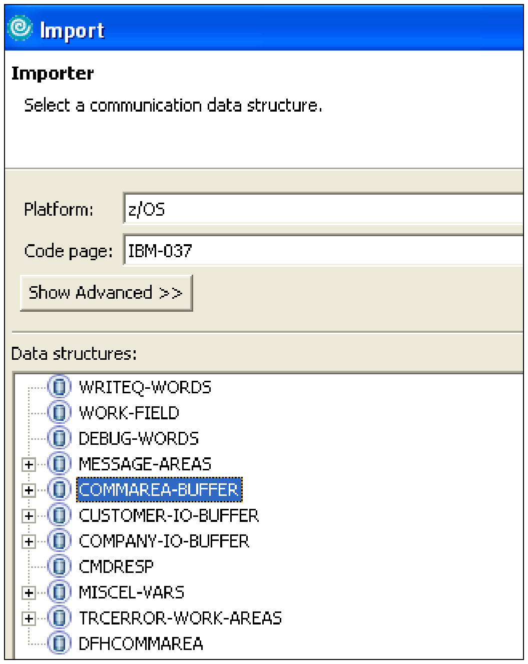
Specify data import configuration properties

Choose mapping: COBOL to Java

COBOL file:* T:\V5 Sample\CICSsetup\traderbl.cbl

Figure 8-7 Locating the COBOL program

- The platform and code page to use to communicate with CICS, and the portion of the COBOL data structure to use (Figure 8-8 on page 263).


Import

Importer

Select a communication data structure.

Platform: z/OS

Code page: IBM-037

Data structures:

```
WRITEQ-WORDS
WORK-FIELD
DEBUG-WORDS
+ MESSAGE-AREAS
+ COMMAREA-BUFFER
+ CUSTOMER-IO-BUFFER
+ COMPANY-IO-BUFFER
CMDRESP
+ MISCEL-VARS
+ TRCERROR-WORK-AREAS
DFHCOMMAREA
```

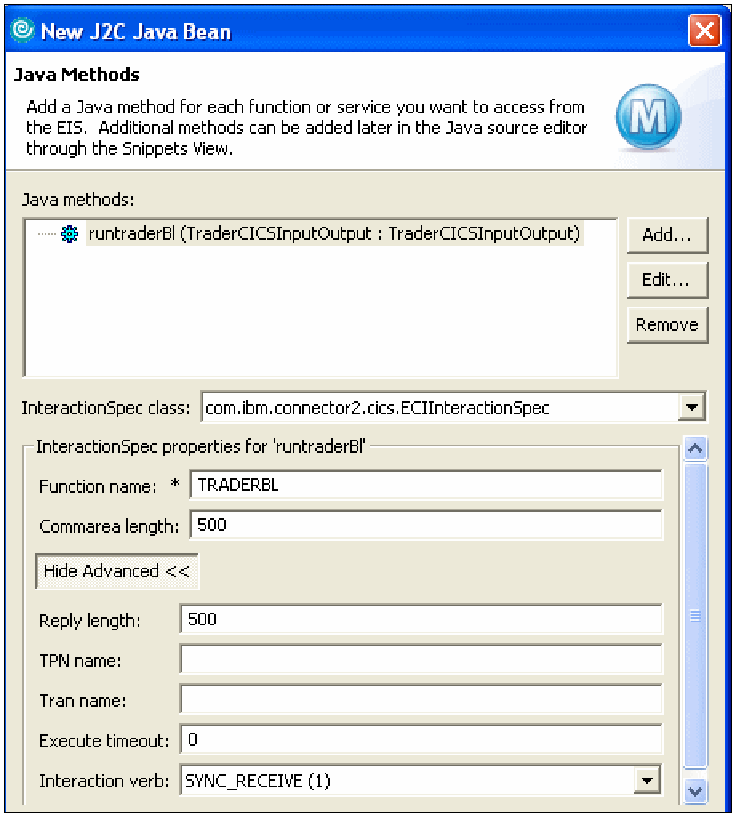
Figure 8-8 Display after setting z/OS as the platform and querying the data structure

This will generate a Java class that will map the COBOL data structure fields into Java.

Connection properties

The final stage is to set the J2C connection properties, including the following (Figure 8-9):

- The CICS program name (function) to invoke
- The size of the COMMAREA to send
- The interaction type (synchronous or asynchronous)



Java Methods

Add a Java method for each function or service you want to access from the EIS. Additional methods can be added later in the Java source editor through the Snippets View.

Java methods:

runtraderBl (TraderCICSInputOutput : TraderCICSInputOutput)

InteractionSpec class: com.ibm.connector2.cics.ECIInteractionSpec

InteractionSpec properties for 'runtraderBl'

Function name: * TRADERBL

Commarea length: 500

Reply length: 500

TPN name:

Tran name:

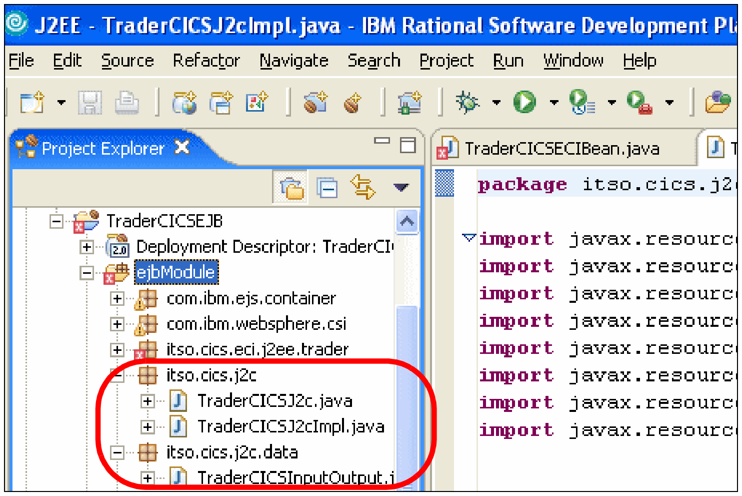
Execute timeout: 0

Interaction verb: SYNC_RECEIVE (1)

Figure 8-9 Completed Java Bean method display

Output of the J2C Java Bean wizard

When you complete the J2C Java Bean wizard, a number of classes will be created (Figure 8-10 on page 265).



```
package itso.cics.j2

import javax.resourc
import javax.resourc
import javax.resourc
import javax.resourc
import javax.resourc
import javax.resourc
import javax.resourc
import javax.resourc
```

Figure 8-10 Display showing the new Java Beans

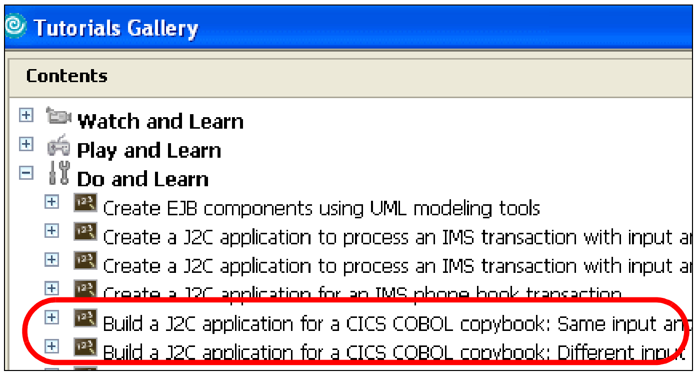
These classes include:

- Java bean classes for interacting with the CICS ECI resource adapter
- Java data handling classes to handle the conversion of Java objects to and from the COMMAREA structure that is passed to the CICS program

After development of the J2C Java bean is complete, you can use the *Web page, Web service, or EJB from J2C bean* wizard to generate a Web service or enterprise session bean from this Java bean. The Web service or enterprise session bean can then be deployed to WebSphere Application Server.

Sample tutorials

Sample tutorials on how to run the J2C wizards are available in Rational Application Developer, by selecting **Help** → **Tutorials** to reach the display shown in Figure 8-11 on page 266.



Tutorials Gallery

Contents

- Watch and Learn
- Play and Learn
- Do and Learn
  - Create EJB components using UML modeling tools
  - Create a J2C application to process an IMS transaction with input a
  - Create a J2C application to process an IMS transaction with input a
  - Create a J2C application for an IMS phone book transaction
  - Build a J2C application for a CICS COBOL copybook: Same input an
  - Build a J2C application for a CICS COBOL copybook: Different input

Figure 8-11 Sample tutorials in Rational Application Developer

8.3.3 WebSphere Integration Developer Enterprise Service Discovery

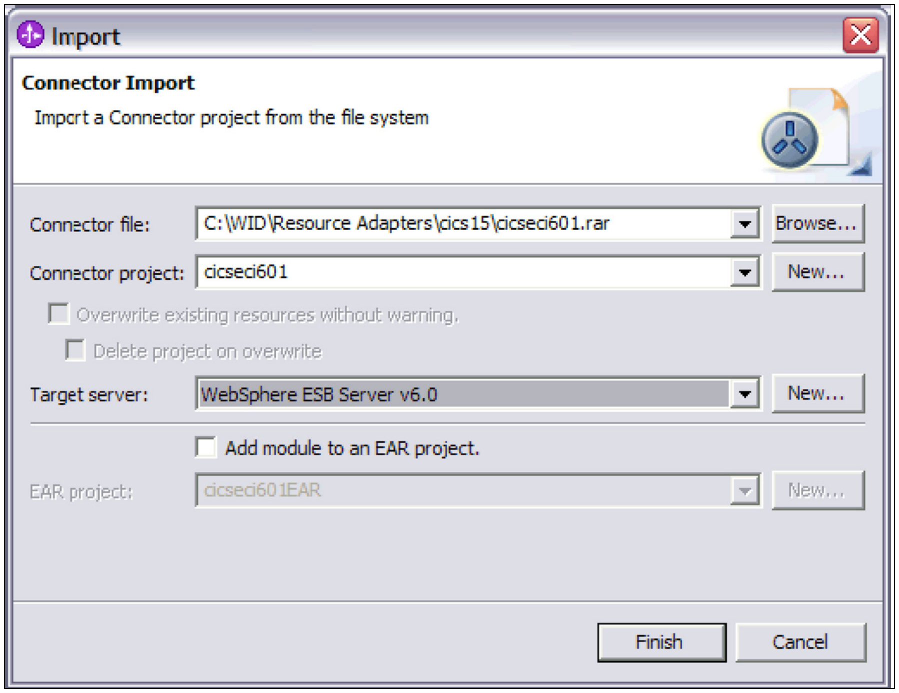
WebSphere Integration Developer is a tool for creating applications for deployment to WebSphere Process Server and WebSphere Enterprise Service Bus.

WebSphere Process Server allows you to construct a business process using WS-BPEL. WebSphere Enterprise Service Bus allows you to mediate requests between service requesters and service providers. Both WebSphere Process Server and WebSphere Enterprise Service Bus are based on Service Component Architecture (SCA).

Using the Enterprise Service Discovery wizard in WebSphere Integration Developer you can create SCA import components that make calls to CICS resources using the CICS ECI resource adapter. Therefore, CICS assets can be used in an SCA environment in the following ways:

- CICS programs can participate in business processes running in WebSphere Process Server
- Service requesters making calls to CICS can be mediated by WebSphere Enterprise Service Bus
- CICS can make calls to service providers through WebSphere Enterprise Service Bus

A step from the Enterprise Service Discovery wizard is shown in Figure 8-12 on page 267.



Connector file: C:\WID\Resource Adapters\cics15\cicseci601.rar

Connector project: cicseci601

☐ Overwrite existing resources without warning.

☐ Delete project on overwrite

Target server: WebSphere ESB Server v6.0

☐ Add module to an EAR project.

EAR project: cicseci601EAR

Figure 8-12 Connector Import screen of the Enterprise Service Discovery wizard

For an example of using the Enterprise Discovery Wizard with WebSphere Enterprise Service Bus, see Chapter 11 of the redbook *Getting Started with WebSphere Enterprise Service Bus V6*, SG24-7212.

8.4 CICS Web support

CICS Web support allows you to build your Web logic entirely within your existing CICS system. There are several options for developing applications using CICS Web support.

- Develop a brand new Web-aware application using the **`EXEC CICS WEB`** and **`DOCUMENT API`** commands.
- Write a Web-aware program to wrapper an existing COMMAREA driven business logic application. This can be written to operate using the converter interface or it can link directly to the business logic.
- Use CICS Web support to return static content such as images.
- Use the 3270 Web Bridge to provide access to 3270 applications from a Web browser.

CICS Web support can also act as an HTTP client. A new Web-aware application would be required, using the **`EXEC CICS WEB`** and **`DOCUMENT API`** to send requests to remote HTTP servers and to receive the responses.

CICS TS V3.1 is an HTTP/1.1 server. By default it expects to process HTTP/1.1 requests and produce HTTP/1.1 responses. It will still process HTTP/1.0 requests and produce HTTP/1.0 responses if the client specifically states that it is using HTTP/1.0.

Important: The API commands should be used in all new applications for CICS Web support. The alternative, of processing the HTTP request and response in the COMMAREA, will be withdrawn in a future release of CICS.

8.4.1 Web-aware/converter presentation logic

You will either have an existing COMMAREA driven business logic application or be developing a completely new Web based application. In both cases the presentation logic, handling the HTTP request and response, should be kept separate from the business logic. This allows for the greatest flexibility in re-using the business logic.

The presentation logic can either be written in a converter program or in an application program that links directly to the business logic itself. Figure 8-13 illustrates the two options.

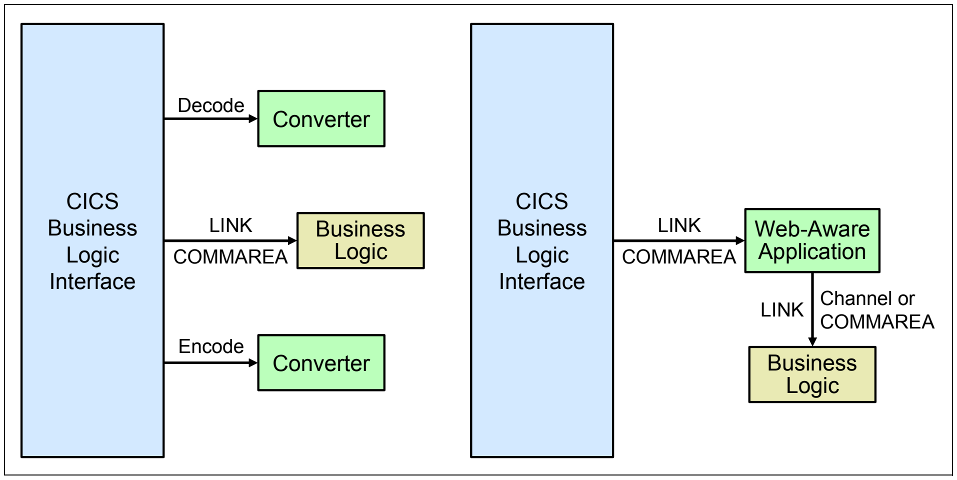


Figure 8-13 Using a converter or a Web-aware application

Table 8-4 on page 269 shows the main differences between the two options.

Table 8-4 Differences between converter program and Web-aware application program

Converter program	Web-aware application
Linked to with a defined COMMAREA format	Linked to with COMMAREA containing first 32K of the HTTP request. Contents can be completely ignored when using the API commands
Called twice. Once for *decode* and once for *encode*	Called once
Link to business logic done by the CICS business logic interface using a COMMAREA only	Link to business logic done directly by the application so can use a CHANNEL or a COMMAREA

Whether a converter program or separate application program is used, CICS supplies a set of API commands to be used by Web-aware presentation logic. These commands enable HTTP requests to be processed and HTTP responses to be constructed easily. The same commands are available whether CICS is acting as an HTTP server or as a client. The commands are as follows:

- **`EXEC CICS DOCUMENT`** allows you to build and manipulate documents, especially HTML pages, possibly based on templates defined as a DOCTEMPLATE resource.
- **`EXEC CICS WEB`** is the basic HTTP interface: you can use it to read information from the incoming HTTP header, handle the native HTTP datastream, and send an HTTP output containing a DOCUMENT.
- **`EXEC CICS EXTRACT TCPIP`** returns TCP/IP information.
- **`EXEC CICS EXTRACT CERTIFICATE`** allows you to extract information from digital certificates exchanged over an SSL direct connection.

It is possible to process the HTTP request and response in the COMMAREA. This is not recommended and new applications should use the API commands.

8.4.2 Making the application work

With the presentation logic and business logic written you need to consider how the end user with a Web browser is going to access the application. This section looks at the options for mapping the URL used on the browser to the resources in CICS representing the requested application.

The recommended way to achieve this mapping is by using the CICS URIMAP resource. The URIMAP allows a URL to be mapped to a set of CICS resources such as alias transid and converter program name. The mapping allows URLs to be used that do not contain any CICS specific information. An example URL would be:

```
http://www.example.com/processpayment
```

The URL has not exposed any CICS internal information. A URIMAP can match this URL and map it to CICS resources. For instance it could specify that the alias transid is WPAY, there is not a converter, and the target application program is WEBPAY.

An alternative to using URIMAPs is to use an analyzer program. This program examines the incoming HTTP request and then determines the CICS resources that should be used to process it. CICS supplies several sample analyzers to perform different functions. DFHWBAAX is now the default analyzer. It forces URIMAPs to be used and issues a more informative error message if a request not matching a URIMAP is received. The default from previous releases of CICS is DFHWBADX. This enables HTTP requests to flow into CICS via a URL of the format:

```
http://<hostname:port>/<converter>/<alias>/<program>
```

In this URL:

- `hostname` is the IP address or hostname of your z/OS system. If `port` is specified, it is the PORTNUMBER specified in the TCPIPSERVICE for the listener transaction.
- `converter` is the name of your converter program used; if there is no converter, you should use `cics` in the URL.
- `alias` is the name for the alias transaction under which your converter and user program will run. The default alias is CWBA. To use this, you specify `CWBA` in the URL. If you want to use a different transaction, its resource definition should specify `DFHWBA` as the program name.
- `program` is the actual CICS application program being Web-enabled and which will be called by the BLI.

To issue the same request as with the URIMAP above, the URL would be in this format:

```
http://www.example.com/cics/WPAY/WEBPAY
```

This format of the URL runs the same alias transaction and same application program, but has exposed the names of those CICS resources directly in the URL. This is not ideal for security or maintainability.

Note: An analyzer can be used in conjunction with a URIMAP. For example, the analyzer is a good place to perform auditing of Web access into CICS. In such a setup the URIMAP would match the URL and specify the CICS resources to be used and request that the analyzer be called. The analyzer can then audit the request and allow processing to continue. It can also audit failed access attempts by identifying that the request did not match a URIMAP. Another good use of the analyzer is to set field WBRA_COMMAREA to indicate that the Web-aware application is going to return an HTTP/1.0 response. This prevents CICS trying to add HTTP/1.1 specific headers to such responses.

8.4.3 Static content

Some of the content being requested by a user may be static. Such content does not require any application logic to be run. Examples of this would be GIF or JPEG images used for icons and background images on Web pages, and Cascading Style Sheets used to describe the formatting of Web pages.

Static content is able to be returned by using a URIMAP resource definition. The URIMAP specifies the URL to match, the location of the static content, and its mediatype. If the response is text based then code page conversion properties can also be set.

Note: Static content can reside in HFS files. A generic URIMAP able to match many different requests can refer to an HFS directory allowing you to easily serve images and other content residing in the same HFS directory using only a single URIMAP.

8.4.4 Chunking and pipelining

Another application design issue is the use of chunking and pipelining with HTTP/1.1 requests.

Chunking is where a potentially very large response is split into much smaller chunks which are sent as separate HTTP requests or responses. Chunking can be very useful when a response is going to take a long time to produce, such as retrieving information from a database query, but some information is available to return without waiting for the whole request to complete. Using the database query as an example the first part of the Web page can be returned as a chunk, then the first 10 records from the database can be formatted and returned as a chunk. Further records can then be formatted and returned in chunks. Finally the end of the Web page is returned in a chunk. While it takes the same time to return all the data to the client as without using chunking, the user does not suffer such a long delay before getting a response.

Pipelining is where multiple HTTP requests are sent by a client without waiting for any responses. Responses must then be returned from the server in the same order that the requests were received. If CICS receives a series of pipelined HTTP requests from a client then each is processed in turn. The next request in the pipeline is not processed until the current request has completed and its response sent back to the client.

8.4.5 Summary

Use URIMAPs to define the URL formats used to access your CICS system with CICS Web support. They allow you to mask the CICS resources being used from the end users. They also allow both dynamic and static content to easily be returned.

Write new Web-aware applications using the API commands and not the COMMAREA. The API commands remove the necessity to know about the format of HTTP requests and responses, while still allowing the flexibility to work with specific HTTP headers if required.

9

Data conversion

When using CICS applications for SOA access, data conversion is a key issue, because CICS, which runs on IBM System z processors, grew up in an EBCDIC world, whereas the Web evolved from the ASCII-based, PC, and UNIX worlds, and has now proliferated into almost every corner of the globe.

Consider these key issues when designing your CICS solution:

- Do you want to perform the data conversion within CICS?
- Are you using an Integrated Development Environment (IDE)? If so does it offer any data conversion functionality?
- Do you have simple character-based data or are you using more complicated numeric formats?
- Do you have any specific national or regional language considerations?

First, however, let us take a step back and look at the basics of data conversion.

9.1 Data conversion basics

This section introduces the following basic concepts involving data conversion:

- Character sets
- Unicode
- z/OS Unicode Services
- HTTP data streams
- Unescaping HTTP user data
- IIOP data streams
- XML
- Data conversion with Java
- CICS data conversion table DFHCNV
- Big-endian and little-endian
- Sharing CICS programs between architectures
- Customizing NLS support

Character sets

Fundamentally, computers just deal with numbers. Thus, to store letters and other character data, encoding schemes have been developed to represent each character using a given numeric value. Principally, two completely unrelated schemes are in use, American National Standard Code for Information Interchange (ASCII) and Extended Binary-Coded Decimal Interchange Code (EBCDIC). ASCII is predominant on UNIX platforms and Intel-based machines, whereas EBCDIC is used on IBM z/OS and OS/400® systems. To further complicate matters, different nationalities use different languages and national alphabets. Thus, to accommodate the different characters required for each alphabet, different versions of ASCII and EBCDIC encoding schemes (termed *character sets*) exist for each local language.

Each character set is referenced by a specific *code page* identifier unique to itself (examples being 037 for US EBCDIC and 297 for French EBCDIC). Most Latin-based character sets are coded with single-byte character sets (SBCS), where one byte represents each character, thus providing for up to 256 characters. However, to support languages with large numbers of alphabetic characters (such as Asian languages), double-byte character sets (DBCS) have been defined, where two bytes are used for each character, thereby providing for over 64,000 characters.

Unicode

Unicode is a double-byte character set developed by the Unicode consortium (`http://www.unicode.org`) that aims to provide a unique number for every character, "*no matter what the platform, no matter what the program, no matter what the language*." It has been adopted as the underlying character set for the Java language, and is fundamental to its portability. It is also required by many modern standards such as XML, LDAP, CORBA 3.0 and WML.

There are several different encodings of Unicode character data in use. UTF-8 and UTF-16 are the common ones. UTF stands for Unicode Transformation Format.

UTF-8 is an octet (8-bit) lossless encoding of Unicode characters. It encodes each Unicode character as a variable number of 1 to 4 octets, where the number of octets depends on the integer value assigned to the Unicode character. It is an efficient encoding of Unicode documents that use mostly US-ASCII characters because it represents each character in the range U+0000 through U+007F as a single octet.

UTF-16 encodes each character in the range U+0000 to U+D7FF and U+E000 to U+FFFF as a single 16 bit value that is the same as the value of the Unicode character. Unicode characters in the range U+10000 to U+10FFFF are encoded as a pair of 16 bit values where each 16 bit value is between D800 and DFFF.

Further details on these encodings can be found on the Unicode consortiums Web site at:

`http://www.unicode.org`

z/OS Unicode Services

z/OS provides data conversion for Unicode data. Typically this will be converting Unicode to and from the EBCDIC code page that the z/OS system is using. However, it can handle conversion to ASCII code pages and can even be used to perform standard ASCII to EBCDIC conversion.

CICS Transaction Server has always had its own code page conversion facility. This is provided by DFHCCNV and does not support Unicode. CICS Transaction Server 3.1 uses the z/OS Unicode Services to provide support for Unicode.

The Unicode Services consist of two major parts:

- Conversion Environment
- Conversion Services

The conversion environment is created for the entire z/OS image. The initial creation requires an IPL. Subsequent modifications to the environment do not require an IPL to take effect. The environment specifies all the supported code page conversions as well as the conversion techniques that are allowed to be used.

The Conversion Services are a set of APIs that allow applications to perform code page conversion.

Note: It is likely that a z/OS Unicode Services conversion environment has already been setup for use with DB2. If the conversion environment was generated before applying APAR OA08723 then the conversion techniques required for DB2 will not work with CICS. The conversion services calls made by CICS require the conversion technique to not be specified. To support both CICS and DB2 each code page combination needs to be specified twice in the conversion environment. Once with the technique left blank, for use by CICS, and once with the technique specified as **ER** required by DB2. A conversion environment built with OA08723 applied will work with CICS and DB2 without requiring duplicate definitions.

HTTP data streams

HTTP data streams consist of a request or response line followed by one or more HTTP headers followed by user data. The HTTP *header* information is always transmitted as ASCII data using the ISO 8859-1 (Latin-1) character set. This is the base character set for HTTP and HTML. The *user data* is transmitted in the code page of the HTTP client, which is specified using the `charset` parameter of the `Content-Type` HTTP header. The `Accept-Charset` and `Accept-Language` HTTP headers are used to indicate the character sets and languages that the client can accept responses in. An example of some HTTP headers is shown in Figure 9-1. For full details on the meaning of these values, refer to the HTTP/1.1specification, RFC2616 at:

http://www.ietf.org/rfc/

```
Connection: Keep-Alive
User-Agent: Mozilla/4.72 [en] (WinNT; U)
Host: <<HOST>>
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Content-Language: en
Content-type: application/x-www-form-urlencoded
```

Figure 9-1 HTTP headers

Unescaping HTTP user data

When a Web browser builds a HTTP datastream, it will convert some special characters in user data fields into an escape sequence of the form %*nn*, where *nn* is the ASCII value of the character. This is because those characters are used as delimiters in the HTTP datastream. For example, the character “&” is used to delimit the end of a parameter=value pair. If the user enters the string “big&blue” in an HTML form, the browser will convert the string to “big%26blue” before sending it in the HTTP datastream.

The analyzer program in CICS Web support has a parameter `wbra_unescape` to specify whether or not it should *unescape* the HTTP user data before passing it on to the Business Logic Interface. By default, the analyzer will not unescape the incoming request.

IIOP data streams

The Internet Inter-ORB Protocol (IIOP) is a TCP/IP based implementation of the General Inter-ORB Protocol (GIOP) that defines formats and protocols for distributed applications. It is part of the Common Object Request Broker Architecture (CORBA). While the IIOP header information is transmitted as ASCII data using the ISO 8859-1 character set, the application data can be any kind of object. More information about IIOP is provided in *The Common Object Request Broker: Architecture and Specification* at:

`http://www.omg.org/technology/documents/formal/index.htm`

XML

The eXtensible Markup Language (XML) is a subset of SGML. Its goal is to enable SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

XML documents are usually encoded in UTF-8 (the default) or UTF-16 and XML processors are required to read both of these encodings. Other encodings may be used by use of an encoding declaration within the entity declaration to which that encoding applies. If the encoding applies to the whole document then the encoding declaration is part of the XML declaration. A couple of examples of this are shown in Example 9-1.

Example 9-1 XML declarations containing encoding declarations

```
<?xml encoding='UTF-8'?>
<?xml encoding='EUC-JP'?>
```

More details about XML can be found in the specification which is located at:

`http://www.w3.org/TR/REC-xml/`

Data conversion with Java

The Java programming language, from its very conception, was designed to be platform neutral. Applications written in Java are compiled into byte code, which can then be transferred unmodified to any platform, and executed without change.

To implement its neutrality, Java uses neither ASCII or EBCDIC, but instead uses the Universal Character Set, also known as Unicode. Unicode is a double-byte character set (DBCS), which stores each character as a 2-byte value, and thus provides for a huge range of possible characters in order to encompass most of the written alphabets of the world. Importantly, this means that in Java the *String* object is represented by a sequence of Unicode characters, whereas primitives (including bytes and byte arrays) are represented in the default encoding of the platform in use. Communication with CICS generally requires the use of byte arrays.

Also, importantly, the first 256 characters of Unicode are the same as the 256 1-byte characters of the ASCII ISO 8859-1 character set, the only difference being that the Unicode characters are prefixed with an all-zero most significant byte, since they are double-byte characters. This makes conversion between Unicode and ASCII relatively easy, but can also lead to some pitfalls when porting code to z/OS. For further details refer to the redbook *Integrating Java with Existing Data and Applications on OS/390*, SG24-5142.

CICS data conversion table DFHCNV

DFHCNV is the CICS data conversion table, and is used by DFHCCNV, the CICS data conversion program. It is invoked from the CICS mirror program and from the CCI Connector for CICS TS to convert both incoming and outgoing data. It is also used by CICS Web support and invoked from CWXN to convert incoming data and by the alias task to convert the outgoing data. There is only one DFHCNV table per CICS region, and it must be assembled and link-edited ahead of time. In it, you need to define a DFHCNV table entry for resources for which you require data conversion (Figure 9-2 on page 279). For CICS Web-enablement, such a resource will typically be a CICS program (RTYPE=PC) that is linked to. For each resource, you specify a data length (DATALEN), a data type (DATATYP), a client code page (CLINTCP), and a server code page (SRVERCP). If you have mixed data types (such as character and binary data) in a resource, you can specify multiple `TYPE=FIELD` entries to signify that different conversion rules apply to each type.

```
        DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=TRADERBL,                  *
        CLINTCP=8859-1,SRVERCP=037
        DFHCNV TYPE=SELECT,OPTION=DEFAULT
        DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=372
        DFHCNV TYPE=FIELD,OFFSET=372,DATATYP=NUMERIC,DATALEN=4,      *
               LAST=YES
```

Figure 9-2 DFHCNV table: CICS data conversion table

Big-endian and little-endian

The representation of numeric (or integer data) is also different on different computer systems. The two different schemes are referred to as big-endian and little-endian. *Big-endian* uses the format where the most significant byte (big end) is stored first (at the lowest storage address). *Little-endian* uses the opposite format, whereby the most significant byte is stored last (at the highest storage address).

Thus a big-endian 2 byte (16-bit) unsigned integer of decimal value 255 (or hexadecimal X'FF') is stored as the value X'00FF', whereas on a little-endian architecture, it would be stored as X'FF00'. The System z and RISC based platforms (UNIX) use a big-endian architecture, whereas Intel platforms (Windows) use a little-endian architecture.

DFHCNV supports the conversion of numeric data from little-endian format to big-endian. This is achieved using DATATYP=NUMERIC as illustrated in Figure 9-3. Either 2-byte (16 bit) or 4-byte (32 bit) data types can be converted. In this example, the COMMAREA has two fields of 4-byte little-endian numeric data, followed by 120 bytes of character data.

```
       DFHCNV  TYPE=ENTRY,RTYPE=PC,                               *
               CLINTCP=8859-1,SRVERCP=037,                        *
               RNAME=USERPGM,USREXIT=NO
       DFHCNV  TYPE=SELECT,OPTION=DEFAULT
       DFHCNV  TYPE=FIELD,OFFSET=0,DATATYP=NUMERIC,               *
               DATALEN=4
       DFHCNV  TYPE=FIELD,OFFSET=4,DATATYP=NUMERIC,               *
               DATALEN=4
       DFHCNV  TYPE=FIELD,OFFSET=8,DATATYP=CHARACTER,             *
               DATALEN=120,LAST=YES
```

Figure 9-3 Sample DFHCNV conversion template for binary fields

If you want to convert 64 bit integers or other types of numeric data (such as packed decimal or floating point), you cannot use the support in DFHCNV. Instead, you have to rely on another method, either in the client using the Java data conversion methods, or using the CICS DFHUCNV user replace module (refer to *CICS Family: Communicating from CICS on System/390*, SC33-1697.

Customizing NLS support

CICS 3270 users have access to different national language message catalogs for the system messages through the use of the LANGUAGE field on the CESN sign-on screen. CICS Web support messages (HTTP responses) are in English, but a user-replaceable module (URM) exists, the *Web error program* DFHWBEP, which allows the error responses to be added or modified. It is described in the *CICS Internet Guide*, SC34-6450. Products that install on non-z/OS platforms, like the CICS TG and the CICS Service Flow Feature, offer translated menus, messages, help, and documentation in a selection of languages, which are chosen at installation time.

9.2 CICS Web services

The CICS Web services support is based on the new CICS TS V3 Channel and Container support. This support allows multiple containers to be passed between application programs in channels. These containers can be placed into, or retrieved from, a channel through simple PUT and GET commands. EXEC CICS API commands that took a COMMAREA as a parameter have been updated to allow the option of passing a channel instead. One new capability available with this function is the ability to perform data conversion on the contents of a container when it is placed into or retrieved from a channel.

9.2.1 Service provider: Inbound pipeline

The transmitted SOAP request will be processed by the PIPELINE in UTF-8 format. In most cases this is the format that the request will have been sent in. The data is stored in containers as a CHAR (character) datatype. If the transport code page is not UTF-8 then the z/OS administrator must set up conversion routines between the transport code page and UTF-8 to be invoked by the CICS container code. The transmitted SOAP request will be placed in the container DFHREQUEST.

In a normal scenario the contents of this container will be processed by a CICS supplied SOAP 1.1 or SOAP 1.2 pipeline message handler. This processing is done on the data as UTF-8. The SOAP body is extracted into a container DFHWS-BODY of type CHAR before being passed to the CICS WEBSERVICE resource or the custom conversion program depending on the PIPELINE definition. The response will also be in UTF-8 format in the container DFHRESPONSE.

The CICS WEBSERVICE resource will always get this container as EBCDIC. It is therefore required that conversion between UTF-8 (1208) and EBCIDIC be defined to the z/OS conversion services tables. The code page the WEBSERVICE resource will use to pass the language structure to the target application is by default the value set for LOCALCCSID. If it is required that the language structure be passed to the target application in a different code page the optional parameter CCSID can specified when generating the Web services artefacts when using the Web Services Assistant. Figure 9-4 shows how the data conversion occurs in an provider pipeline.

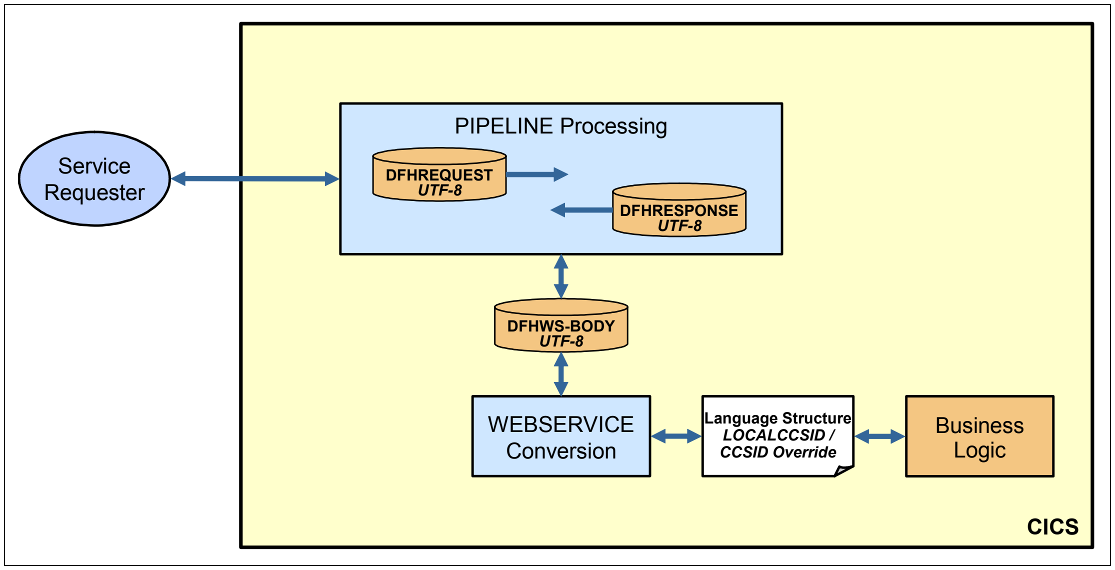


Figure 9-4 CICS Web services and data conversion

For more information about the z/OS conversion services see *z/OS(R) Support for Unicode: Using Conversion Services.*

9.2.2 Service requester: Outbound pipeline

As with the service provider case, the final SOAP message that is transmitted over the transport will be in UTF-8 format. This message is normally built by a CICS supplied SOAP 1.1 or 1.2 pipeline message handler and placed into the container DFHREQUEST.

When using the EXEC CICS INVOKE WEBSERVICE API command to send a Web service request, the container DFHWS-DATA must be placed on a channel and passed as a parameter to the API. The contents of this container will be converted to EBCDIC for processing by the CICS WEBSERVICE conversion routines before being passed to the PIPELINE. This code page will be the value set in LOCALCCSID unless an override has been specifically set when creating the Web service artefacts using the CCSID parameter of the Web Services Assistant. If the application program invoking the Web service has placed the contents of the container as a different code page then a conversion routine between the code pages must be specified in the z/OS conversion services tables.

9.3 CICS Transaction Gateway

Because WebSphere Application Server runs on ASCII based platforms and CICS runs on an EBCDIC based platform, the need arises for code page conversion to be performed on data passed between J2EE applications and CICS programs. Here we explain how the data that is passed to a CICS program on an ECI call can be converted correctly between ASCII and EBCDIC with the following data conversion considerations:

- DFHCNV
- Code page aware Java programs with DFHCNV
- Code page aware Java programs without DFHCNV
- Using Rational Application Developer

9.3.1 DFHCNV and the mirror program

ECI applications use the facilities of the CICS mirror program to link to the specified user program, passing the COMMAREA for input and output. The CICS mirror program (DFHMIRS) invokes the services of the data conversion program (DFHCCNV) to perform the necessary code page conversion of the COMMAREA (Figure 9-5 on page 283).

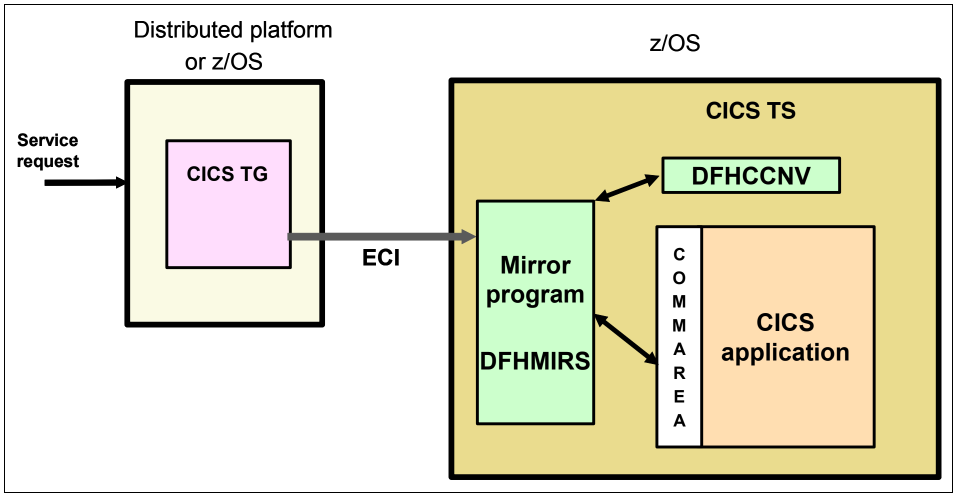


Figure 9-5 ECI data conversion

If DFHCCNV finds a conversion template in the DFHCNV table whose resource name (RNAME) matches the target program name, it performs code page conversion for the COMMAREA associated with the ECI request.

Example 9-2 DFHCNV entry

```
DFHCNV  TYPE=ENTRY,RTYPE=PC,RNAME=EC01,USREXIT=NO,
        SRVERCP=037,CLINTCP=850
DFHCNV  TYPE=SELECT,OPTION=DEFAULT
DFHCNV  TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=32767,
        LAST=YES
```

Example 9-2 shows an example DFHCNV entry for the CICS application EC01. SRVERCP represents the EBCDIC code page in which the data is stored within CICS. CLINTCP is the default code page for client requests. DATALEN is the maximum length of the COMMAREA you require to be translated.

9.3.2 Code page aware Java programs

All Java strings are stored in Unicode, a double byte character set which is similar to ASCII. The trailing byte maps to the ASCII code point for the common ASCII characters, for example, the character “A” represented by the ASCII code point X ‘41’ is represented in Unicode by X ‘0041’. The COMMAREA flowed to CICS in an ECI request must be a byte array (composed of single byte characters).

Typically, within Java, the `getBytes()` method on the String object is used to convert a String to a byte array, and similarly, the default String constructor is used to convert a byte array into a String. Unless an encoding is specified on these calls, the conversion from Unicode to the byte array will be performed using the default platform encoding. On Intel or UNIX platforms, this encoding will usually default to U.S. ASCII (ISO 8859-1); however, within the z/OS JVM, it could also be 1047 (an extended 037 EBCDIC code page).

Note: Since the arrival of WebSphere Application Server V5 on z/OS, the default JVM encoding has been U.S. ASCII (ISO 8859-1). This allows EJB components that rely on the JVM behavior instead of explicit specification of encoding to execute unchanged when ported to WebSphere Application Server for z/OS.

Because Unicode and ASCII share the first 256 code points, String to ASCII byte array conversion performs well, as it just involves removal of the high-order byte. Example 9-3 shows an example of this technique using the ECI Sample JavaStringRecord class provided by the CICS TG to perform the data conversion from Unicode to an ASCII byte array.

Example 9-3 Code page-aware Java application, ASCII COMMAREA

```
Context ic = new InitialContext();
cxfn = (ConnectionFactory) ic.lookup("java:comp/env/eis/ECICICS");
Connection cxn= cxnf.getConnection();
Interaction ixn= cxn.createInteraction();
ECIInteractionSpec ixnSpec= new
ECIInteractionSpec(SYNC_SEND_RECEIVE,"ECIPROG");
JavaStringRecord jsr = new JavaStringRecord("8859_1");
jsr.setText("DATA1");
ixn.execute(ixnSpec, jsr, jsr);
ixn.close();
cxn.close();
```

This technique means that the COMMAREA character data always flows to CICS in ASCII, and so a DFHCNV table entry is required in CICS to convert the COMMAREA from ASCII to EBCDIC.

Numeric data

If you want to flow numeric data to CICS from a Java application then it will be necessary to convert all integer values into a byte array before they can be passed to CICS. The description of how to do this is beyond the scope of this book, but further details are provided in Appendix B of *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401.

9.3.3 Code page aware Java programs without DFHCNV

An alternative to passing an ASCII COMMAREA to CICS and having the ASCII data converted to EBCDIC is to create an EBCDIC byte array, using the code page IBM037 as the encoding. This removes the need for the DFHCNV usage in CICS but can increase the cost within the Java code, because conversion from Unicode to EBCDIC in Java requires a table lookup rather than simply removing the high-order byte.

Example 9-4 shows a Java code example.

Example 9-4 Code page-aware Java application — EBCDIC COMMAREA

```
Context ic = new InitialContext();
cxfn = (ConnectionFactory) ic.lookup("java:comp/env/eis/ECICICS");
Connection cxn= cxnf.getConnection();
Interaction ixn= cxn.createInteraction();
ECIInteractionSpec ixnSpec= new ECIInteractionSpec();
ixnSpec.setInteractionVerb(ixnSpec.SYNC_SEND_RECEIVE);
ixnSpec.setFunctionName("ECIPROG");
JavaStringRecord jsr = new JavaStringRecord("IBM037");
jsr.setText("DATA1");
ixn.execute(ixnSpec, jsr, jsr);
ixn.close();
cxn.close();
```

9.3.4 Using Rational Application Developer

Instead of hand-coding JCA-based applications the facilities of Rational Application Developer can be used to build dynamically generated code. A key advantage of the use of the tooling is that it allows use of the COBOL importer wizard. This provides a find-grained set of controls for automatic marshalling of the CICS-COBOL data types from Java to native z/OS formats and vice-versa. The use of the Rational Application Developer tooling for developing JCA applications is discussed further in, 8.3.2, “Rational Application Developer J2C Java Bean wizard” on page 260.

Using the COBOL importer means the programmer can rely on the data marshalling capabilities of the tooling, and no longer need be concerned with creation of DFHCNV macro entries or the manual marshalling of data within the Java environment. Instead, all the fields can now be correctly accessed using supplied getter and setters dynamically generated by the tooling.

The Rational Application Developer COBOL importer provides the following functions:

- Character data conversion between ASCII and EBCDIC code pages

- Numeric conversion for big and little-Endian binary data types, including 16, 32 or 64-bit integers
- Numeric conversion for internal decimal (packed-decimal) data types
- Handling of external decimal signs in character data
- Handling of quote, trunc and symbol options

Note: If CICS is already configured to perform data conversion using the DFHCNV conversion table entries, then you should ensure that double conversion of the data does not occur, which will result in corruption of data.

9.4 CICS Web support

When CICS exchanges HTTP messages with a Web client or server, character data in the messages normally needs to undergo code page conversion when entering and leaving CICS. Non-character data, such as images or application specific data formats, does not require conversion.

In releases of CICS prior to CICS TS V3.1, code page conversion for CICS Web support was handled using a code page conversion table (DFHCNV). In CICS TS V3.1, the code page conversion table is no longer required for CICS Web support, except in limited circumstances for migration purposes. CICS Web support handles code page conversion using z/OS conversion services.

Note: The DFHCNV table is now only required when:

- The incoming HTTP request is processed by the IBM HTTP Server and the CICS WebServer plugin.
- The analyzer program returns a value in field wbra_dfhcnv_key that specifies a conversion table entry to be used to convert the user data associated with this HTTP request.

9.4.1 HTTP header conversion

HTTP headers are always transmitted in code page ISO-8859-1 (819). CICS converts them to the local EBCDIC code page specified by the **LOCALCCSID** system initialization parameter. If the **LOCALCCSID** is not compatible with ISO-8859-1, for example it is a DBCS code page, then code page 037 is used instead. Appropriate definitions need to be made to allow the z/OS conversion environment to convert between 819 and the local EBCDIC code page or 037. Example 9-5 on page 287 shows the statements that need to be added to the z/OS conversion environment to allow the HTTP header conversion to work for code page 037 and code page 500.

Example 9-5 z/OS conversion statements

```
CONVERSION 037,819;
CONVERSION 819,037;

CONVERSION 500,819;
CONVERSION 819,500;
```

When sending the HTTP response, the HTTP headers get converted from EBCDIC code page 037 to ASCII code page ISO-8859-1 (819).

9.4.2 HTTP user data conversion

CICS assumes a default ASCII code page of ISO-8859-1 and a default EBCDIC code page of the **LOCALCCSID** or 037, for HTTP user data. When performing conversion CICS will attempt to determine the character set of the incoming request based on the *content-type* HTTP header and only use the default, ISO-8859-1, if the character set cannot be determined or is not supported.

Data in the COMMAREA is converted using the code pages specified by the analyzer program. A URIMAP can be used in place of an analyzer program. If this is the case then the default code pages are used for conversion.

Data accessed by Web-aware applications using the **`EXEC CICS WEB RECEIVE`** command can choose:

- To not perform conversion
- To allow CICS to determine the ASCII and EBCDIC code pages to be used
- To explicitly specify the code pages that should be used

When the HTTP response is sent, data in the COMMAREA gets converted using the same code pages that were used to convert the inbound user data. Data sent using the **`EXEC CICS WEB SEND`** command gets converted based on the options specified on the command by the Web-aware application.

Note: Data in a CICS document cannot be converted to unicode (UTF-8 or UTF-16) when being sent by the **`EXEC CICS WEB SEND`** command. To return a response in UTF-8 or UTF-16 then the data must be in a buffer and the **`FROM`** option used on the **`EXEC CICS WEB SEND`** command.

Static responses can also be returned by CICS. Conversion of such responses is based on the **MEDIATYPE**, **CHARACTERSET** and **HOSTCODEPAGE** attributes of the associated **URIMAP** resource. Only responses with a text based **MEDIATYPE** are eligible to be code page converted.

Part 3

Customer scenarios

In Part 3, we provide details of different customer scenarios, in which existing corporate information systems were successfully Web-enabled using the technologies described in this book.

10

Customer scenario: CICS Web services

In this chapter we discuss how a large banking organization has used CICS TS V3.1 Web services support to integrate an existing CICS application with newly available data in a remote location.

10.1 Infrastructure

The bank has over 1700 CICS regions spread across 48 LPARS running 24 by 7. They process over 5 billion transactions every month. The CICS systems are a mix of CICS TS 2.2 and TS 3.1. About half of them have been upgraded to TS 3.1.

Through mergers and acquisitions the bank has many different data sources that are holding information on its customers. It is very expensive and time consuming to move the information on these systems to a central repository and manipulating the data into a standard format. This is a long term goal though. The alternative, shorter term solution, has been to expose access to the information using Web services and an Enterprise Service Bus. In this case webMethods is being used as the service endpoint to facilitate the integration of the data sources and the rest of the organization.

The first application that has been extended to use Web services in CICS TS 3.1 is a PIN mailing application. This is used to mail out new PINs for customers who have lost their Internet banking or ATM card PIN. Figure 10-1 shows the different systems and protocols used to process a single request for a new PIN.

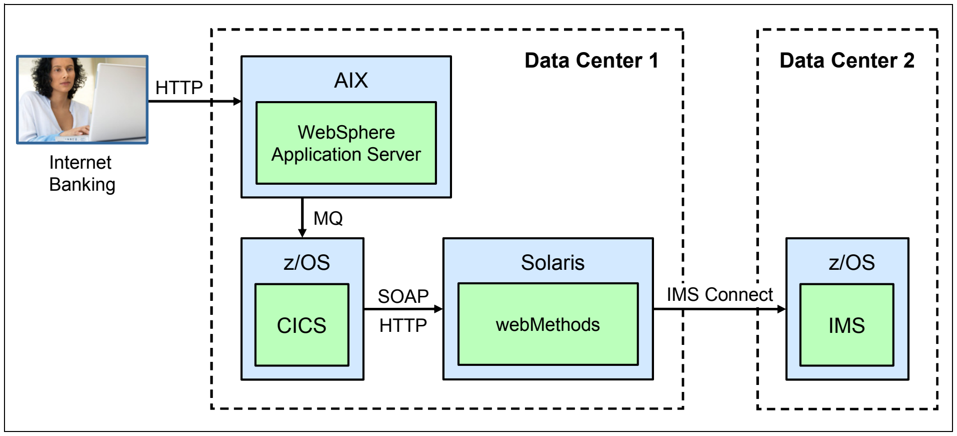


Figure 10-1 Application infrastructure

The customer uses Internet banking from a Web browser and requests a new PIN. This uses SSL and comes into WebSphere Application Server running on AIX. WebSphere MQ is then used as the transport to get the PIN request to the mailing application in CICS TS V3.1. The CICS application acts as a service requester and uses **DFHPIRT** to send a SOAP request over HTTP to a service provider running in webMethods on Solaris. The service provider in webMethods then sends a SOAP request which IMS Connect processes, enabling the required name and address details to be extracted from the geographically remote IMS database. The information from IMS is returned in a SOAP response to the CICS application which is then able to mail the new PIN to the customer.

10.2 Implementation

When using Web services in CICS there are several options for implementing the SOAP processing.

- CICS Web Services Assistant
- Converter program generated by WebSphere Developer for System z
- Perform your own SOAP/XML parsing

The original intention (and what was used for the early prototypes) was to use the CICS Web Services Assistant to process the WSDL describing the Web service hosted on webMethods, and produce a container structure for use by the CICS application on the **`EXEC CICS INVOKE WEBSERVICE`** command. By using the assistant, CICS will perform the SOAP processing and XML parsing at runtime. The application would only have had to populate the generated container structure, invoke the Web service and process the container that was returned.

Using the CICS Web Services Assistant requires files to be available in the HFS. Due to system management and process issues surrounding use of the HFS that are mentioned in 10.3, "Issues raised" on page 294, the assistant could not be used. Another reason for not using the assistant was the requirement for multiple namespace support within a WSDL document. That support was provided by APAR PK15904 but was not going to be available in the banks production systems within the timeframe of this project.

The decision was made to generate a template for the SOAP request and install that in CICS as a DOCTEMPLATE resource. The template contained all the XML required in the SOAP request and variables in place of user data. The **`EXEC CICS DOCUMENT API`** commands are then used to build a document containing the SOAP request. The application provides values for all of the variables within the document. Once the document is built it is sent to the remote Web service using an **`EXEC CICS LINK PROGRAM('DFHPIRT')`**. DFHPIRT had to be used instead of **`EXEC CICS INVOKE WEBSERVICE`** because it tells CICS not to perform any SOAP processing due to the request already being a complete SOAP request. **`EXEC CICS INVOKE WEBSERVICE`** also requires the CICS Web Services Assistant to have been used to generate the Web service. This was another reason that the command could not be used by this application.

When the SOAP response is received then it is returned directly to the CICS application which then has to parse the XML and extract the relevant information.

In this case that includes the name and address of the customer requiring the new PIN.

The rest of the infrastructure was already in place. All that was required to change was a minor alteration to the data returned by the Web service hosted in webMethods. The CICS application required some additional data that was not currently returned to existing users of that service.

10.3 Issues raised

The initial Web services project in CICS TS 3.1 has been a success and more are planned for the future. There were some important issues raised during this project which are summarized here. Those relating to use of the HFS delayed the uptake of the CICS Web Services Assistant for creation of Web services.

- Traditional CICS mainframe developers have little or no UNIX skills. The CICS Web services functionality makes significant use of the HFS. There is a significant learning curve to get existing developers able to create and test new Web services in CICS.
- There were many issues relating to the system management of HFS files, directories, and the underlying datasets. The bank did not have in place sufficient processes to handle backup, recovery, change management or promotion of files in the HFS. Also each LPAR had its own HFS structure instead of sharing a single HFS within the SYSPLEX. This meant that the files required by the CICS regions had to be duplicated in all the individual HFS structures.
- The process issues with the HFS resulted in a DOCTEMPLATE containing the format of the SOAP request being used by the CICS application. The CICS application also performed its own XML parsing of the SOAP response. Late in the project the WSDL for the service hosted in webMethods was changed. This required a change to the DOCTEMPLATE as well as application changes to populate it and to parse out new response elements. This led to a 2 month delay. If the CICS Web Services Assistant (DFHWS2LS) had been used to generate the language structures necessary to call the service then the delay would have been significantly reduced as only minor application changes would have been required.
- A security issue encountered was that it was not possible to use basic authentication with the outbound HTTP call made by the CICS requester pipeline. The user ID and password could only be passed by adding them to the SOAP message as elements within the SOAP envelope header element. The service provider would then need modifying to extract and use this data. The extra effort was not justified for this purely internal Web service so no authentication of any kind was used.

10.4 Conclusions

The first use of CICS TS 3.1 and Web services was very successful and many more applications are being written to use Web services. CICS will primarily be used as a service requester interacting with existing Web services hosted in webMethods.

The HFS management issues will be resolved and the CICS Web Services Assistant will be used for all future projects. This will reduce the application complexity and lessen the effects on development and test from changes to the WSDL describing the services being used.

The Web services support provided in CICS TS 3.1 has enabled the bank to easily integrate CICS applications with disparate data sources scattered across the organization.

11

Customer scenario: CICS Web support

In this chapter, we discuss how CICS Web support was used by the New Jersey State Police, to Web-enable a new interface to the Federal Bureau of Investigation (FBI) National Crime Information Center.

11.1 Business description

The New Jersey State Police provides a computer system which services 767 local, county, state, and Federal agencies located across the state. This CICS/COBOL system contains information such as: motor vehicle data, "state only" warrants, firearms licensing, and criminal history information. It has over 10,000 users, which access the system 24 hours a day, 7 days a week.

Figure 11-1 shows the CICS Web infrastructure.

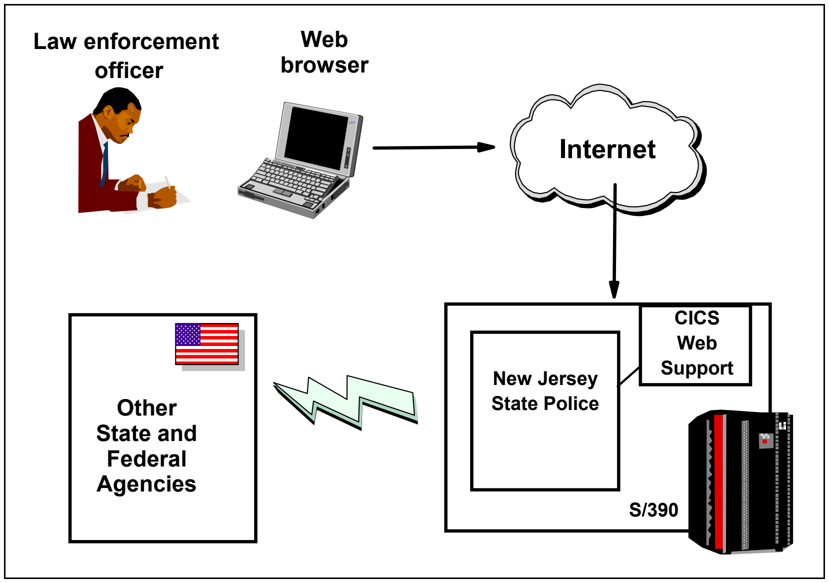


Figure 11-1 New Jersey State Police — CICS Web infrastructure

In addition, New Jersey State Police is the conduit through which New Jersey law enforcement agencies access other state and federal agencies such as the New Jersey Administrative Office of the Courts, the New Jersey Department of Corrections, the National Law Enforcement Telecommunications System, and the National Crime Information Center.

The New Jersey State Police services are about as mission critical as you can get. The difference between a 2 second response time and a 30 second response time can sometimes mean the difference between life and death. It is for that reason that the New Jersey State Police chose to implement their system on the mainframe using CICS and MQSeries®.

The National Crime Information Center is a computer system provided by the FBI, serving as the national repository for all sorts of criminal justice information. Located in Clarksburg, West Virginia, the FBI maintains 17 separate databases, containing information about wanted persons, stolen vehicles, violent gangs, and a host of other things.

The National Crime Information Center uses a tree architecture, where over 80,000 criminal justice agencies across the nation access a designated control terminal agency to derive services. It is the responsibility of the control terminal agency to provide the interface to the end user, and a message switch to National Crime Information Center. The New Jersey State Police is the control terminal agency for the State of New Jersey.

Each day the National Crime Information Center processes over 2 million transactions in sub-second fashion. In 1999 alone, information returned resulted in 113 000 individuals being arrested; 39,000 missing children and 8 500 missing adults being located; and 110 000 cars valued at over half a billion dollars being recovered.

11.2 Technology description

Using a CICS Web support solution designed by IBM Global Services, New Jersey State Police have provided the Web browser interface for their 10,000 users to access images of wanted or missing persons, stolen property, fingerprint data and more, in addition to text information such as person information and criminal history. This was accomplished by implementing a CICS TS region supporting CICS Web support, a Generic Converter solution, a Graphics Converter, application programs, and an MQSeries back-end. The design is such that business logic and presentation logic are completely separate.

The specific software technology used consisted of the following products:

- OS/390 V2.8
- CICS Transaction Server V1.3
  - CICS Web support
  - CICS to TCP/IP Sockets interface
- IBM MQSeries V5.1
- IBM HTTP Server V5.2

When a Person Inquiry is submitted (based on name and date of birth) from the Web browser, the response that is returned from the National Crime Information Center includes all relevant text and images relating to that inquiry. The text is displayed in keyword/value format to maintain compatibility with existing systems and mitigate training issues.

The Fingerprint Inquiry performs biometrics inquiries using file uploads to CICS from a Web browser. The response to a fingerprint inquiry could contain both text and images.

JavaScript™ embedded in the Web pages prevent the user from transmitting an inquiry with the mandatory fields left blank, or basic relational edit errors. After the page is submitted, more comprehensive editing occurs within CICS. If any errors are found, the same Web page is returned to the client, with an error message, focus on the error field (cursor positioning), and all check boxes and selection boxes set correctly as the end user had submitted them.

Any relevant images, such as "mugshots" or identifying tattoos are stored in a CICS VSAM data table, as a JPEG grayscale image using a compression ratio designed to reduce the image size to between 4 - 8K. An image reference is placed in the HTML response. Upon receiving the response, the browser links to a CICS based graphics converter that retrieves the image, which is inserted in the browser window.

An entire response is usually received at the browser in under 3 seconds. That is a significant achievement considering the bundling being done, the number of systems which are traversed, the telecommunications issues, and the data requests coming from West Virginia.

Business logic interface

Figure 11-2 on page 301 demonstrates the isolation of presentation and business logic which is supported by the CICS business logic interface. The converter programs perform conversion of symbol strings from the browser into a fixed length COMMAREA expected by the business logic application program and back again into HTML after processing.

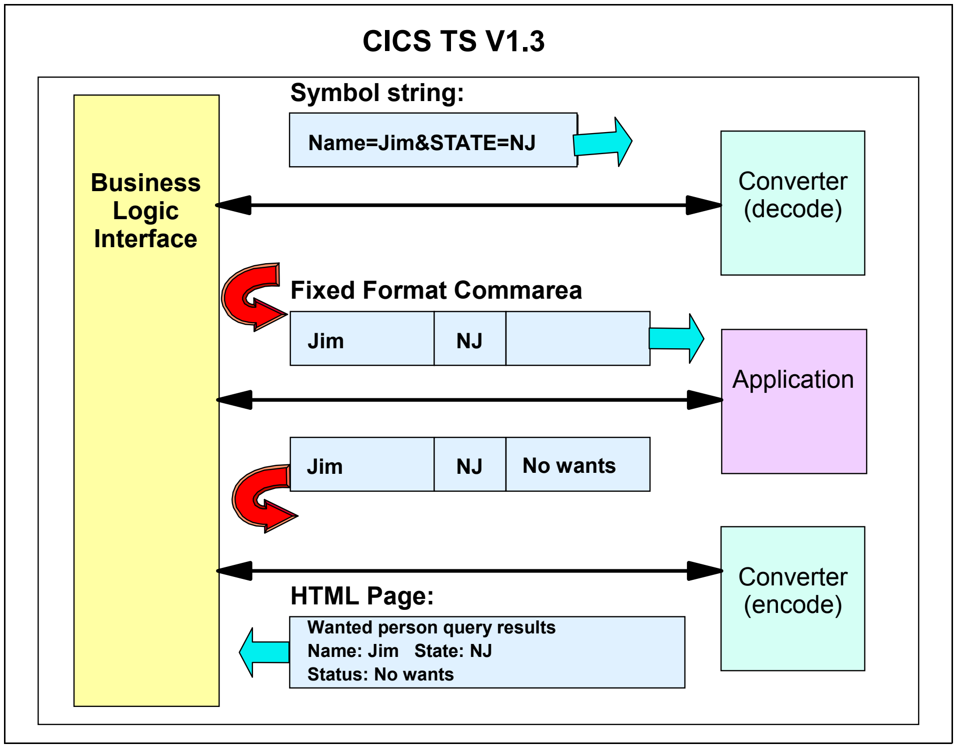


Figure 11-2 Solution of presentation and business logic

11.3 Technical implementations

The "Generic Converter" solution, available from IBM Global Services, takes this idea one step higher. This single converter can be used in place of all other application specific converters because it is driven by a fast access VSAM Data table. This meta data table contains the output HTML template names and the COMMAREA data format for the business application program. Each application program entry is defined by using the "Application Entry" program.

Generic Converter

When Web-enabling a CICS application using the Generic Converter a relationship is established between an HTML template, the Generic Converter, an Application entry, and a CICS application program.

The Generic Converter shields the application program from interfacing with State Management, Template Management, and HTTP environmental modules. It also extends CICS Web support function by supporting input HTTP with enctype=multipart to handle file uploads.

Graphics Converter

While static images may be served from any Web server, the Graphics Converter provides a means to display JPEG or GIF images from a VSAM file. This allows CICS to store and serve dynamic images received in real time from remote systems.

Statistics collection and reporting

This feature of the Generic Converter system is used for stress testing, application problem determination, and system performance monitoring. It provides the following daily and interval statistics in real time:

- Generic Converter statistics
  - Maximum, average, and last application program response time
  - Maximum, average, and last environmental response time
  - Transaction counts
  - Transaction rate
  - Maximum, average, and last HTML input and output sizes
- Graphics Converter statistics
  - Transaction counts
  - Transaction rate
  - Maximum, average, and last image output size

11.4 Issues raised

This project has been a great success for both New Jersey State Police and IBM, however the following issues encountered during the project are summarized here:

- A security design should be planned as early as possible. The solution required customization of the supplied sample CICS security analyzer, which now links to a customer security program that authorizes the target application program. Additional Web environmentals are retrieved to authorize the client IP address. An SSL implementation was undertaken in 2000 to improve authentication and provide text encryption.
- File upload required further customization to the supplied security analyzer to prevent ASCII/EBCDIC data conversion of inbound binary data. Additionally, an ActiveX® control was used to process digital images into JPEG files, and

also to automate the selection of the JPEG file for upload at the browser workstation.

- The supplied sample state management program uses a default size of 256 bytes for allocating state data area. While this size can be increased, it is a global value used for all state data requests. To provide support for update application programs, the state data program was modified to provide cleanup of temporary storage queues, created by application programs to store large amounts of state data.
- Graphics support for the retrieval of "mugshot" images required further customization of the security analyzer to pass the image key to the Graphics Converter.
- A "cookie cutter" design approach was developed as a standard methodology for Web-enabling applications. This approach proved to increase productivity for new members of the New Jersey State Police Web-enabling team.

11.5 Conclusions

The production CICS region supporting the Web components listener region was installed and a successful production system test was conducted on February 10, 2000. Beta sites began using the system in early March 2000. The New Jersey State Police was the first state agency in the US with the ability to perform fingerprint query of the National Crime Information Center 2000 database.

CICS Web support, along with the Generic Converter suite of programs, is viewed by the New Jersey State Police as an excellent solution for their business requirements. Performance is a critical factor for this application and the fact that a textual inquiry is able to return a response, including a mugshot image, in an average of 3 seconds, exceeded expectations. The implementation was achieved at a low cost, mainly due to leveraging existing CICS COBOL skills, supplemented with basic HTML training. The Generic Converter solution is easy to use and additional applications are being Web-enabled without assistance from IBM.

The New Jersey State Police is positioned to make use of WebSphere Application Server. All or part of their present system can be migrated into a Java environment. Since their present system has successfully implemented business logic that is separate from presentation logic, this business logic could be accessed from a Java application using the CICS Transaction Gateway and the External Call Interface (ECI). With minor modifications, the HTML templates could be converted to Java Server Pages.

Part 4

Appendixes

Part 4 describes two additional Web-enablement technologies in these appendixes:

- CICS EJB support
- SOAP for CICS feature

A

CICS EJB support

This appendix provides an overview of the CICS Enterprise JavaBean (EJB) support introduced in CICS V2.2, and improved in CICS TS V2.3. It contains the following sections:

- Enterprise JavaBeans
- Security
- Transactional scope
- Performance and scalability
- Application development
- Data conversion

Enterprise JavaBeans

This appendix details CICS support for the Enterprise JavaBean (EJB) specification. We begin by introducing the concepts of EJBs then give details of CICS implementation.

Introduction

Enterprise beans are reusable Java server components written to Sun Microsystem *Enterprise JavaBeans* (EJB) specification. They can be used in an application server called an Enterprise Java Server (EJS). The EJS provides interfaces and services defined by the EJB specification.

Enterprise beans execute within a container provided by the EJS. They are located by looking up their names in a name server using the Java Naming and Directory Interface™ (JNDI). The EJB container provides services such as transaction support, persistence, security and concurrency.

CICS TS V3.1 provides partial support for Version 1.1 of the EJB specification, in that it supports session beans only, and not entity beans. The EJB container within CICS provides the services required by enterprise beans running within the CICS EJB server.

Session beans

A session bean is instantiated by a client and represents a single conversation with the client. In most cases, this conversation only lasts as long as the client is active. From this point of view, the session bean is very similar to a pseudo-conversational transaction in CICS.

A session bean performs business actions such as transferring data, or performing calculations on behalf of the client. These business actions can be transactional or non-transactional. If the actions are transactional, the session bean can manage its transaction using the Object Transaction Service (OTS), or it can use the container-managed transaction services provided by the EJB container.

The CICS EJB server, like any other EJB server which complies with EJB specification, provides support for stateful as well as stateless session beans. The option as to whether an enterprise bean is to be deployed as a stateful or stateless session bean is specified in the deployment descriptor.

Stateless session beans

A stateless session bean does not store conversation state specific to a client. It provides a service without storing the state between method calls. An example would be a share quote service which returns the share price of a company and does not store details of the conversation for any subsequent inquiry. If a stateless session bean manages its own transactions, it must commit or roll back the transaction in the same method which began the transaction. In most EJB servers a pool of stateless session beans can be created to serve multiple clients. However, in CICS, each bean instance runs in a new request processor transaction, which is terminated after usage, and so the bean is removed.

Stateless session beans could implement their own state management techniques, such as returning state to the client or storing state on the EJB server (perhaps in a CICS temporary-storage queue). However, before designing such a solution, you need to consider the complexity of providing state management, as opposed to using the automatic state management process provided by stateful session beans.

Stateful session beans

When using stateful session beans, the EJB container automatically manages object state on behalf of the enterprise bean. In practice, this means all class instance variables are stored across method invocations. This conversation state is specific to the client that instantiated the session bean, and the session bean maintains this state across methods and transactions. The conversation state within a stateful session bean is stored in memory and may be moved to auxiliary storage by the CICS EJB container once the method request has terminated. This process is called *passivation* and the converse is called *activation*, whereby the state data is recreated. EJB containers typically use a least recently used algorithm to identify which beans are to be *passivated*. The CICS EJB server, however, has a more aggressive policy on passivation, and *passivates* all beans not in an OTS transaction at the end of the client method call. This ensures optimal performance in CICS, but is still compliant with the EJB specification.

Entity beans

An entity bean is an object representation of business data, such as a customer or an account. Usually an instance of an entity bean corresponds to a row in a relational database. This allows the data to be manipulated in a normal object-oriented manner by invoking methods on the entity bean. CICS TS v2 does not support entity beans. Session beans are a natural extension to the existing transactional capabilities of CICS; however, entity beans have no obvious mapping to existing CICS functionality.

How CICS provides EJB support

This section describes how CICS provides the capability to run enterprise beans in the CICS environment. The components that make up the CICS EJB Server environment in CICS TS are:

- TCP/IP listener
- Request receiver
- Request models
- Request stream
- Request processor
- EJB container
- Object store
- Java Virtual Machine
- CorbaServer
- Deployed JAR files

These components are illustrated in Figure A-1. If the components are split between a listener region and an AOR in a logical CICS EJB Server, then the components on the left must run in the listener region and the components on the right in the AOR.

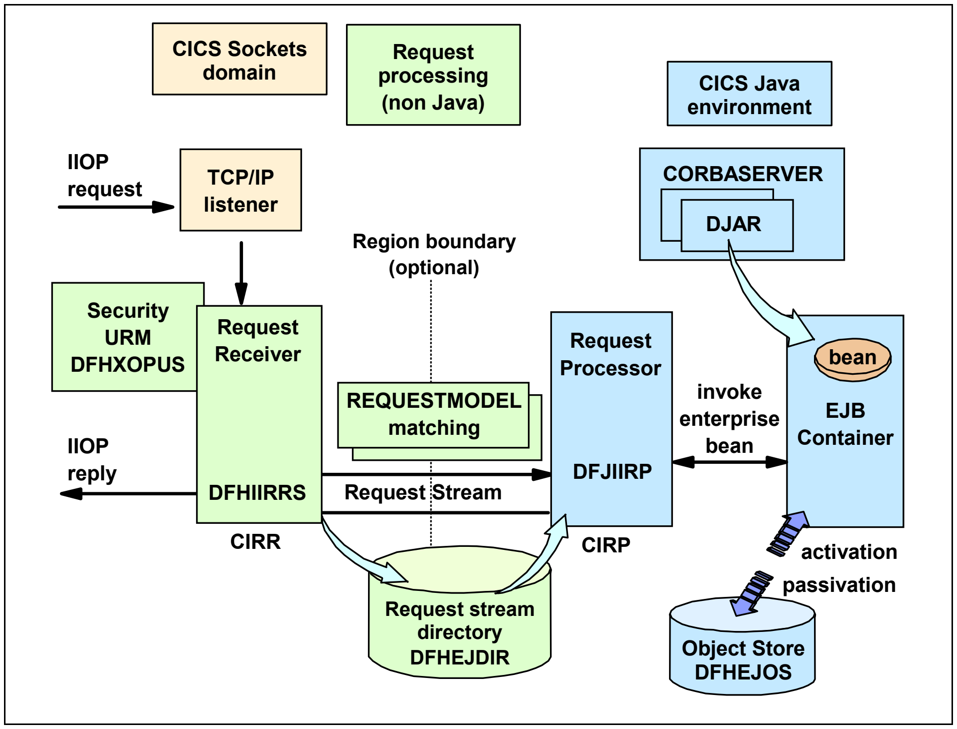


Figure A-1 Components of the CICS EJB Server

- **TCP/IP listener**

  The CICS TCP/IP listener monitors a specified port for inbound IIOP requests. When a request is received, the listener starts the CICS CIRR transaction which invokes the DFHIIRRS, the request receiver program.

- **Request receiver**

  The request receiver retrieves the incoming message and examines the contents of the IIOP message stream. If a message is processed, a response is sent to the client. The request receiver transaction stays running while the IIOP connection is open, and terminates when it has no more work to do.

- **Request models**

  To associate the incoming request stream with a CICS transaction, you need to provide and install REQUESTMODEL resource definitions for all the possible requests that CICS can process. CICS compares fields in the request against values defined in the REQUESTMODELs, to find the best match. The REQUESTMODEL definitions must be installed in both the listener regions and the AORs in a logical EJB Server.

- **Request stream**

  Requests are passed from the request receiver to the request processor using the associated request stream. Request streams are a new task-to-task communication mechanism in CICS. They are used for distributed routing of method requests for enterprise beans.

- **Request processor (DFJIIRP)**

  The request processor, DFJIIRP, manages the execution of the IIOP request and is responsible for locating the object identified by the request and calling the container to process the bean method for an enterprise bean request.

  The request processor is also responsible for processing the request for a stateless Common Object Request Broker (CORBA) object.

- **EJB container**

  The EJB container creates and manages enterprise bean instances at runtime, and isolates the enterprise beans from direct client access.

  The EJB container supports a number of implicit services, including life cycle, state management, security, transaction management, and persistence. These services are required by each enterprise bean running in the container.

  Life cycle: Individual enterprise beans do not need to manage process allocation, thread management, object activation, or object passivation explicitly. The EJB container automatically manages the object life cycle on behalf of the enterprise bean.

State management: Individual enterprise beans do not need to save or restore object state between method calls explicitly. The EJB container automatically manages object state on behalf of the enterprise bean.

Security: Individual enterprise beans do not need to authenticate users or check authorization levels explicitly. The EJB container can automatically perform all security checking on behalf of the enterprise bean.

Transaction management: Individual enterprise beans do not need to specify transaction demarcation code to participate in distributed transactions. The EJB container can automatically manage the start, enrollment, commitment, and rollback of transactions on behalf of the enterprise bean.

Persistence: Individual enterprise beans do not need to retrieve or store persistent data from a database explicitly. The EJB container can automatically manage persistent data on behalf of the enterprise bean.

Restriction: CICS TS V3 does not support entity beans and so does not support the persistence of enterprise beans.

- **Object store (DFHEJOS)**

  The object store is a non-recoverable shared file used by the CICS AORs in a logical EJB Server to store stateful session beans that have been passivated. In CICS enterprise beans are passivated at the earliest opportunity, which will occur for all enterprise beans which are not within an OTS transaction and have finished executing method requests.

- **Java Virtual Machine**

  CICS uses the IBM SDK for z/OS Java 2 Technology Edition, Version 1.4. This can be used in resettable mode, continuous mode, or as part of a shared class cache involving a master JVM and a set of worker JVMs. The use of the JVM is described in more detail in *IBM Developer Kit for z/OS Java 2 Technology Edition V1.4 Persistent Reusable Java Virtual Machine User's Guide,* SC34-6201.

- **CorbaServer**

  The CorbaServer provides the execution environment for enterprise beans. This execution environment provides a means to logically group enterprise beans within a CICS EJB server. The CorbaServer environment is configured by installing a CORBASERVER resource definition.

- **Deployed JAR files**

  A deployed JAR file is an EJB-JAR file containing enterprise beans on which code generation has been performed, and which has been stored in the HFS used by the CICS region. The DJAR CICS resource definition is used to specify the attributes of the deployed JAR file. Installing the DJAR definition in CICS results in the deployed JAR file (and the classes it contains) being copied to the CICS shelf directory in the HFS. At the same time, CICS reads the deployed JAR file from the shelf, parses its deployment descriptor, and stores the information it contains.

Web access using CICS EJB support

In this section we discuss the different ways that the EJB support in CICS can be used to enable Web access to CICS applications. We also discuss how these applications can be enhanced by adding business components which can invoke services within WebSphere or enable database access.

Session beans in CICS

Figure A-2 on page 314 shows how you can first provide Web access to existing CICS applications by developing a session bean which uses the CICS Connector for CICS TS or the JCICS classes to invoke the existing application. In this scenario, the Web presentation logic for the application can be provided by developing servlets and JSPs which can be deployed within WebSphere on z/OS or on a distributed platform. For further details on this, refer to Chapter 10 in the redbook *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2*, SG24-6284.

Database access

Having provided Web access to the application, new functionality can be added, or the existing application can be rewritten to use session beans and JDBC, SQLJ, or Data Access Beans to provide database access. For details on developing a session bean using these methods, refer to Chapter 12 in the redbook *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2*, SG24-6284.

Entity beans

CICS TS V2 does not provide support for entity beans. However, from CICS, you can access entity beans on any other EJB server that does support them. WebSphere Application Server does support entity beans, and therefore new components can be added to your session beans to invoke entity beans in WebSphere Application Server. One reason why you would run your session beans in CICS and entity beans in WebSphere Application Server is that CICS is optimized to run pseudo-conversational transactions, which are similar to session beans in the EJB model. WebSphere Application Server, however, is optimized for long-lived data objects, which are similar to entity beans in the EJB model.

Session beans in WebSphere Application Server

It may be appropriate for some business objects to be deployed as session beans in WebSphere Application Server. This may be due to specific resource or platform requirements. In this case, these session beans can be part of an application which encapsulates business logic across session beans running in CICS and in WebSphere Application Server. The WebSphere Application Server, in this scenario, may be running on a distributed platform which has the specific resources required by the CICS application.

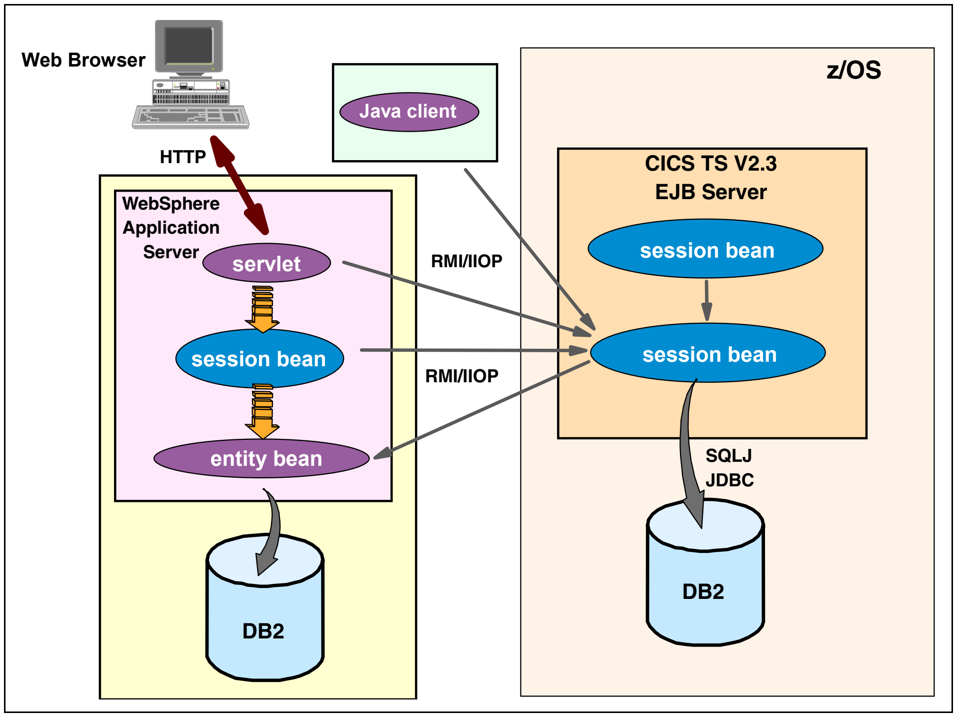


Figure A-2 Web access using CICS EJB support

This flexibility of design enables business components to be deployed where the platform and the products provide the most efficient environment for the business logic embodied within the application.

Security

Accessing enterprise beans requires the use of the IIOP protocol. Support for IIOP is provided by the CICS TCP/IP listener. Figure A-3 on page 315 shows a typical CICS enterprise bean environment.

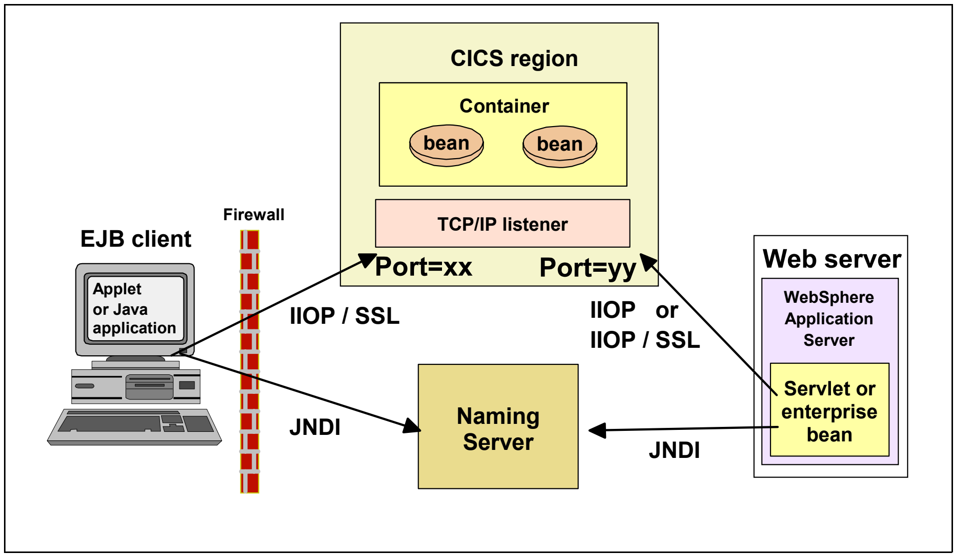


Figure A-3 Using SSL with EJB clients

Applets or Java applications may require the use of SSL. But a servlet or an enterprise bean running in WebSphere may not need SSL security, since both CICS and WebSphere are likely to be in your private network.

Thus providing security for enterprise beans involves:

- Providing an appropriate level of privacy and data integrity (SSL support)
- Confirming the identity of the client (authentication)
- Controlling what the client can do (authorization)

SSL support

To activate SSL support for an *incoming* IIOP request for an enterprise bean, specify one of the following for the value of the SSL parameter of the TCPIPSERVICE definition:

- YES
- CLIENTAUTH

CICS also support *outbound* IIOP requests. This enables an enterprise bean in a CICS EJB Server to invoke an enterprise bean in another EJB Server using Remote Method Invocation over IIOP (RMI/IIOP). This communication can also be encrypted using SSL; the Interoperable Object Reference (IOR) of the remote enterprise bean will specify whether SSL should be used.

The remote server can request CICS to authenticate itself with a client certificate. CICS finds the label of the client certificate that it should use in the CERTIFICATE parameter of the CORBASERVER definition and then obtains the certificate from the KEYRING (as specified in the SIT).

Improved SSL support in CICS TS V2.3 and above

CICS allows you to restrict the cipher suites which CICS advertises for connections that use the *HTTP* protocol. You can use the same values of the PRIVACY parameter of the TCPIPSERVICE definition, namely:

- NOT SUPPORTED
- SUPPORTED
- REQUIRED

This is in order to also restrict the cipher suites which CICS advertises for *inbound IIOP* requests.

To restrict the cipher suites which CICS uses for *outbound* IIOP requests, you may specify one of the same values for the OUTPRIVACY parameter of the CORBASERVER definition.

Authentication

Since CICS authorization is based on a user ID, CICS must derive one from the IIOP request and authenticate it. It can do so by using information associated with or provided by the following:

- An SSL client certificate
- The security user-replaceable module (DFHXOPUS)
- Asserted identity

SSL client certificate

If you are using SSL to send an IIOP message, you can use client certificates to authenticate the user. This is achieved by setting SSL(CLIENTAUTH) and AUTHENTICATE(CERTIFICATE) in the TCPIPSERVICE definition and CLIENTCERT(*tcpipservicename*) in the CORBASERVER definition.

The mapping from a certificate to a user ID exists when:

- The certificate is already registered to a user ID in your external security manager's data base (single certificate).
- The information sent in the client certificate matches a RACF Certificate Name Filtering profile, which allows multiple SSL client certificates to be associated with a single user ID.

Security user-replaceable module (DFHXOPUS)

If no SSL client certificate is provided, you can assign a user ID by coding a user-replaceable module (URM) and specifying its name in the URM parameter of the TCPIPSERVICE definition. If you do not specify a module name, CICS will assign the default user ID set in the DFLTUSER system initialization parameter.

CICS provides the sample DFHXOPUS module, which is a C program. An alternative COBOL version (named COBXOPUS) is available as additional material for the redbook *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2*, SG24-6284.

Since it is possible for a CICS region to have several TCP/IP listeners, we can use different URM modules. The URM should return a pointer to an 8-byte character field in its COMMAREA, and CICS will associate the user ID in that field with the server transaction.

The URM can use a variety of designs to choose the correct identification; possibilities include:

- Passing a user ID and password in the IIOP message and performing custom authentication using the `EXEC CICS VERIFY PASSWORD` command.
- Extracting user information, such as the client TCP/IP address and port number, using the `EXEC CICS EXTRACT TCPIP` command.
- Mapping user IDs based on enterprise bean names and method names.
- Assigning a single user ID for all IIOP messages.

Asserted identity

Assume that an HTTP request drives a servlet in WebSphere which then invokes a method on an enterprise bean in CICS, as shown in Figure A-4 on page 318.

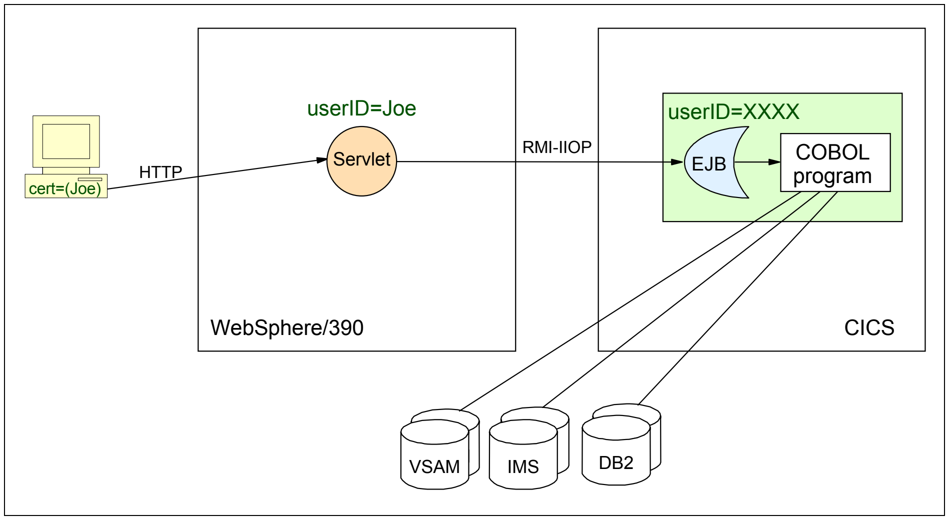


Figure A-4 Invoking a bean method through an intermediate server

In CICS TS V2.2, CICS does not recognize that there is a server on the other end of the IIOP connection. WebSphere may present a client certificate, but it is not the end-user's certificate. Rather, it is WebSphere's own certificate (and thus WebSphere's identity). So the work in CICS will execute under some user ID other than "Joe." This may be the CICS region default user ID, one assigned by the URM, or the one associated with a certificate presented by WebSphere.

IBM has developed a proprietary protocol called Asserted Identity to address this problem. Implemented by WebSphere V4 and above and CICS TS V2.3 and above, Asserted Identity will allow "Joe" to be known in all the servers and his security identification will be passed from WebSphere to CICS.

Asserted identity authentication can be used when an IIOP client communicates with the target server (CICS TS V2.3 and above) through an intermediate server (CICS TS V2.3 and above, WebSphere V4.0 and above), and both servers use the same security manager. It works as follows:

1. The intermediate server's identity is authenticated by the target server using SSL client certificate authentication.
2. Through the security manager, the target server verifies that the intermediate server can be trusted to authenticate its clients.
3. When the intermediate server receives a request, it authenticates the client using whatever authentication protocol is appropriate. If the client is successfully authenticated, the intermediate server passes the request to the target server.

4. Because the target server trusts the intermediate server to authenticate the client, it makes no further checks of the client's authenticity before processing the client's request.

To establish a trust relationship between the intermediate and target servers, where the target server is a CICS CorbaServer, perform the following steps:

1. Configure your CICS region to use SSL authentication and specify `AUTHENTICATE(ASSERTED)` and `SSL(CLIENTAUTH)` in the TCPIPSERVICE definition.
2. Associate the intermediate server's client certificate with a RACF user ID.
3. Create a profile named DFH.*applid.corbaserver.*ASSERTID in the SERVAUTH general resource class, where:
   - *applid* is the APPLID of the CICS region
   - *corbaserver* is the name of the target CorbaServer.

   For example, use the following RACF command:

   ```
   RDEFINE SERVAUTH DFH.applid.corbaserver.ASSERTID UACC(NONE)
   ```
4. Give the intermediate server's user ID (established in step 2) READ authority to the profile. For example, use the following RACF command:

   ```
   PERMIT DFH.applid.corbaserver.ASSERTID CLASS(SERVAUTH) ID(server_user_ID)
   ACCESS(READ)
   ```
5. If the intermediate server is:
   - WebSphere, then specify `Send asserted identities allowed` on its properties form. Fill in the values for the SSL-related elements (SSL RACF-keyring, SSL V2 timeout, and SSL V3 timeout).
   - CICS, then no additional setup is required. CICS will always send asserted identities if the target server is capable of receiving them.

User ID authentication summary

Table A-1 on page 320 summarizes the relation between the TCPIPSERVICE and CORBASERVER parameters, concerning SSL, and shows how the user ID is obtained.

Table A-1 User ID authentication summary

Authentication method	**TCPIPSERVICE AUTHENTICATE parameter**	**TCPIPSERVICE SSL parameter**	**Associated CORBASERVER parameter**	**Client Cert associa-ted with user ID**	**How user ID of IIOP client is identified**
IIOP with no authentication	NO	NO	UNAUTH (*tcpipservice*)	N/A	User ID provided by URM specified on TCPIPSERVICE (2)
IIOP with no authentication	NO	YES	SSLUNAUTH (*tcpipservice*)	N/A	User ID provided by URM specified on TCPIPSERVICE (2)
IIOP with SSL client certificate optional	NO	CLIENTAUTH	SSLUNAUTH (*tcpipservice*)	NO	User ID provided by URM specified on TCPIPSERVICE (2)
				YES	That user ID is used
IIOP with SSL client certificate required	CERTIFICATE	CLIENTAUTH	CLIENTCERT (*tcpipservice*)	NO	Connection is rejected
				YES	That user ID is used
IIOP with asserted identity authentication	ASSERTED	CLIENTAUTH	ASSERTED (*tcpipservice*)	NO (1)	Connection is rejected
				YES (1)	User ID sent in IIOP request by intermediate server
1. Intermediate server's certificate 2. If a program is not specified in the URM parameter of the TCPIPSERVICE definition, then the CICS default user ID is used					

Authorization

Authorization involves verifying that a user is allowed to access a particular method in a bean, and then to access the resources which that method uses.

Associating methods with transaction IDs

CICS runs each method call as a CICS transaction. The association between a method and a CICS transaction is specified in a REQUESTMODEL resource definition. A CICS-supplied transaction, CREA, enables you to generate REQUESTMODEL definitions that can be installed dynamically into the CICS region, or written to the CICS system definition file (CSD), or both. CREA displays a list of beans and bean methods within the JAR file related to an installed DJAR resource definition, and enables you to associate transaction IDs with those beans and methods. Once you have finished associating transaction IDs with beans and methods, a list of request models, which best match the transaction IDs with the beans and methods in the JAR file, is generated. Each REQUESTMODEL definition is presented to you with the option to:

- Install the REQUESTMODEL in CICS
- Define the REQUESTMODEL to the CSD
- Both install and define the REQUESTMODEL
- Ignore the REQUESTMODEL

Next, a list of the REQUESTMODELs that you have installed and defined is displayed. Any REQUESTMODELs installed in CICS that are not used by the DJAR (but refer to beans within the DJAR) are also shown. Similarly, any transaction IDs that are used but not currently installed are listed as well.

Matching method calls to REQUESTMODELs

Figure A-5 on page 322 illustrates the process of matching a method request to a REQUESTMODEL. CICS uses the bean name, the interface type (home or remote), the CorbaServer name and the method name (in this order) to match the incoming method request with a REQUESTMODEL. A more specific match overrides a generic match. If you do not define a REQUESTMODEL, CICS uses the default REQUESTMODEL, which always maps the method name to transaction CIRP. The CIRP transaction points to the default request processor DFJIIRP.

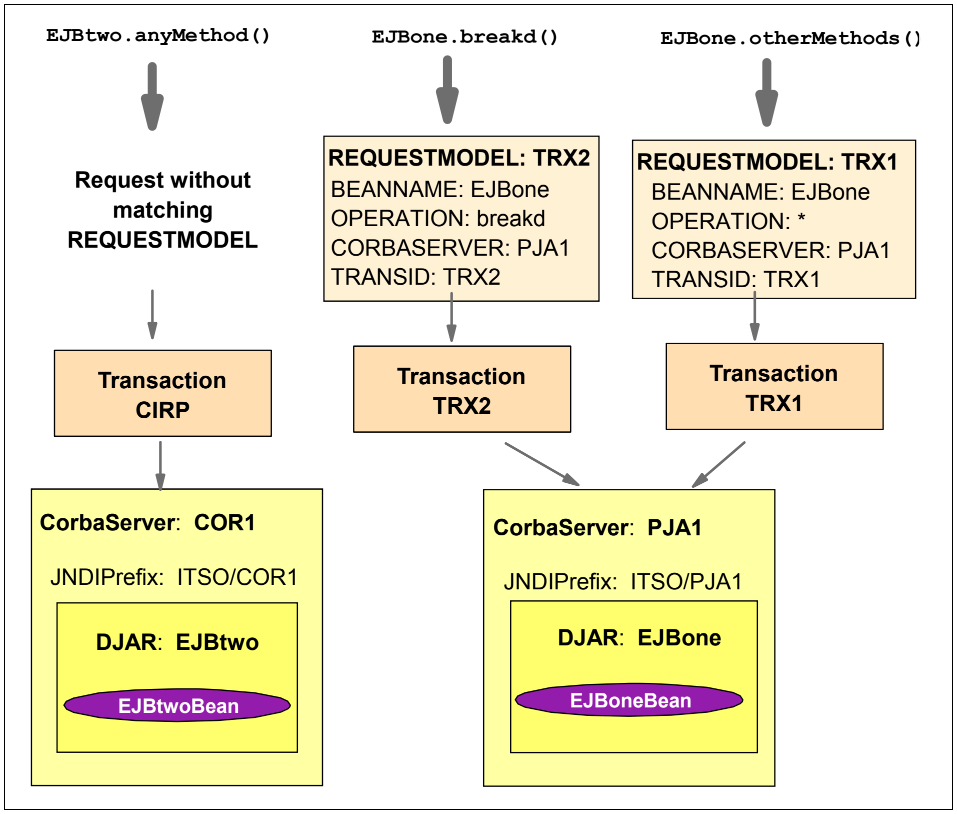


Figure A-5 Using REQUESTMODELs to map a Bean method to a transaction ID

In your external security manager, you would have to give access to the transaction CIRP to the user ID associated with any request for a method in EJBtwo. Similarly, you would have to give access to transaction TRX2 to the user ID associated with a client request for the breakd() method in EJBone, and access to transaction TRX1 to the user ID associated with a client request for any other method in EJBone.

Security roles

In the Enterprise JavaBeans Specification v1.1 access to enterprise bean methods is based on the concept of security roles. A security role represents a type of user of an application in terms of the permissions that the user must have to successfully use the application. A role is a logical security identification.

For example, in a payroll application:

- A manager role could represent users who are permitted to use all parts of the application

- A team_leader role could represent users who are permitted to use the administrative functions of the application
- A data_entry role could represent users who are permitted to use the data entry functions of the application

The security roles for an application are defined by the application assembler, and are specified in the bean's deployment descriptor, as shown in Figure A-6.

```
<security-role>
   <description>
       A user with this role may access any method
   </description>
   <role-name>
       Manager
   </role-name>
</security-role>
<security-role>
   <description>
       A user with this role may use administrative functions
   </description>
   <role-name>
       Team_leader
   </role-name>
</security-role>
```

Figure A-6   Definition of security roles in XML deployment descriptor

The security roles that are permitted to execute a bean method are also specified in the bean's deployment descriptor, again by the application assembler. The association between security roles and the bean methods is defined by using method-permission tags. Figure A-7 shows an example of giving the security role Manager access to all the methods in the bean named Payroll.

```
<method-permission>
   <role-name>Manager</role-name>
      <method>
         <ejb-name>Payroll</ejb-name>
            <method-name>*</method-name>
      </method>
</method-permission>
```

Figure A-7   Access rights for the Manager role

In the example, methods which update the hours worked by employees each week might be assigned to the data_entry role, while methods which delete an employee from the payroll might be assigned to the team_leader role.

To distinguish similarly named security roles in different applications, or in different systems, the security roles specified in the bean's deployment descriptor can be given a one- or two-part qualifier when the bean is deployed in a CICS system.

For example:

Security role with no qualifier	`team_leader`
Security role with one qualifier	`payroll.team_leader`
Security role with two qualifiers	`prodcics.payroll.team_leader`

In this example, `payroll` qualifies the security role at the application level and is used to distinguish between the `team_leader` role in the payroll application and the `team_leader` role in other applications. Similarly, `prodcics` qualifies the security role at the system level and is used to distinguish between the `payroll.team_leader` role in the prodcics region and the `payroll.team_leader` role in test regions.

At the application level, security roles are qualified by the **display name**, if one is specified in the deployment descriptor. At the system level, security roles are optionally qualified with a prefix which is specified in the EJBROLEPRFX system initialization parameter.

A security role with its qualifiers is known as a deployed security role. The mapping of individual users or groups to deployed security roles is done in the RACF entity class EJBROLE or the RACF grouping class GEJBROLE.

Using the RACF EJBROLE generator utility

The RACF EJBROLE generator utility (`dfhreg`) is a Java application program that extracts security role information from a deployment descriptor, and generates an output file which contains the RACF commands that define security roles as members of a profile in the GEJBROLE class.

Example A-1 on page 325 shows an example of a Unix System Services (USS) command that invokes the RACF EJBROLE generator utility. In this example:

- `-secprfx PRODCICS` specifies the name used to qualify the security role at the system level. The value you specify must match the value of the EJBROLEPRFX system initialization parameter for the CICS system where the security roles will be used.

- `PayrollSessionBeanEJB.roles` specifies the name of the file to which `dfhreg` will write its output.
- `PayrollSessionBeanEJB.jar` specifies the name of the input file which contains the deployment descriptor.

Example: A-1   Executing dfhreg from a USS command line

```
dfhreg -secprfx PRODCICS -out PayrollSessionBeanEJB.roles
PayrollSessionBeanEJB.jar
```

Example A-2 shows the output file generated by `dfhreg`.

Example: A-2   PayrollSessionBeanEJB.roles file created by dfhreg

```
/******************************************************************/
/*                    RACF EJBROLE GENERATOR TOOL                 */
/******************************************************************/
/* Default enterprise role used to group EJBROLE definitions. */

RDEFINE GEJBROLE DFLTROLE

/* Manager role */
RALTER GEJBROLE DFLTROLE ADDMEM(PRODCICS.Payroll.manager)

 /* Team Leader role */
RALTER GEJBROLE DFLTROLE ADDMEM(PRODCICS.Payroll.team_leader)

/* Data Entry role */
RALTER GEJBROLE DFLTROLE ADDMEM(PRODCICS.Payroll.data_entry)
```

These RACF commands define the three deployed security roles (`PRODCICS.Payroll.manager, PRODCICS.Payroll.team_leader, and PRODCICS.Payroll.data_entry`) as members of the profile DFLTROLE in the RACF grouping class GEJBROLE. You would then use the RACF **`PERMIT`** command to give the appropriate users READ access to the DFLTROLE profile:

```
PERMIT DFLTROLE CLASS(GEJBROLE) ID(user1,user2) ACCESS(READ)
```

Alternatively, you could use the following commands to define deployed security roles in the EJBROLE entity class:

```
RDEFINE EJBROLE (PRODCICS.Payroll.manager) UACC(NONE)
RDEFINE EJBROLE (PRODCICS.Payroll.team_leader) UACC(NONE)
RDEFINE EJBROLE (PRODCICS.Payroll.data_entry) UACC(NONE)
```

Then you could use the RACF **PERMIT** command to give a user access to one security role without giving him access to the other security roles:

```
PERMIT PRODCICS.Payroll.manager) CLASS(EJBROLE) ID(user1) ACCESS(READ)
PERMIT PRODCICS.Payroll.team_leader) CLASS(EJBROLE) ID(user2) ACCESS(READ)
PERMIT PRODCICS.Payroll.data_entry) CLASS(EJBROLE) ID(user3) ACCESS(READ)
```

Application-managed security

Applications can query the session context object to retrieve security information. Normally, this should be avoided, since the objective of EJB is to let the application programmer concentrate on the business logic, but sometimes it may be necessary.

The EJB 1.1 specification provides the following two methods:

- **isCallerInRole(String *roleName*)**

  Returns `true` if the caller is in the security role specified as the single string argument; otherwise `false` is returned.

- **getCallerPrincipal()**

  Returns an object of the class `java.security.Principal` which may then be used to extract more information such as the distinguished name (DN) which is associated with this session. Figure A-8 shows a Java code sample for extracting the distinguished name.

```
// obtain the caller principal
   callerPrincipal = ejbContext.getCallerPrincipal();
// obtain the caller principal’s name
   callerName = callerPrincipal.getName();
```

Figure A-8 Extracting distinguished name

When a Java program calls the `getName()` method, the distinguished name is extracted from the X.509 certificate provided by the client. The distinguished name includes user information like: Common Name, Title, E-mail address, Organization Unit, Organization, Location, State, and Country.

If a certificate is not available, the DFHEJDNX user-replaceable module is invoked to generate the distinguished name, given the following inputs:

- The Common Name, that is, the *user name* associated with the caller’s user ID. The user name is a 20-character name of the user obtained from the external security manager (ESM).
- Certain fields from the CICS *client* certificate, used by outbound SSL and specified in the CERTIFICATE parameter of the CORBASERVER definition.

If a CORBAServer certificate is not available, the default certificate provided in the keyring is used instead.

Design issues

The solution design must consider the security needs, the performance issues, and possible implications of flowing IIOP requests through firewalls that may not provide an IIOP capable proxy server.

Transactional scope

The transaction management rules which are applied to enterprise bean method invocations are based on the Object Transaction Service (OTS) which is part of the CORBA specification.

OTS permits transaction management in a distributed heterogeneous environment. The instigator of an OTS transaction registers the start of the transaction with an OTS Transaction Coordinator which returns a transaction context that uniquely identifies the transaction. The transaction context, which includes a reference to the OTS Transaction Coordinator responsible, is passed as part of the protocol for method invocations on remote objects such as enterprise beans.

As shown in Figure A-9 on page 328, the transaction context is propagated throughout all method invocations on any enterprise beans which are involved during the life of the transaction.

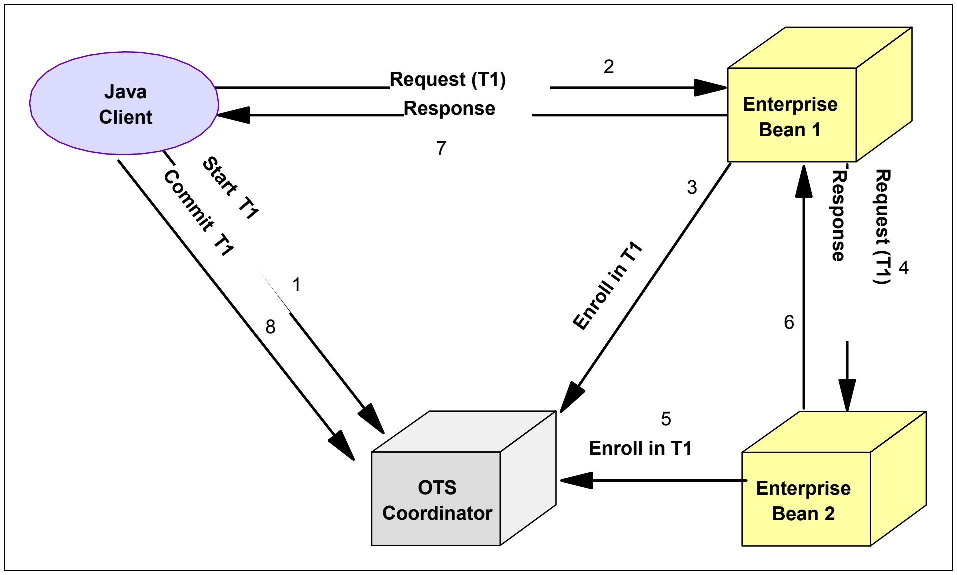


Figure A-9   EJB transaction propagation

That is not to say that the enterprise beans involved necessarily participate in the OTS transaction. The transaction context which is sent with the method invocations is an invitation to participate in the transaction, but the invitation may be declined.

For the remainder of this section we use the term *OTS transaction* to differentiate these transactions from *CICS-transactions*.

When designing a session bean that will run in CICS, the bean provider must decide whether the session bean will demarcate OTS transactions programmatically in the business methods (*bean-managed* transactions), or whether the OTS transaction demarcation is to be performed by the container based on the *transaction attribute* in the deployment descriptor (*container-managed* transactions).

The bean provider of a session bean must use the `transaction-type` element in the deployment descriptor to declare whether the session bean is of the bean-managed or container-managed transaction demarcation type. The `transaction-type` element must be one of the two following:

- `<transaction-type>Bean</transaction-type>`
- `<transaction-type>Container</transaction-type>`

With *bean-managed* transactions, the session bean code demarcates OTS transactions using the `javax.transaction.UserTransaction` interface. All resource manager accesses between the `UserTransaction.begin()` and `UserTransaction.commit()` method calls are part of an OTS transaction. With *container*-managed transactions, the container demarcates OTS transactions using the transaction attribute provided by the application assembler in the deployment descriptor. The transaction attribute is used to tell the container whether it should include the work performed by an enterprise bean method in a client's OTS transaction, run the enterprise bean method in a new OTS transaction started by the container, or run the method with no OTS transaction.

Every client method invocation on a session bean object, via the bean's remote interface is interposed by the container, and every connection to a resource manager used by a session bean is obtained via the container. It is this managed execution environment that allows the container to affect the session bean's OTS transaction management.

We discuss bean-managed transactions further in , "Bean-managed transactions" on page 329,and container-managed transactions in , "Container-managed transactions" on page 330.

Bean-managed transactions

The J2EE specifications provide an application programming interface to allow the application developer to explicitly set OTS transaction boundaries. This interface is called the Java Transaction API (JTA).

The Java Transaction API is used by obtaining an instance of the class UserTransaction from the enterprise bean's session context, and then invoking methods on the instance to request OTS transaction management services. Here are four of the methods provided by the JTA:

begin()	Start a new OTS transaction
commit()	Commit the current OTS transaction
getStatus()	Retrieve the status of the current OTS transaction
rollback()	Roll back the current OTS transaction

Figure A-10 on page 330 shows a fragment of code in which a bean manages its own OTS transaction.

```
public void deposit(int amt) throws AccountException{

 /* get a user transaction object from the session context */
 javax.transaction.UserTransaction userTran = ctx.getUserTransaction();

 /* start the transaction       */
 userTran.begin();

 /* update the balance          */
 balance += amt;

 /* code to update the database */
 /* commit the transaction      */
 try{
     userTran.commit();
 }
 catch(Exception e){
     throw new accountException("error:"+e.toString());
 }
}
```

Figure A-10 An example of a bean-managed transaction

There are two kinds of session bean:

Stateful A stateful session bean has a client-specific conversational state, which it maintains across methods and OTS transactions; for example, a shopping cart object would maintain a list of the items selected for purchase by the user. A stateful session bean that manages its own OTS transactions can begin a transaction in one method and commit or roll it back in a subsequent method.

Stateless A stateless session bean has no client-specific (nor any other kind of) non-transient state; for example, a stock quotation object might simply return current share prices. A stateless session bean that manages its own OTS transactions and begins a transaction must commit (or roll back) the transaction in the same method in which it started it.

Container-managed transactions

We recommend that you leave OTS transaction management to the EJB container in order to simplify your application development. The EJB container uses the transaction attribute to automatically manage the start, enrollment, commitment, and rollback of OTS transactions on behalf of your session bean's methods. The transaction attribute is set in the assembly descriptor section of the

EJB deployment descriptor (that is, the file `ejb-jar.xml`) using the `trans-attribute` element. The `trans-attribute` element associated with a business method specifies how the container must manage transactions for that method when a client invokes the method via the session bean remote interface.

The Enterprise JavaBeans 1.1 specification, defines the possible values for the transaction attribute as: *NotSupported*, *Required*, *RequiresNew*, *Supports*, *Mandatory* and *Never*. Table A-2 shows how the CICS EJB container behaves for different settings of the `trans-attribute` element.

Table A-2 CICS behavior for different values of `trans-attribute`

Transaction attribute	**If client is participating in an OTS transaction**	**If client has no OTS transaction**
NotSupported	OTS transaction is suspended for duration of the method call. CICS starts a unit-of-work and commits at end of method. OTS transaction is resumed when method is completed. Equivalent to SYNCONRETURN behavior.	CICS starts a unit-of-work and commits at end of method. Equivalent to SYNCONRETURN behavior.
Required	CICS registers interest in the OTS transaction with the OTS coordinator.	Container starts a new OTS transaction; CICS functions as the OTS transaction coordinator and commits or rolls back the OTS transaction when the bean method ends.
Supports	CICS registers interest in the OTS transaction with the OTS coordinator.	CICS starts a unit-of-work and commits at end of method. Equivalent to SYNCONRETURN behavior.
RequiresNew	OTS transaction is suspended for the duration of the method call.Container starts a new OTS transaction. CICS functions as the OTS transaction coordinator and commits or rolls back the OTS transaction when the bean method ends. OTS transaction is resumed when the method is completed.	Container starts a new OTS transaction. CICS functions as the OTS transaction coordinator and commits or rolls back the OTS transaction when the bean method ends.

Transaction attribute	If client is participating in an OTS transaction	If client has no OTS transaction
Mandatory	CICS registers interest in the OTS transaction with the OTS transaction coordinator	CICS throws TransactionRequiredException exception
Never	CICS throws Remote Exception exception.	CICS starts a unit-of-work, and commits at end of method. Equivalent to SYNCONRETURN behavior.

Session bean methods always execute in a CICS-task, under a CICS unit-of-work. Even if a session bean method executes under no OTS transaction, any updates that the method makes to recoverable resources are committed only at normal termination of the CICS task, and backed out if there is a need to roll back.

The setting of a method's `trans-attribute` element determines whether or not the CICS task under which the method executes makes its unit-of-work part of a wider, distributed transaction.

Application assembler's responsibilities

The application assembler uses the `container-transaction` elements to define the transaction attributes for the methods of the session bean. Each `container-transaction` element consists of a list of one or more `method` elements, and the `trans-attribute` element. The `container-transaction` element specifies that all the listed methods are assigned the specified transaction attribute value.

The method element uses the `ejb-name`, `method-name`, and `method-params` elements to denote one or more methods of the session bean.

Figure A-11 on page 333 shows an example of the specification of the transaction attributes in the deployment descriptor. The `updatePhoneNumber()` method of the `EmployeeRecord` session bean is assigned the transaction attribute *Mandatory*; all other methods of the EmployeeRecord bean are assigned the attribute *Required*.

```
<ejb-jar>
  ...
  <assembly-descriptor>
    ...
    <container-transaction>
       <method>
          <ejb-name>EmployeeRecord</ejb-name>
          <method-name>*</method-name>
       </method>
       <trans-attribute>Required</trans-attribute>
    </container-transaction>

    <container-transaction>
       <method>
          <ejb-name>EmployeeRecord</ejb-name>
          <method-name>updatePhoneNumber</method-name>
       </method>
       <trans-attribute>Mandatory</trans-attribute>
    </container-transaction>
    ...
  </assembly-descriptor>
</ejb-jar>
```

Figure A-11 Transaction attributes in a deployment descriptor

Performance and scalability

CICS provides support for Enterprise JavaBeans which enables clients such as servlets, session beans, and Java applications to invoke session beans running within the CICS EJB server. These clients use Remote Method Invocation (RMI) over IIOP to invoke business methods on the session beans.

We begin this section by outlining some of the improvements to support for Java in CICS TS V2.3. Next we discuss some things that you should consider when configuring a CICS TS V2.3 and above region to support a particular Java workload. Then we help you estimate how many JVMs you will need and how many JVMs your region will support. After that we discuss ways to balance IIOP requests across CICS listener regions and method requests across AORs. We conclude this section by looking at some network considerations.

Improvements to support for Java in CICS TS V2.3

The improvements to support for Java in CICS TS V2.3 include the following:

- A new execution mode for long-lived Java Virtual Machines (JVMs) called the continuous JVM
- A Shared Class Cache facility which permits Java classes to be cached and shared by a number of JVMs
- Extension of storage protection to Java programs
- Improved algorithms for assigning requests for Java programs to JVMs

We discuss each of these in turn.

The continuous JVM

CICS has a small number of Task Control Blocks (TCBs) which it uses when running application programs. Of these, the Quasi Reentrant (QR) TCB is used to single thread the entire workload. At any point in time there may be a number of transactions running in a CICS region. Only one of these will have use of the QR TCB at any time. If that application needs to pause for some reason, for example, when writing to a data set or waiting for a resource lock, it becomes suspended by CICS. This allows another application to gain control of the QR TCB and carry out some work. This permits CICS to serialize the execution of all the applications it has concurrently running, preventing the actions of one from interfering with those of any others.

This mechanism was developed for applications written in fully compiled languages such as COBOL, PL/1, or C. However, the introduction of support for Java programs presented some additional challenges which CICS has had to address.

Pure Java requires an execution environment provided by a JVM written for the operating system on which it runs. On z/OS, a JVM can sometimes issue blocking calls, such as MVS waits, which cause the TCB used by the JVM to go into a wait state. If a JVM was started on the QR TCB then blocking calls would cause the entire CICS workload to pause. To address this issue, CICS TS 1.3 introduced a new type of TCB, known as the J8 TCB. CICS uses one of these for each JVM it starts. As we will see later, with CICS TS V2.3 there is now a J9 TCB as well.

The JVM that is used by CICS supports multithreaded applications. However, only one application can run in a JVM at any one time. More than one TCB can be used when a single CICS region has to support multiple concurrent Java applications. In this way a CICS region can have a single pool of J8 and J9 TCBs to support its Java workloads.

All new CICS transactions start on the QR TCB. If the transaction makes use of a Java program, then CICS switches to either a J8 or a J9 TCB and runs the program under the control of a JVM there. When the program terminates, or if it needs to access a CICS-managed resource, then CICS switches control back to the QR TCB for that piece of processing. In this way the JVM can pause without interrupting the rest of the CICS workload, so serialization of access to CICS resources and the management of the start and end of all transactions is done using the QR TCB.

There are significant Central Processing Unit (CPU) costs in starting and stopping a JVM. Often these are greater than those incurred from the running of an application within a JVM. If a JVM can be serially reused by a number of CICS programs, then the overhead of starting and stopping the JVM is confined to the first and last program in the sequence, and those in between gain significant performance benefits from this.

However, if a JVM is reused then the potential exists for one Java application program to leave around objects whose state may interfere with the successful execution of programs that later reuse the same JVM. EJBs are inherently self-contained and as such cannot leave state around after their execution. Individual Java applications, on the other hand, tend to treat the JVM where they run as something to which they have exclusive use. They may assume that the JVM will be destroyed when the application finishes, and that all object instances, created during program execution, will be removed as a result of the termination of the JVM. This assumption is not valid when a JVM is serially reused and side effects, resulting from objects remaining in the JVM's storage heap following the termination of one application, may interfere with other applications which later reuse the same JVM.

To address this issue, the IBM Development Kit for z/OS, Java 2 Technology Edition (SDK) provides a JVM with a modified heap structure; this JVM can be driven in an attempt to reset its storage areas between separate program invocations. CICS TS V2 can be configured to make use of this mechanism. Reset processing extends the operation of the JVM but does have some adverse effect on its performance characteristics.

Not all applications cause these problems and, for them, this reset processing is an unnecessary overhead. CICS TS V2.3 and above and the SDK 1.4.1 provide support for another JVM configuration, which offers the potential for reuse, but which places responsibility for reset processing onto the applications which run there. This provides significant performance benefits but does present opportunities for badly behaved programs to interfere with others that reuse the same JVM.

Configuring JVMs for reuse

CICS determines the JVM reuse option from a JVM profile which it reads prior to starting a new JVM. The JVM profile is a text file, stored in the UNIX Hierarchical File System (HFS), which is named in the PROGRAM resource definition for the Java program. CICS keeps track of the reuse setting for each JVM that it manages. The reuse setting is included as part of the Java Native Interface (JNI) call which CICS uses to start a new JVM. When a Java program has finished executing, CICS decides whether the JVM should be called for reset processing, and whether to allow the JVM to persist or to be terminated, based on its reuse property.

Three such settings are supported in CICS TS V2.3 and above. These are defined with the REUSE option in a JVM profile and illustrated in Figure A-12.

- `REUSE=NO`

  This setting causes CICS to terminate the JVM after it has been used once. Such a JVM is referred to as a single-use JVM.

- `REUSE=RESET`

  This setting indicates that the JVM is long-lived and that a controlled reset is required between program invocations. This is referred to as a resettable JVM.

- `REUSE=YES`

  This setting indicates that a JVM is intended for reuse without the reset operation at the end of every program invocation. It is known as a continuous JVM.

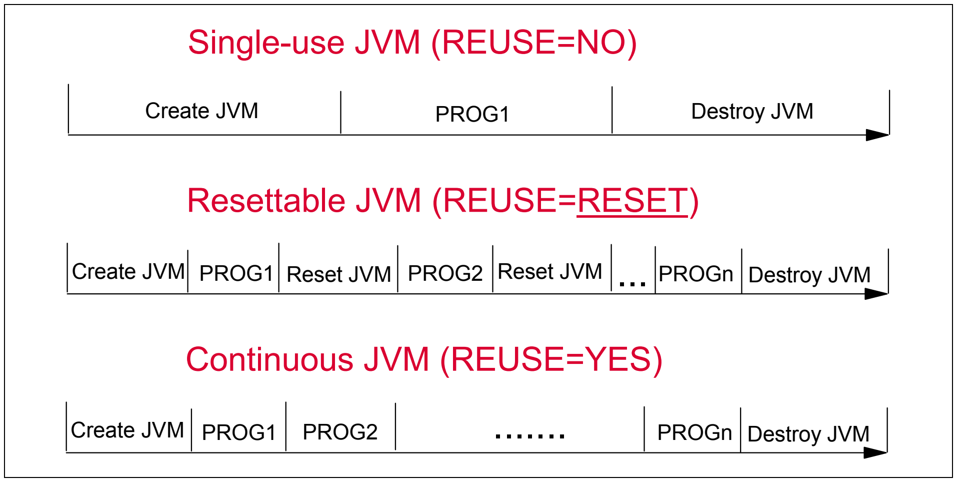


Figure A-12 JVM modes in CICS TS V2.3

Single-use JVMs

Single-use means that a JVM is created for each request, the Java application program is run there, and then the JVM is thrown away. In this way, CICS effectively prevents the actions of one Java program from interfering with any other. This level of transaction isolation is expensive because every application incurs the cost of starting and stopping the JVM, and of loading all of the classes which it needs.

The JVM has a single heap structure, unlike the one used by the resettable JVM, and does not undergo any reset processing prior to termination. This is how Java runs on most other platforms.

The resettable JVM

The resettable JVM provides the potential to run consecutive Java application programs under its control. When an application program finishes executing there, CICS calls the JVM to reset its storage areas, and waits for this operation to complete before terminating the current program. If the reset operation is successful, then the JVM becomes available for use by another CICS program. If the reset fails, then CICS terminates the JVM.

The storage heap used by the resettable JVM is divided up into a number of areas, each with its own class loader. These areas are treated differently during reset processing, as shown in Figure A-13 on page 338. In particular, there is a Transient Heap which contains objects created by a Java application as it executes. All these objects are deleted at reset time. There is also a Middleware Heap, used for objects constructed by middleware classes which are trusted to reset the objects they create to some known state, as part of reset processing.

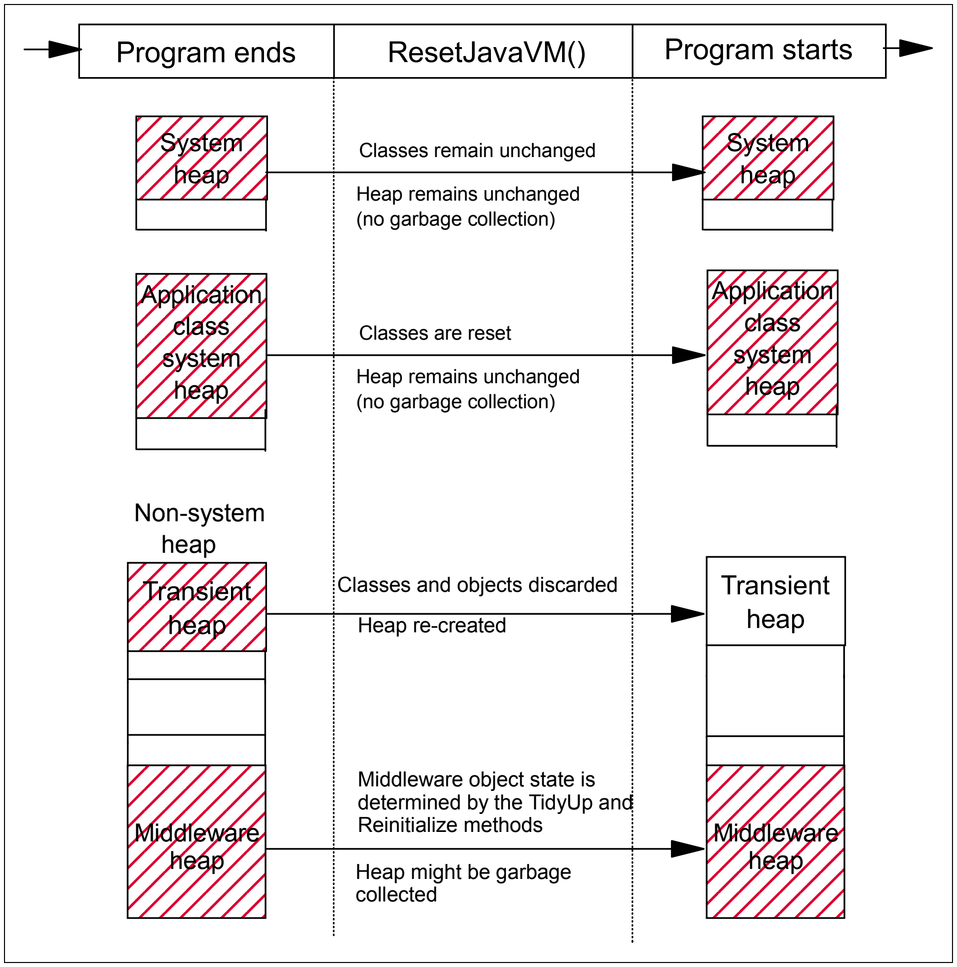

Program ends
ResetJavaVM()
Program starts
System heap
Classes remain unchanged
Heap remains unchanged (no garbage collection)
System heap
Application class system heap
Classes are reset
Heap remains unchanged (no garbage collection)
Application class system heap
Non-system heap
Transient heap
Classes and objects discarded
Heap re-created
Transient heap
Middleware heap
Middleware object state is determined by the TidyUp and Reinitialize methods
Heap might be garbage collected
Middleware heap

Figure A-13 Reset operations against the Resettable JVM's heap structure

This heap structure is extensively documented in the publication *IBM Developer Kit for z/OS Java 2 Technology Edition V1.4 Persistent Reusable Java Virtual Machine User's Guide,* SC34-6201.

Reset processing can fail for a number of reasons, such as a Java application changing one of the JVM's system properties or loading a Native Library. If the reset operation fails, CICS issues a JNI call to delete the JVM. This deletion prevents another application from encountering objects which erroneously remain in the storage heap after a reset failure.

The continuous JVM

This is a potentially long-lived JVM that is started with a non-system heap which is not divided into separate middleware and transient heaps. There is no mechanism for automatically resetting this storage when a Java program finishes execution. Each program is responsible for removing unwanted state data when it terminates, or restoring objects to their initial state prior to completion. Programs can intentionally leave data for subsequent applications to use, and by doing so they make use of a form of caching which is not available through the resettable JVM.

The simplified storage management of the continuous JVM offers large performance benefits over the resettable JVM. However, there are risks associated with running applications which do not conform to the reuse requirements of this mode.

Java programs running in this mode should not carry out actions which alter the JVM's state unless they restore the original values before ending. The most common cause of this is from the inclusion of static storage objects in Java programs. These are not automatically reinitialized on subsequent program invocations and should be used with care when running in this mode. One way to prevent this is by defining all class fields as `final`.

The Shared Class Cache facility

We begin by providing an overview of the Shared Class Cache and then we describe the benefits of using it.

Overview

The SDK V1.4 provides a mechanism which allows Java classes to be cached centrally and shared between different JVMs. CICS TS V2.3 introduces a shared class cache facility which extends this function to some, or all, of the JVMs that it controls. See Figure A-14 on page 340.

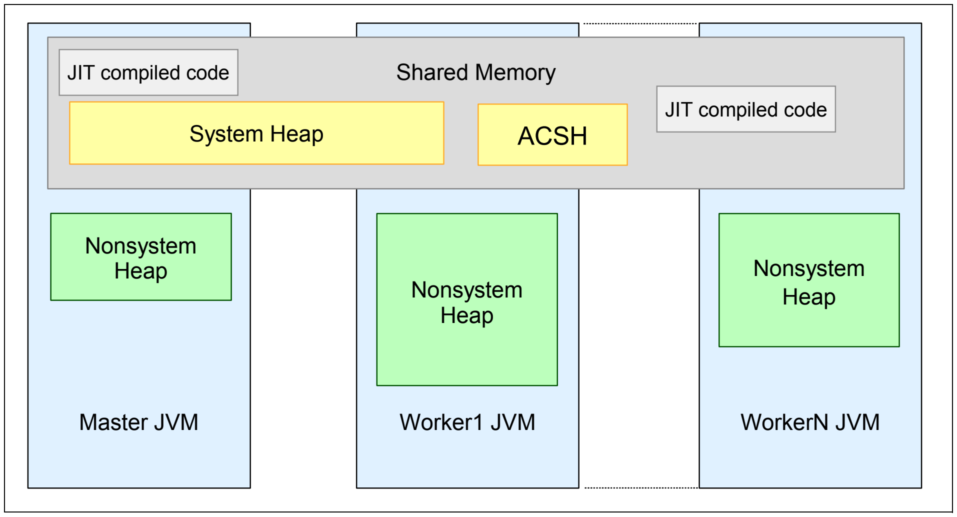


Figure A-14   One view of the Shared Class Cache

The collection of JVMs which use this facility is referred to as a JVMset. This consists of a single JVM, known as the *master*, whose role is to manage the shared class cache, together with other JVMs, called *workers*, which service individual Java requests. These requests can be to run a Java program, an EJB, or an IIOP application. Figure A-15 on page 341 shows another view of the relationship between master and worker JVMs within a CICS region's JVM pool; in this figure DFHJVMPR, DFHJVMPC, USERJVM1, and USERJVM2 are the names of JVM profiles.

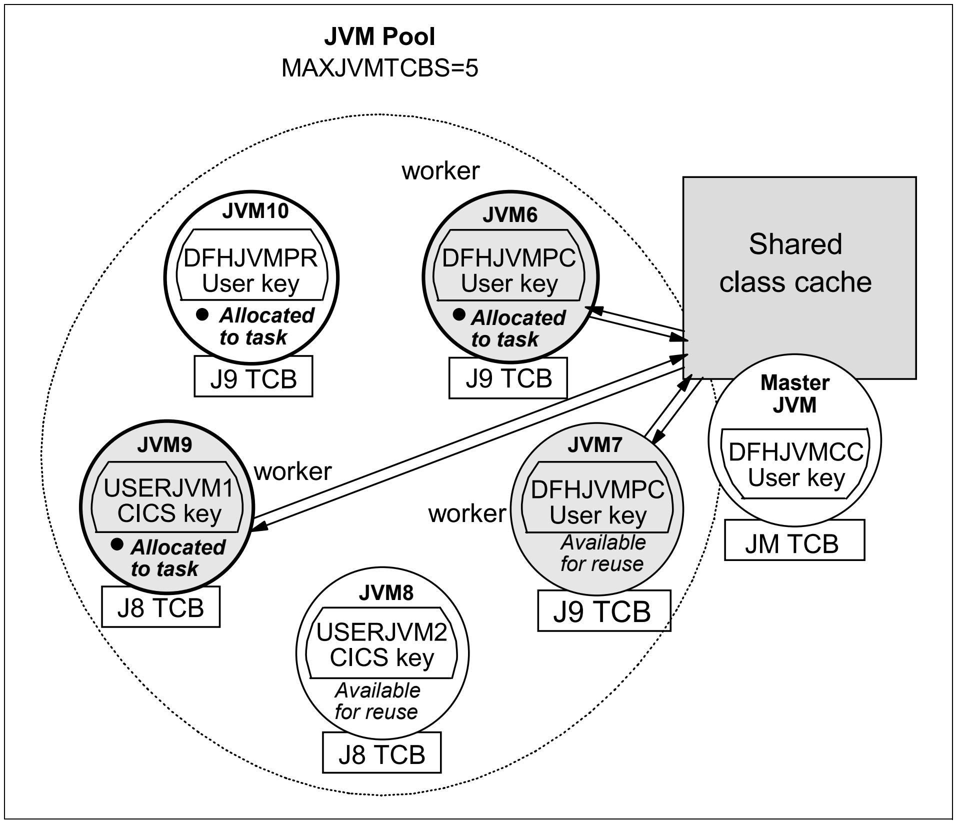


Figure A-15 Another view of the shared class cache

A CICS region can only have a single active shared class cache which uses this feature. CICS controls the launching of the master JVM and any workers that are needed to service requests to run Java components. Individual worker JVMs in a JVMset can have different characteristics, defined in their associated JVM profiles and JVM properties files. These may be used to define the size of a JVM's storage heaps, to set the trace options it uses, or to add some security controls.

Some characteristics of a JVMset are common to all of its JVMs. They are taken from the JVM profile of the master JVM and include the REUSE option discussed earlier. A master JVM is potentially long-lived and as such must be started in either resettable or continuous mode. The master JVM cannot be configured with REUSE=NO and, as a result, worker JVMs cannot run in this mode either. Figure A-16 on page 342 shows a master JVM running with REUSE=RESET. Therefore, the worker JVMs must also run with REUSE=RESET. However, the standalone JVMs (JVM8 and JVM10) may run with any of the REUSE options.

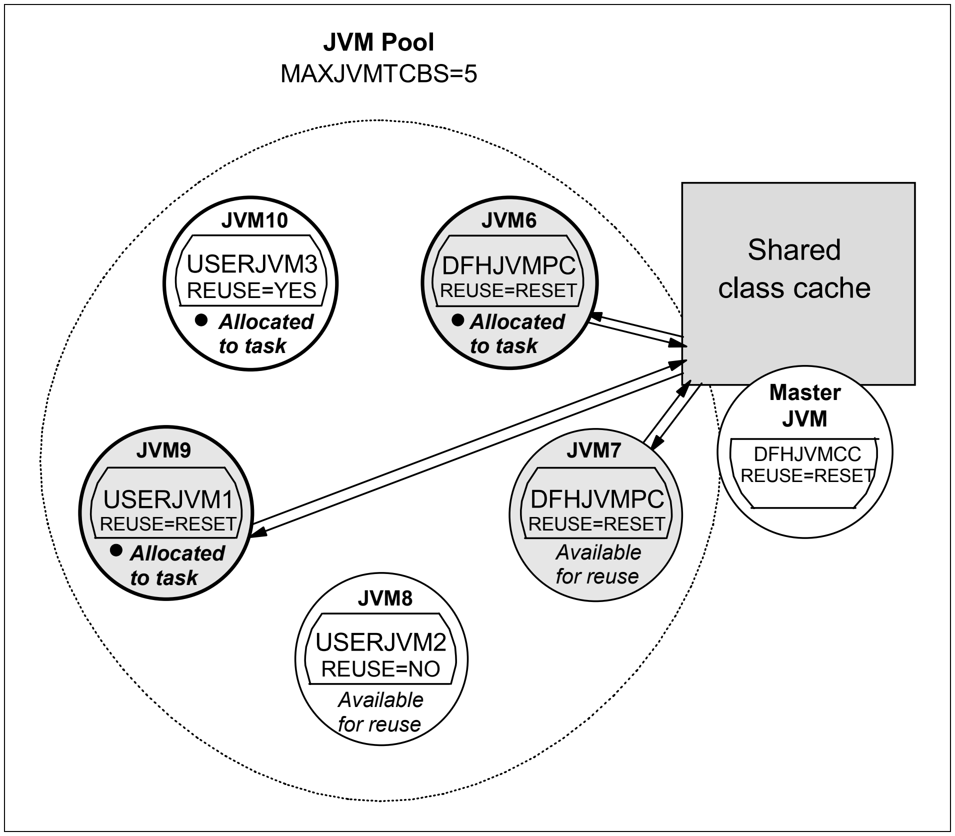


Figure A-16 REUSE and the shared class cache

Benefits of using the shared class cache

The shared class cache facility offers a number of benefits to customers. Java classes are loaded once per CICS region rather than once per JVM, reducing the class loading overhead for all the workers in a JVMset. It also reduces the overall storage requirement for the JVMset, by having one copy of each class in the cache area, instead of one in each worker JVM's storage heap. Because worker JVMs run in either resettable or continuous mode they receive the performance benefits described earlier, when the JVM is reused for a number of programs. Early internal IBM testing showed that startup for worker JVMs is at least 60% better than for CICS TS V2.2 standalone JVMs and the storage footprint is down by over 30%.

Storage protection for Java programs

Storage protection prevents a user application from inadvertently overwriting CICS control blocks or its own programs. Prior to CICS TS V2.3, all Java programs ran in CICS key 8 storage areas. They made use of JVMs which were started on a separate open TCB, known as a J8 TCB. As a result, the CICS storage protection mechanism could not be used with Java programs. In CICS TS V2.3, customers can choose to run Java programs in either CICS or User key. Those selected to run in User key do so by using JVMs which are started on a new open TCB: the J9 TCB.

The storage key information is derived from the Java program's PROGRAM resource definition. This defaults to User key if not explicitly defined there. However, if storage protection is turned off in the CICS region, then all Java applications will run on JVMs which use J8 TCBs.

Assigning requests to JVMs

A number of factors influence the way that a CICS region matches Java requests with the JVM resources it controls. Options in the CICS PROGRAM resource definitions, the JVM profiles, and the System Initialization Table (SIT) all affect the running of Java workloads. This section explains how these options can be used to configure the behavior of a CICS region.

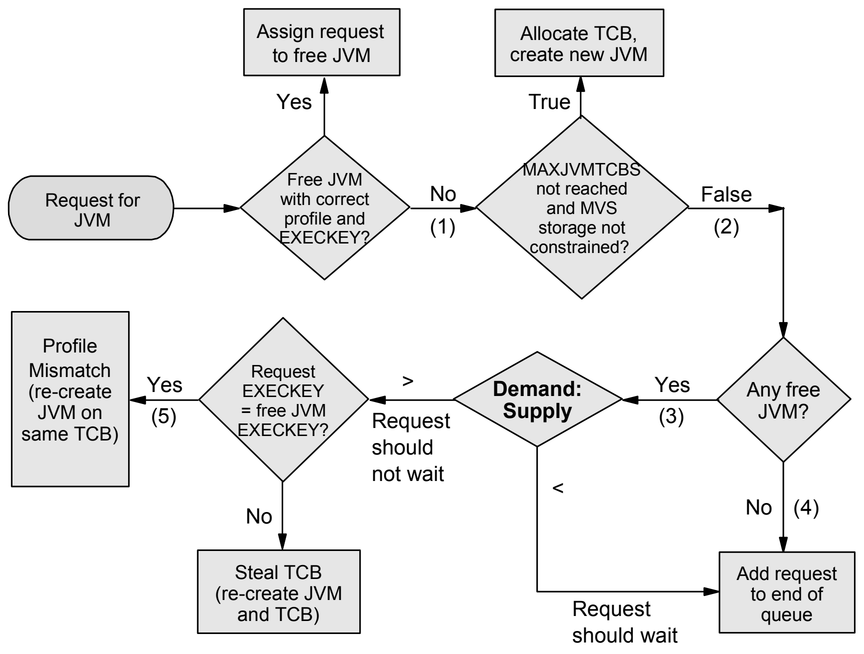
Each CICS region has a maximum number of TCBs that it can use for its Java workload. This value is defined using the MAXJVMTCBs option in the SIT. The default value of 5 TCBs can be overridden during CICS initialization or by using System Programming Interface (SPI) commands or the CEMT transaction.

One TCB is used for each JVM. The set of JVMs that a CICS region controls is referred to as the JVM pool. The maximum pool size is equal to the value of MAXJVMTCBS. The master JVM is started using a special TCB, the JM TCB, and is not considered as part of the overall JVM pool size.

With storage protection active, the JVM pool can contain a mixture of JVMs running with J8 and J9 TCBs at the same time. Those that have REUSE=NO run for one transaction and are then removed from the JVM pool. Others can remain there for some time. When a pool contains a number of long-lived JVMs, then at any time some of these may be servicing program requests while others are waiting for work.

The JVM pool can contain a number of JVMs which have been started with the same JVM profile. JVMs started with different JVM profiles are distinguished from each other by CICS, even if different JVM profiles contain identical sets of options.

Figure A-17 shows the algorithm that CICS uses to satisfy a request for a JVM.


Assign request to free JVM
Allocate TCB, create new JVM
Yes
True
Request for JVM
Free JVM with correct profile and EXECKEY?
No
(1)
MAXJVMTCBS not reached and MVS storage not constrained?
False
(2)
Profile Mismatch (re-create JVM on same TCB)
Yes
(5)
Request EXECKEY = free JVM EXECKEY?
>
Request should not wait
Demand: Supply
Yes
(3)
Any free JVM?
<
No
(4)
No
Steal TCB (re-create JVM and TCB)
Request should wait
Add request to end of queue

1. One of the following is the case:
   - There are no free JVMs
   - There is a free JVM with the wrong profile
   - There is a free JVM with the wrong EXECKEY
2. One of the following is the case:
   - System has reached MAXJVMTCBs
   - System is MVS storage constrained
3. This free JVM has one of the following:
   - Wrong profile
   - Wrong EXECKEY
4. Since we don't have any free JVMs and cannot create any more, we add the request to the end of the queue of requests waiting for a JVM to become free
5. Since we have a free JVM with the right EXECKEY, the profile of the free JVM must be wrong. Thus we can keep the TCB but we must destroy the free JVM and recreate one with the correct profile.

Figure A-17 Algorithm CICS uses to satisfy a request for a JVM

When a new request for a JVM arrives, CICS attempts to match the storage key and the JVM profile name, taken from the Java program's PROGRAM resource definition, with that of a JVM which it has within the JVM pool. If a match is found and that JVM is free, then CICS dispatches the new request to it. If no match can be found, or if all JVMs that are suited to the request are currently carrying out other work, then CICS looks to see if the JVM pool has reached its maximum size and if CICS is MVS storage constrained. If neither is the case, then a new JVM is started to service this request. However, once the pool is full or CICS is MVS storage constrained, no more JVMs can be created. If there are no free JVMs or if there is a free JVM which has the wrong profile or EXECKEY and the demand for that type of JVM is less than the supply, then the new request is placed on the end of a work queue for processing when a suitable opportunity arises. Figure A-18 shows the results of applying this algorithm to two requests for a JVM.

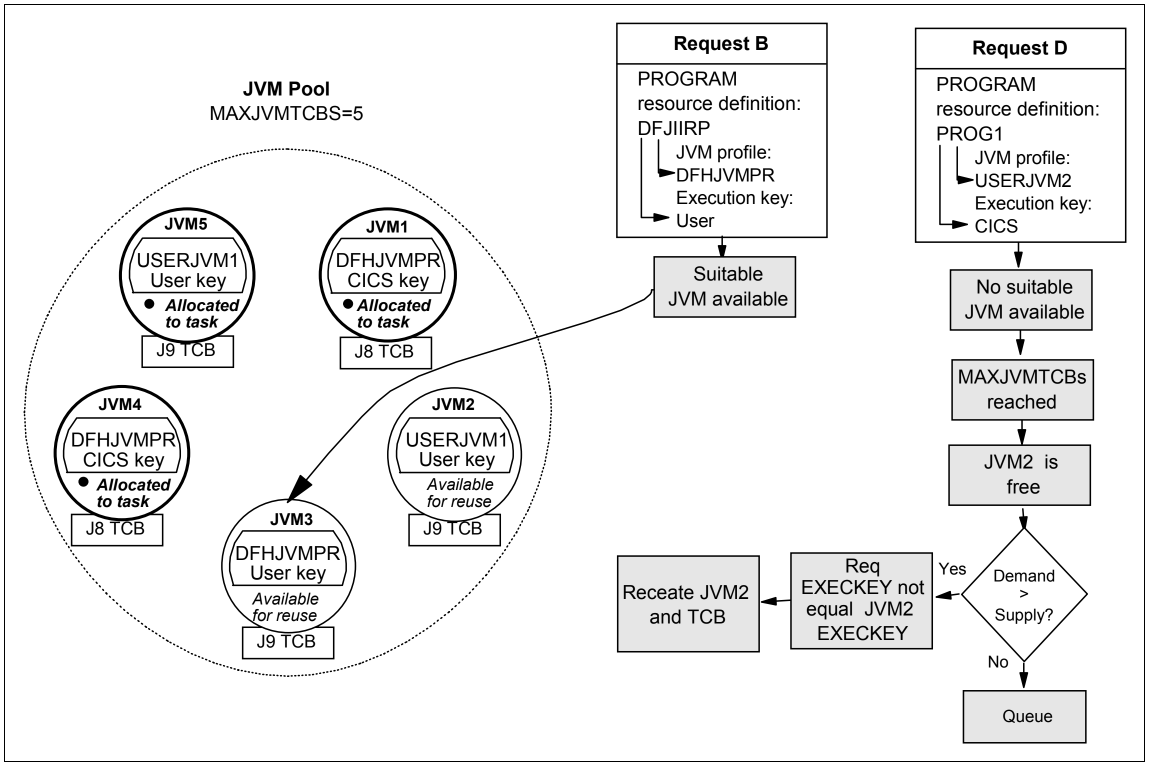


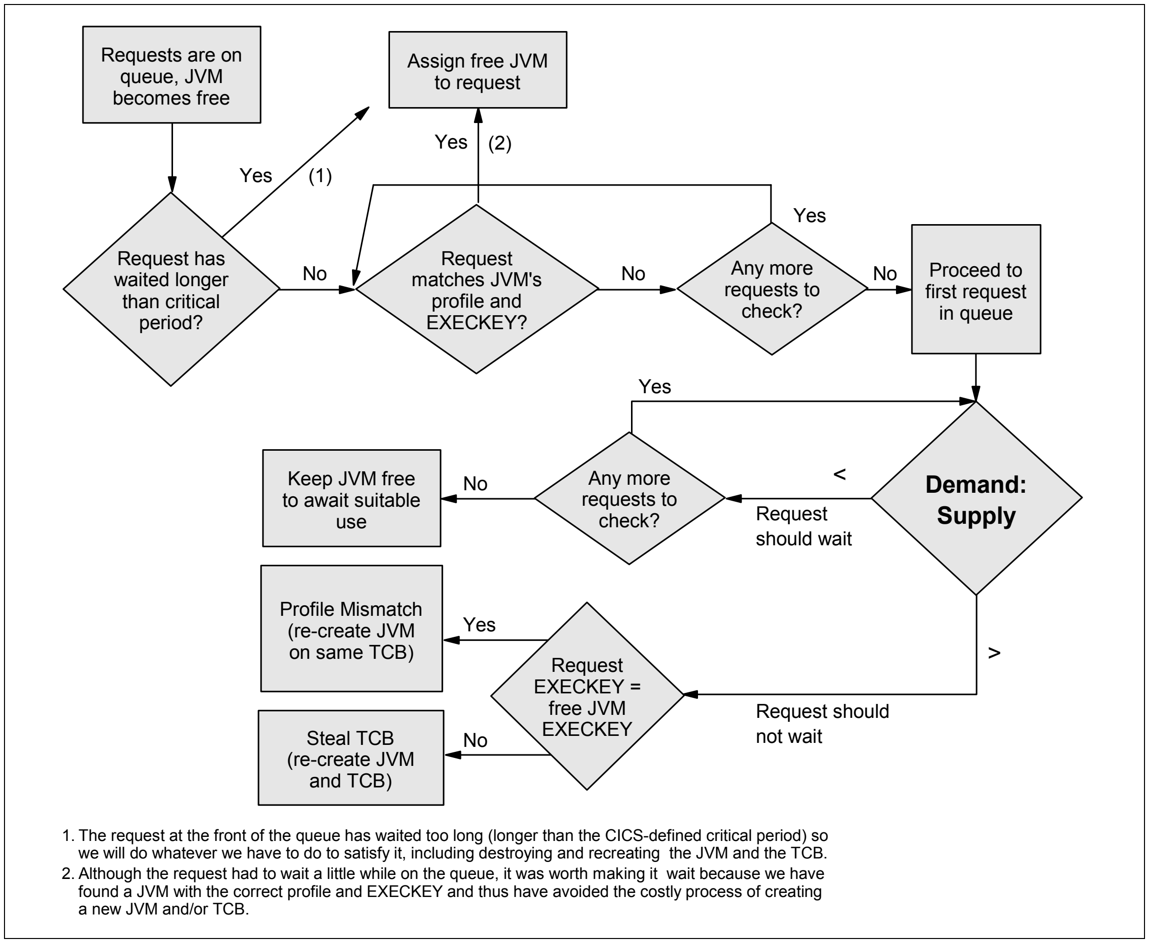
Figure A-18 Two requests for a JVM

Request B wants to run program DFJIIRP. The PROGRAM resource definition for DFJIIRP specifies a JVM profile of DFHJVMPR and an execution key of User. JVM3 in the JVM pool runs with the desired JVM profile and execution key and JVM3 is available for use, so CICS assigns request B to JVM3.

Request D wants to run program PROG1. The PROGRAM resource definition for PROG1 specifies a JVM profile of USERJVM2 and an execution key of CICS. There are no JVMs in the JVM pool running with the desired JVM profile. Furthermore, since there are already five JVMs in the JVM pool and MAXJVMTCBs is five, CICS will not create a new JVM. However, JVM2 is available for reuse, so CICS uses an algorithm to calculate the demand for, and supply of, JVMs with a profile of USERJVM2 and an execution key of CICS. If the demand is *greater* than the supply, then request D should not wait so CICS will compare request D’s execution key with the execution key of JVM2. Since they are not equal, CICS will destroy JVM2 and its TCB and recreate JVM2 on a new TCB. If the demand is *less* than the supply, then request D can wait and CICS adds request D to the end of the work queue.

When the JVM pool is full, it may have some JVMs which match the requirements of a new request, but which are currently active. There may be others which are not active, but which do not match the JVM profile and/or the storage key requirements of the new request. CICS makes use of an elaborate algorithm to match requests in its work queue with its JVMs. CICS can destroy a JVM which is free, and then start a new one using a different JVM profile on the same TCB. Alternatively, CICS may recycle both TCB and JVM.

Figure A-19 on page 347 shows how CICS handles the requests in its work queue when a JVM becomes free.


Requests are on queue, JVM becomes free
Request has waited longer than critical period?
Yes
(1)
No
Assign free JVM to request
Yes
(2)
Request matches JVM's profile and EXECKEY?
No
Any more requests to check?
Yes
No
Proceed to first request in queue
Demand: Supply
<
Request should wait
Any more requests to check?
Yes
No
Keep JVM free to await suitable use
>
Request should not wait
Request EXECKEY = free JVM EXECKEY
Yes
Profile Mismatch (re-create JVM on same TCB)
No
Steal TCB (re-create JVM and TCB)

1. The request at the front of the queue has waited too long (longer than the CICS-defined critical period) so we will do whatever we have to do to satisfy it, including destroying and recreating the JVM and the TCB.
2. Although the request had to wait a little while on the queue, it was worth making it wait because we have found a JVM with the correct profile and EXECKEY and thus have avoided the costly process of creating a new JVM and/or TCB.

Figure A-19 Algorithm CICS uses when a JVM becomes free

When a JVM finishes its current work assignment, CICS looks at all outstanding requests on the work queue. Priority is not always given to the oldest item, and CICS may decide that some other request is a better use for this JVM. However, once a request has been on the work queue for a prolonged period it will be given priority over all other requests, thus preventing it from being overlooked indefinitely.

The termination and startup of a JVM is expensive, both in terms of the CPU these processes consume, and in the slowing of response rates for individual requests. There is a larger overhead when a TCB has to be recycled as well as a JVM.

The request matching process will reduce the number of JVM and TCB recycle events. The delay this might cause for individual requests which are made to wait for a JVM that matches their precise requirements is often less than the time it takes to recycle a JVM and its TCB.

Recommendations for configuring to support a Java workload

There can be large differences between the needs of individual workloads. As a result, there is no single configuration which matches the needs of all customers who use Java within CICS. Therefore, we now offer advice to help customers select the best configuration for their specific needs.

Reuse settings

Selection of the REUSE setting for a particular JVM is largely influenced by the applications it is to service. You might not thoroughly understand the reuse characteristics of *existing* applications, whether they have been written in-house or have been provided by another party. You should extensively test those which are not understood to determine if they can be used with a persistent reusable JVM. When developing *new* CICS Java applications, you should give consideration to how they may eventually be run within a long-lived JVM.

Another factor which may influence this decision is whether the application is running in a test or production environment. In production, a mixture of performance and system stability criteria apply. In a test system, the emphasis might be more on examining the behavior of a new application rather than getting it to run quickly. As a result you may want to develop an application in a single-use JVM and then move it to a resettable or a continuous JVM for system testing before transferring it to the most suitable environment for the production system.

The *single-use* JVM is best suited to a development environment. Other programs running in CICS cannot inadvertently affect an application running in a single-use JVM. Since the JVM will load fresh copies of classes each time a program is run, it will automatically pick up code changes. You may find that helpful when you develop new applications where the classes are being modified frequently so the developer needs to pick up the latest version each time he runs a test.

There is seldom a role for single-use JVMs in production systems. The overhead of starting and stopping them for each application makes them unattractive for all but infrequently-used applications. If the bulk of the workload runs on one set of JVMs with the occasional execution of some other work which requires a JVM with a different profile, there may be some advantage to using a single-use JVM for the rarely run applications as the JVM is automatically thrown away leaving a spare slot in the JVM pool for the main part of the workload to use. However, the majority of applications run in a production system require the performance gains which persistent reusable JVMs offer.

The *resettable* JVM provides the benefits of reuse while offering the greatest protection between program invocations. Applications whose reset characteristics are not well understood, or which do not comply with those needed by the continuous JVM, should be run in this mode. Some performance overhead will result from management of the non-system heap.

An application run in a resettable JVM may take actions which prevent the reset processing operation from completing successfully. This will result in the JVM being destroyed, and CICS may have to start up a new JVM for later requests before it can process them. Such unresettable events will adversely affect a Java workload as a whole. As discussed in "Unresettable events" on page 355, you can use trace options with the resettable JVM to help you determine the causes of these events; when you have found them, you should take remedial action so that the reset operation completes successfully.

The continuous JVM offers significant performance benefits over a resettable or single-use JVM. *Existing* applications whose behavior is not well-understood are better run in the resettable JVM unless you can extensively test them in continuous mode before placing them in a production system. *New* applications should be written in such a way that they can receive the performance benefits of the continuous JVM without unduly affecting other CICS transactions which reuse the same environment.

If the enterprise bean developer follows the Enterprise JavaBeans Specification, an enterprise bean will not leave state around after it has finished executing. Therefore, you should be able to run your EJBs in a continuous JVM.

Sharing cached classes

Your decisions about whether and how to use the shared class cache facility will be more affected by the scale and scope of the Java workload that your CICS region supports than by the nature of the individual transactions. If a system has more than one JVM running in it at any time supporting a common workload, then there are benefits to be gained from the use of shared classes, as we discussed earlier. As the number of duplicate JVMs increases, the benefits increase as well.

A single CICS region can only have one *active* JVMset. All the JVMs that make up a JVMset must be running either in continuous mode or resettable mode. If a Java workload makes use of a mixture of continuous and resettable JVMs, then you will need multiple CICS regions if both the continuous JVMs and the resettable JVMs are to make use of the shared class cache facility.

If you have a mixed workload which uses different JVMs for different applications, you will have to suitably size the shared class cache facility to hold the classes of the combined workload rather than that for an individual worker. Classes are not removed from the shared cache when workers are recycled for different workloads. The starting of new workers may result in different classes being loaded into the shared class cache, and this will only be possible if it has space for them.

If you are using the Just-In-Time (JIT) compiler, then you will also need space in the shared class cache for the JIT code. Internal IBM testing showed that specifying *java.compiler=jitc* in the JVM properties files of both the master and worker JVMs significantly improved performance.

You specify the size of the shared class cache by setting the SIT JVMCCSIZE parameter. When you are trying to choose a value for JVMCCSIZE, the CICS performance group recommends that you overallocate rather than try to calculate the size by adding up the sizes of the components of the shared class cache. Then use the CEMT INQ CLASSCACHE command and the Class Cache section of the DFH0STAT JVMPOOL report to monitor the results.

JVM recycling events

The recycling of JVMs in the JVM pool is controlled by CICS, but you can have some influence over how often CICS recycles them. In a busy system, recycling events will have an impact on the overall performance of the system and should be kept to a minimum.

If an entire Java workload can be run in a particular storage key then TCBs will never have to be recycled. If all of the Java programs also share a common JVM profile, then the recycling mechanism is in effect disabled as all queued requests will match all JVMs in the pool.

The use of a different JVM profile for every application in a workload will not usually be necessary. If you use two JVM profiles with different names but identical options, you will unnecessarily increase the potential for recycling events to take place in the JVM pool. To avoid this you should use a common JVM profile.

You might also investigate whether you could combine compatible options in different JVM profiles to create a single JVM profile. For example, if you found two infrequently-used JVM profiles that contained similar options, but one specified a larger non-system heap size, you could consider combining these into a single JVM profile that specified the larger non-system heap size. This would mean that some applications would be using an unnecessarily large JVM, but the reduction in the incidence of mismatches and steals might make this worthwhile.

You can use a small number of JVM profiles to combine Java applications into groups so that individual JVMs can be used to run any of the applications in a group rather than only one type of application. If the number of JVM profiles is small compared to the size of the JVM pool and the number of requests against each of these JVMs is roughly the same, then little recycling should take place. On the other hand, if the number of JVM profiles is large compared to the number of JVMs in the JVM pool or if some JVMs are used less frequently than others, then recycling events are more likely to occur. As a result, we recommend that you keep the ratio between the number of JVM profiles and the number of JVMs in the JVM pool as low as possible.

If you have a common profile for those applications which require the same options and thus allow a range of applications to run in a standalone JVM, the size of the application class system heap for that JVM will end up a lot larger than if it was used for a single application. As a result the collective heap size of the JVM pool will end up being larger than if it was made up of JVMs supporting single applications. You can use the shared class cache facility to centralize much of this heap and remove the duplication of class data across the members of the JVM pool.

JVM storage management

A JVM in CICS runs as a UNIX System Services (USS) process in a Language Environment® (LE) enclave created using the LE pre-initialization module, CEEPIPI. (Using LE pre-initialization services allows CICS to initialize an environment once, perform multiple executions using that environment, and then explicitly terminate that environment). The JVM uses LE services rather than CICS services to obtain storage. As a result, all storage obtained by the JVM is MVS storage which resides within the CICS address space but outside of the CICS dynamic storage areas (DSAs).

Each JVM executes within its own LE enclave. The JVM's system heap, application class system heap, middleware heap, and transient heap are allocated from the **heap** storage of the LE enclave. You set the sizes of these heaps using parameters in the JVM's profile as shown in Table A-3 on page 352.

Table A-3 JVM profile parameters which set heap sizes for the JVM

Parameter	Meaning
Xinitsh	Initial size of system heap
Xinitacsh	Initial size of application class system heap
Xms	Initial size of middleware heap
Xinitth	Initial size of transient heap
Xmx	Maximum total size of non-system heap (which consists of the middleware and transient heaps)

The LE enclave for each JVM needs to contain not only the heaps shown in Table A-3 but also a basic amount of storage for each JVM as shown in Figure A-20.

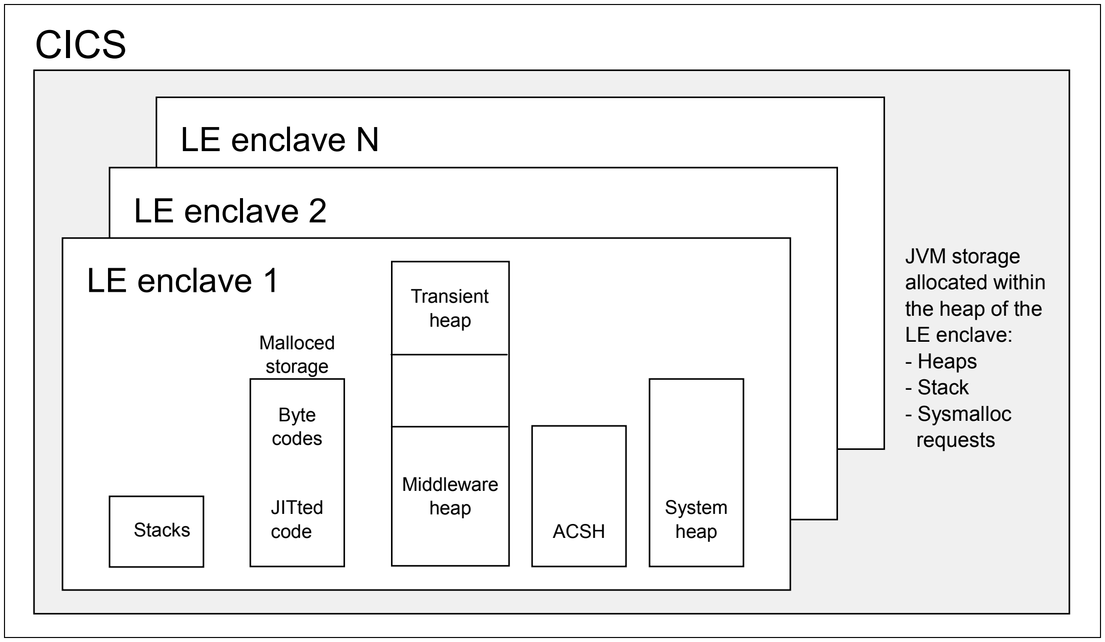


Figure A-20 Each JVM has an LE enclave

The CICS JVM domain enforces some of the LE runtime options for this enclave; CICS takes other options from the DFHJVMRO user-replaceable module. Table A-4 on page 353 shows the runtime options contained in the CICS-supplied version of DFHJVMRO.

Table A-4 LE runtime options used by CICS for the JVM enclave

LE runtime option	Value in CICS-supplied DFHJVMRO
Library heap storage that is not restricted to a location below 16 MB	ANYHEAP(4K, 8176, ANY, FREE)
Library heap storage that must be located below 16 MB	BELOWHEAP(4096, 2048, FREE)
Storage for user-controlled dynamically allocated variables	HEAP(4M, 1M, ANY, FREE, 0K, 4080)
Library stack storage	LIBS(8, 900, FREE)
Controls the allocation of the thread's stack storage for both the upward and downward- growing stacks	STACK(128K, 128K,ANY,KEEP)
Amount of storage reserved for the out-of-storage condition and the initial content of storage when allocated and freed	STORAGE(,,,0K)

Of particular interest is the HEAP parameter, which specifies that the initial size of the enclave's heap storage should be 4M and the size of additional increments should be at least 1M.

You can modify these settings to match more closely with the storage use of your JVMs. To improve the use of MVS storage, you should use DFHJVMRO to set the initial allocation for the HEAP runtime option to a value that approximates the amount of storage actually used by your Java applications. Note that the settings that you make using DFHJVMRO apply to all the JVMs that run in your CICS region (with the exception of the master JVM that initializes the shared class cache), so you should consider the different storage heap sizes that JVMs with different JVM profiles might have.

You can use the CICS statistics to see how much LE enclave heap storage is used by your JVMs. The field "Peak Language Environment (LE) heap storage used" in the JVM profile statistics shows the peak amount of LE enclave heap storage that was actually used by a JVM with the specified execution key and profile. See Figure A-21 on page 354.

```
JVMprofiles

Jvmprofile Name. . . . . . . . :  DFHJVMPR
Jvmprofile Class Cache . . . . :  No
JVMprofile Reuse status        :  RESET
Jvmprofile HFS File Name . . . :  /u/ideeley/JVMProfiles/DFHJVMPR
                                                                    CICS     User Total

Total number of requests for this profile. . . . . . . . . . . . . . :     0        1     1
Current number of JVMs for this profile. . . . . . . . . . . . . . . :     0        1     1
Peak number of JVMs for this profile . . . . . . . . . . . . . . . . :     0        1
Number of new JVMs created for this profile. . . . . . . . . . . . . :     0        1     1
Number of times JVMs were unresettable for this profile. . . . . . . :     0        0     0
Number of times this profile stole a TCB . . . . . . . . . . . . . . :     0        0
Number of times this profile was the victim of TCB stealing. . . . . :     0        0
Peak Language Environment (LE) heap storage used . . . . . . . . . . :    0K  16,460K
Peak JVM heap storage used . . . . . . . . . . . . . . . . . . . . . :    0K   2,846K
Number of JVMs destroyed due to Short-on-Storage . . . . . . . . . . :     0        0     0
-Xmx value for this profile. . . . . . . . . . . . . . . . . . . . . :       83886080
```

Figure A-21 DFH0STAT JVM Profiles report

To obtain this information:

1. Use the EXEC CICS INQUIRE JVMPROFILE command to identify each of the JVM profiles in use in your CICS region. (There is no CEMT equivalent for this command). Note that this command will not find a JVM profile which has only been used for the master JVM.
2. Specify the option `LEHEAPSTATS=YES` in each of the JVM profiles that you have identified.
3. Purge your JVMs using the **`CEMT SET JVMPOOL PHASEOUT`** command (or the equivalent **`EXEC CICS`** command) around the time of a statistics reset (either before or immediately afterwards). This ensures that the statistics collected in the next statistics interval are a more accurate reflection of the storage used for your JVMs. It also ensures that your JVMs will be re-created using the `LEHEAPSTATS=YES` option.
4. Run a representative sample of the transactions that use your JVMs. When you have finished, you can use the STAT transaction (which invokes the DFH0STAT sample statistics program) to view the JVM profile statistics that have been collected during the statistics interval.
5. Examine the field “Peak Language Environment (LE) heap storage used” in the JVM Profile statistics for each JVM profile. Use this value to set the initial heap size in the HEAP runtime option in DFHJVMRO. If the peak amount of storage used varies between JVM profiles, select a suitable value based on the relative use of each JVM profile. Try to select a value that is close to the storage used by most of your JVMs, bearing in mind that LE can make

additions to the heap storage, but it cannot remove unnecessary storage that is given in the initial allocation.

Collecting this statistic affects the performance of JVMs, so you should not carry out this process in a production environment.

Unresettable events

If an application is using a resettable JVM, you need to make sure that the Java programs are not performing unresettable actions, because this causes the JVM to be destroyed instead of being reset, which greatly reduces the performance of your application. In the CICS statistics for JVM profiles, the field “Number of times JVMs were unresettable for this profile” shows the number of unresettable JVMs for each JVM profile and execution key. If the statistics show that there were unresettable JVMs for a JVM profile that specifies the option REUSE=RESET, then it is probable that some of the applications requesting that JVM profile are performing unresettable actions. You can identify those applications by looking at byte 6, bit 0 of the transaction flags field, TRANFLAG (field 164 in group DFHTASK), in the CICS performance class monitoring record for each transaction.

To determine the cause of an unresettable event, you need to specify a value for `ibm.jvm.events.output` in the JVM properties file; this value will tell the JVM where to log three kinds of events:

- Unresettable events
- Reset trace events
- Cross heap events

Then you must specify a value of `min` or `max` for `ibm.jvm.unresettable.events.level` in the JVM properties file to enable the logging of unresettable events and to specify the level of logging detail. Figure A-22 on page 356 shows an example of the message written to the `ibm.jvm.events.output` destination in response to the detection of an unresettable event. The event types are documented in the manual *Persistent Reusable Java Virtual Machine User’s Guide*, SC34-6201.

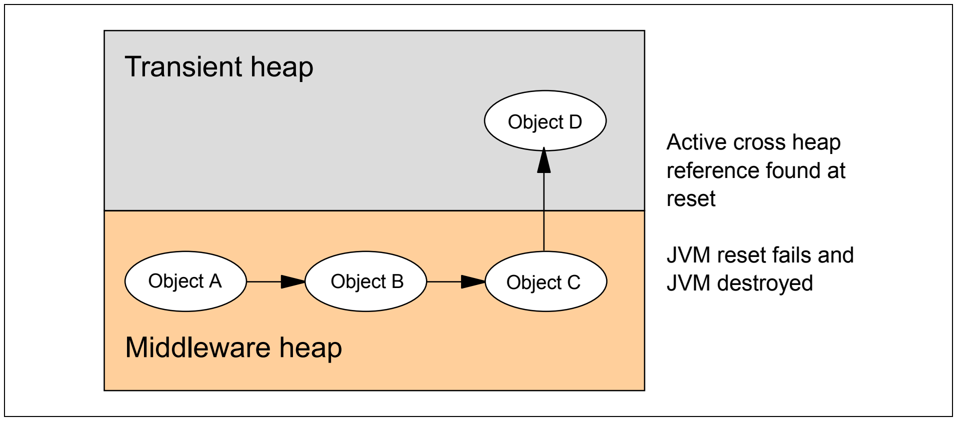
In this example, `0x0000000e` is a SCJVM_REF_FROM_MW_TO_TH condition which is raised because a middleware object (instance or array) contained a current reference to an application object when a reset was requested.

```
[EVENT 0x0000000e]
TIME=15/03/2001 at 11:59:24.456
THREAD=TRGQIIRP.TASK29785.TRGQ (0:17129590) CLASS=UnresettableEvent
DESCRIPTION=0x5172A4F0 is an instance of MickSF/MickeySFBean from address
0x505FAFA0
[END EVENT]
```

Figure A-22 Example of an unresettable event message

Reset trace events

The existence of references from a middleware object to an object in the transient heap causes a reset trace event. When the JVM finds one of these references, it has to determine if the middleware object that contains it is actually dead and hence eligible for garbage collection. This process is expensive in terms of CPU costs. If the middleware object is actually alive, the JVM is marked unresettable. Figure A-23 shows an example of an active cross heap reference.



Transient heap
Object D
Active cross heap reference found at reset
JVM reset fails and JVM destroyed
Object A
Object B
Object C
Middleware heap

Figure A-23 Active cross heap reference

Figure A-24 on page 357 shows an example of an inactive cross heap reference.

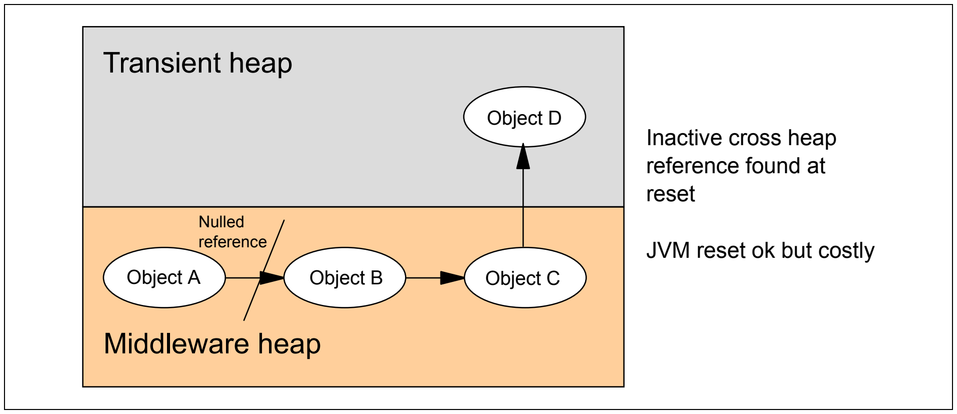


Figure A-24 Inactive cross heap reference

All references from middleware objects to application objects should be set to *null* before the application terminates so that this trace is avoided as shown in Figure A-25.

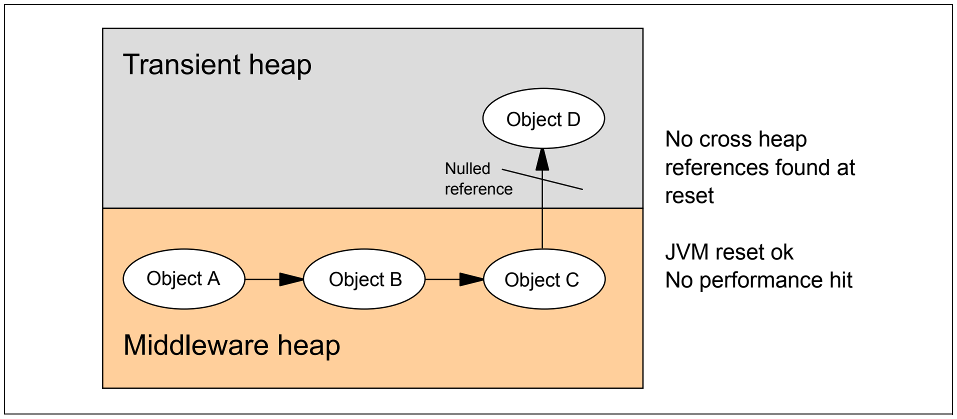


Figure A-25 No heap cross references

Setting `ibm.jvm.resettrace.events=on` will log the presence of these traces at reset time and indicate the object that has the reference.

Setting `ibm.jvm.crossheap.events=on` will only work when the JVM is in debug mode and will trace every time a cross heap reference is actually created. Many of these trace entries will be innocent as they may well have been set to null by the time the reset happens. The trick here is to match up the reset trace event to the rogue cross heap reference event. The cross heap reference entry will have a stack trace to identify the creator.

Figure A-26 shows an example of the message written to the `ibm.jvm.events.ouput` destination in response to the detection of a reset trace event. The event types are documented in *Persistent Reusable Java Virtual Machine User's Guide*, SC34-6201.

In this example, `0x1` is a RESETTRACEEVENT_REFERENCE_TO_TH condition. A middleware object contained a current reference to an application object when a reset was requested and it is necessary to determine whether the middleware object is live or unreferenced.

```
[EVENT 0x1]
TIME=14/03/2001 at 12:03:58.439
THREAD=TRGQIIRP.TASK29685.TRGQ (0:17129580)
CLASS=ResetTraceEvent
DESCRIPTION=0x18C24D70 is an instance of zc01/_callcobolImpl
from address 0x1809Fd80 within obj or array at 0x1809FD78.
0x1809FD78 is an instance of vendor/io/ObjectClass/cache
[END EVENT]
```

Figure A-26 Example of a reset trace event message

System limits

One of the first considerations you will have with regard to running enterprise beans or any other Java programs in CICS is how many JVMs you will need to run. To estimate how many JVMs you need to support a desired level of transaction throughput, use the formula:

```
ETR x Response time = Number of JVMs
```

Where:

- `ETR` is the desired level of transaction throughput
- `Response time` is the time taken to run your transaction in a JVM

By following the tuning processes described in the *CICS TS V2.3 Performance Guide*, SC34-6247, you might be able to reduce the `Number of JVMs` by decreasing the `response time` for your transactions. For example, you should:

- Check that you have chosen the optimum level of reusability for each of your JVMs, considering the design of the applications that use them. This will ensure that you are not wasting CPU time by performing a JVM initialization or reset in situations where these actions are not required for the correct operation of the applications that use the JVM.

- Make sure that you have stopped any unnecessary CPU usage caused by any of the following:
  - The use of tracing
  - The use of the USEROUTPUTCLASS option in JVM profiles (new in CICS TS V2.3)
  - Application programs that take unresettable actions and therefore cause resettable JVMs to be re-initialized instead of reset

When you have calculated the required `Number of JVMs`, then decide if you want to increase or reduce the number of JVMs in your CICS region. You should take the following into consideration:

- The time that transactions are having to wait to acquire a JVM

  Look at the delay time for the JVM pool, shown in the statistics field "Total Max TCB Pool Limit delay time" in the CICS dispatcher TCB pool statistics. This tells you how long your transactions waited to acquire a JVM at those times when the MAXJVMTCBs limit had been reached for the JVM pool. If the delay time seems high, divide it by the statistics field “Total Attaches delayed by Max TCB Pool Limit” in the CICS dispatcher TCB pool statistics, to find the average time a transaction had to wait.
- Your level of QR TCB utilization

  Calls made by a Java program for CICS services, such as using a JCICS class to access VSAM data, require a switch to the QR TCB. Once the QR TCB has reached a high level of utilization, then adding more JVMs (on J8 and J9 TCBs) might produce no further increase in the throughput of your CICS system. You can check your level of QR TCB utilization by looking at the statistics field "Accum CPU Time / TCB" for the QR mode in the CICS dispatcher TCB mode statistics.

If you find that you want to increase the number of JVMs, calculate the maximum number of JVMs that your CICS region can support with the free storage available in its address space. The *CICS TS V2.3 Performance Guide,* SC34-6247, shows you how to do this.

Taking your findings about CPU usage and storage availability into account, you can now choose and set an appropriate MAXJVMTCBS limit for the CICS region. The MAXJVMTCBs system initialization parameter limits the total number of TCBs in the pool of J8- and J9-mode TCBs that CICS uses for JVMs. Each JVM executes on a J8 or J9 TCB, so MAXJVMTCBs limits the number of JVMs that can be active in the CICS region. The JM TCB, which is used by the master JVM that initializes the shared class cache, does not count towards the MAXJVMTCBs limit. You can change MAXJVMTCBs dynamically with the **`CEMT SET`** command.

If you set a MAXJVMTCBs limit that is too high, CICS might attempt to create too many JVMs for the available MVS storage, resulting in an MVS storage constraint. CICS has a storage monitor for MVS storage, which notifies it when MVS storage is constrained or severely constrained, so that it can take short-term action to reduce the number of JVMs in the JVM pool. (The storage monitor uses exits in Language Environment routines; it is not a monitoring transaction.)

If a JVM makes a request for MVS storage and the storage monitor determines that the largest unallocated contiguous block of MVS storage above 16 MB has dropped below the system-defined threshold of 40 MB, CICS issues message DFHSM0137 - `The amount of storage available to MVS is low` - and deletes JVMs not currently in use, thereby freeing any MVS storage the inactive JVMs are holding. Incoming requests for a JVM can reuse an existing free JVM of the correct profile and EXEC key, be assigned a mismatching JVM, or be queued to wait for a JVM to become free. When the MVS storage shortage reported by DFHSM0137 has ceased, CICS issues DFHSM0138.

When there is no unallocated contiguous block of storage large enough to satisfy an MVS request for storage, CICS issues message DFHSM0139 'The amount of storage available to MVS is critically low'. Either CICS has been forced to satisfy the request by releasing storage from an MVS storage cushion of 20 MB that it keeps in reserve or there is at least one task suspended due to insufficient contiguous free storage. In this case CICS terminates all JVMs as soon as they are released by programs and does not start new JVMs until the storage shortage is relieved. When the MVS storage shortage reported by DFHSM0139 has ceased, CICS issues DFHSM0140.

Workload balancing

Figure A-27 on page 361 shows the components of a CICS EJB server, which are also discussed in the following text.

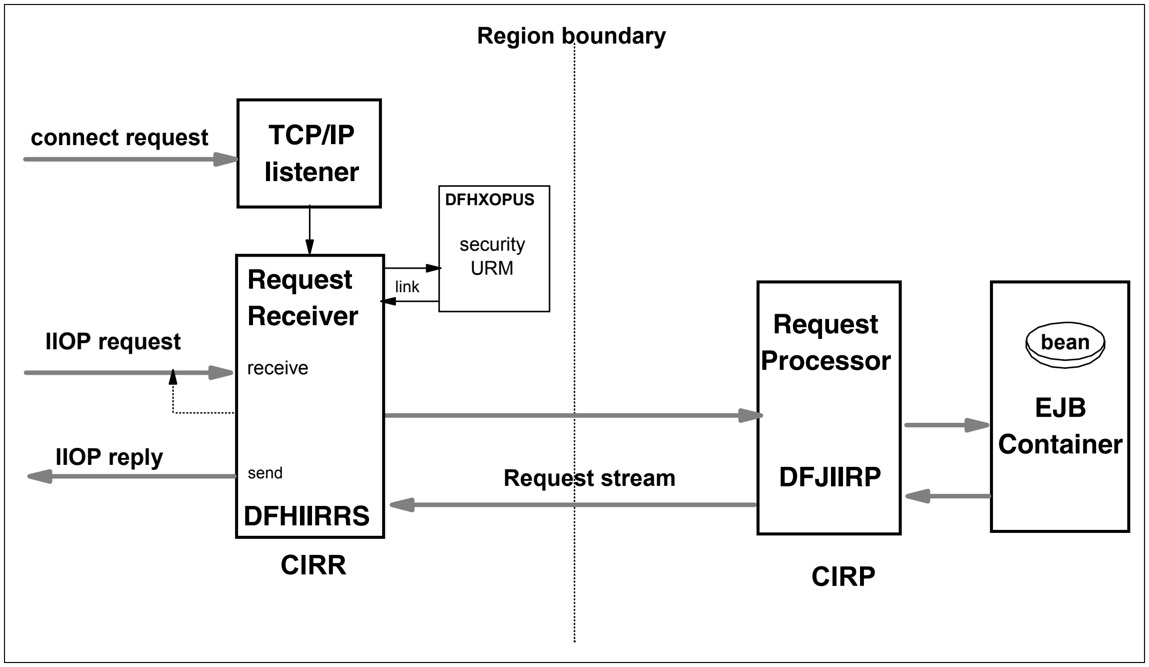


Figure A-27 Components of a CICS EJB server

- TCP/IP listener

  The job of the CICS TCP/IP listener is to listen for (and respond to) incoming TCP/IP connection requests. An IIOP listener is configured by a TCPIPSERVICE resource definition to listen on a specific TCP/IP port and to attach an IIOP request receiver transaction to handle each connection. For IIOP services, the request receiver TRANSACTION definition must have the PROGRAM attribute set to DFHIIRRS, the request receiver program. The default transaction name is CIRR.

  Once an IIOP connection has been established between an client program and a particular request receiver, all subsequent requests from the client program over that connection flow to the same request receiver.

- Request receiver

  The request receiver analyzes the structured IIOP data. It passes the incoming request to a request processor by means of a *request stream,* which is an internal CICS routing mechanism. The object key in the request received, or a transaction service context, determines whether the request must be sent to a new or existing request processor. (A *transaction* in this context means a unit of work defined and managed using the Object Transaction Service - OTS - specification).

If the request must be sent to a new request processor, a CICS TRANSID is determined by comparing the request data with templates defined in REQUESTMODEL resource definitions. (If no matching REQUESTMODEL definition can be found, the default TRANSID, CIRP, is used). The TRANSID defines execution parameters that are used by the request processor.

- Request processor

  The request processor is a transaction that manages the execution of the IIOP request. For an enterprise bean it performs the following functions:

  - Locates the object identified by the request
  - Calls the container to process the bean method

You can implement a CICS EJB server in a single CICS region. However, in a sysplex, it is likely that you will want to create a server consisting of multiple regions. Using multiple regions makes failure of a single region less critical and enables you to use workload balancing to handle higher peak workloads, but still provides a single system image.

A *CICS logical EJB server* consists of one or more CICS regions configured to behave like a single EJB server (Figure A-28) and consists of the following elements:

- A set of cloned *listener* regions which have identical TCPIPSERVICE definitions to listen for incoming requests
- A set of cloned *application-owning regions* (AORS), each of which supports an identical set of enterprise beans in identically-defined CorbaServers

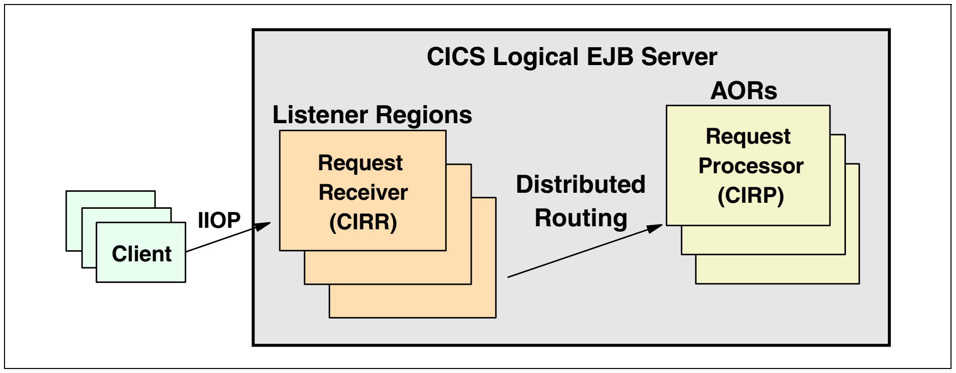


Figure A-28 A CICS logical EJB server

Figure A-28 also shows that you can implement workload balancing at two levels:

- Balancing client connections across the listener regions
- Balancing method requests across the AORs

Balancing client connections across listener regions

You have five options for balancing client connections across the listener regions:

- TCP/IP port sharing
- Network dispatcher
- Domain Name System (DNS) connection optimization
- Sysplex distributor
- Sysplex distributor with MultiNode Load Balancing

Chapter 5 “Setup of the CICS logical EJB Server” in the redbook *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2,* SG24-6284, provides a brief overview of the first four of these options and tells you where you can find more information about them. In addition the redbook *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 5: Availability, Scalability, and Performance*, SG24-6517, provides a detailed discussion of the last three of these options which includes the advantages and disadvantages of each.

Balancing method requests across the AORs

The CICS listener regions use REQUESTMODEL resource definitions to match the incoming method requests to CICS transaction IDs. Therefore, to balance method requests across AORS, you have the following two options:

- Write a customized version of the CICS distributed routing program

  For information about writing a customized distributed routing program, refer to the publication *CICS TS V2.3 Customization Guide*, SC34-6227.

- Use the distributed routing program, EYU9XLOP, provided with CICSPlex SM

  For information about CICSPlex SM Workload Management refer to the manual *CICS TS V2.3 CICSPlex SM Managing Workloads*, SC34-6259.

Clients can begin, commit, and roll back transactions using an implementation of the Java Transaction Service (JTS) or the CORBA Object Transaction Service (OTS). Whether the method of an enterprise bean needs to run under a client’s OTS transaction (if there is one) is determined by the setting of the `trans-attribute` specified in the bean’s deployment descriptor. The possible settings of the `trans-attribute` attribute determines whether the method runs under the client’s OTS transaction, under a separate transaction which is created for the duration of the method, or under no OTS transaction.

CICS invokes the routing program for requests for methods that will run under a *new* OTS transaction, but not for requests for methods that will run under an *existing* OTS transaction, these it directs automatically to the AOR in which the existing OTS transaction runs.

What we mean by new and existing OTS transactions is:

- By a *new* OTS transaction, we mean an OTS transaction *in which the target logical server is not already participating* prior to the current method call; it is *not* necessarily an OTS transaction that was started immediately before the method call.
- By an *existing* OTS transaction, we mean an OTS transaction *in which the target logical server is already participating*, prior to the current method call; not simply an OTS transaction that was started some time ago.

For example, if a client starts an OTS transaction, does some work, and then calls a method on an enterprise bean with the *Supports* transaction attribute, as far as the CICS EJB server is concerned this is a new OTS transaction, because the server has not been called within this transaction's scope before. If the client then makes a second and third method call to the same target before committing its OTS transaction, these second and third calls occur within the scope of the existing OTS transaction.

CICS invokes the distributed routing program for method requests that will run under a new OTS transaction, or outside the scope of an OTS transaction. However, requests that will run under an existing OTS transaction are not dynamically routed; these are directed automatically to the AOR in which the existing OTS transaction is already running.

Enterprise beans in CICS can be used in a number of ways. If they are used to wrap *existing* applications, you will need to consider any particular affinity characteristics of the application. For *new* CICS Java applications you also need to consider transaction affinities, especially if a transaction uses JCICS classes to access resources within a region as this may force subsequent transactions to run in the same region.

Network considerations

In a multi-region EJB server configuration, the request stream, which is an internal CICS mechanism for routing of IIOP requests between request receivers and request processors, is transported using MRO. You therefore need to configure MRO connections between your listener regions (request receivers) and your AORs (request processors). Note that request streams cannot be transported over ISC connections. In a sysplex configuration, you can use MRO/XCF connections between listener regions and AORs running on separate LPARs.

For optimal performance of your CICS EJB applications running in a multi-region EJB server, you need to make sure that the connections between the listener regions and AORs are tuned for the expected IIOP workload.

In the TCPIPSERVICE definition the SOCKETCLOSE parameter specifies if, and for how long, CICS should wait before closing the socket, after issuing a receive for incoming data on that socket. When SOCKETCLOSE is set to `NO`, the socket is left open until data is received, or until it is closed by the client. If SOCKETCLOSE is set to a non-zero period of time, CICS will close the socket after that period of time. Choose a value that is appropriate to the responsiveness of the client, and the reliability of your network. This will allow the majority of requests to use persistent connections, but still allow dormant connections to time out.

The BACKLOG parameter of the TCPIPSERVICE definition specifies the number of TCP/IP connections for this service which are queued in TCP/IP before TCP/IP starts to reject incoming client requests. The number should be less than or equal to the value of the TCP/IP SOMAXCONN parameter. The default value for BACKLOG is one; you may want to set a higher value.

SSL considerations

Authentication and encryption of IIOP requests coming into the listener regions requires the use of SSL. You specify the use of SSL on the TCPIPSERVICE definition which the listener and AOR regions share and the associated CORBASERVER definition which the AORs share.

You can reduce the cost of SSL encryption by using one of the following:

- **Persistent IIOP connections:** Use of a persistent IIOP connection, whereby a subsequent IIOP connection re-uses a previously opened persistent TCP/IP socket connection, ensures that after the initial SSL handshake, no other handshake is performed until the persistent IIOP connection is broken, which will usually only occur when the IIOP connection is timed out by the server.
- **SSL session ID reuse:** When an SSL client and server negotiate a handshake, the SSL client obtains a session ID. If the same client presents this session ID in a new SSL connection (after closing and re-opening the socket, for example) the server can re-establish the connection with a highly optimized handshake. However, the session ID should only remain valid for a limited time (to avoid impersonation of the client). The SSLDELAY parameter specifies what this time limit should be.
- **Cryptographic hardware:** The Cryptographic Coprocessor feature is a hardware feature available on S/390 and System z processors. It consists of dual cryptographic module chips protected by tamper-detection circuitry and a cryptographic battery unit. It can be used to off-load CPU processing from the main CEC processors when performing cryptographic operations, and as such, can provide a significant reduction in CPU usage. The Cryptographic Coprocessor Feature can be used to reduce the CPU costs of SSL data

transmission when using the DES or triple DES ciphers, and SSL handshaking when using the RSA PKCS#1 cipher. In order to use this hardware feature, the z/OS Integrated Cryptographic Service Facility (ICSF) has to be installed and operational.

- **Encryption algorithms:** Different SSL key sizes (1024 or 512 bit) used in the SSL handshake, and the different SSL data encryption algorithms (DES, Triple DES, RC4, and so on) have considerably different performance characteristics.

Application development

A significant benefit of the introduction of support for Enterprise JavaBeans (EJB) into CICS is that it enables a whole new set of software development tools and practices to be used to write software for the CICS environment.

The following sections give an overview of how to design, develop, and deploy an enterprise bean in a CICS EJB server.

Design

CICS provides support for *session beans* that conform with Sun Microsystem Enterprise JavaBeans Specification V1.1, which is described at this URL:

`http://www.javasoft.com/products/ejb`

These session beans can:

- Invoke a session bean in CICS or another EJB server.
- Invoke an entity bean in another EJB server, such as WebSphere Application Server V5.1 and above.
- Call a CICS program, using the *resource adapters for CICS TS via* CCI Interface.
- Use the JCICS classes to access CICS resources, such as linking to a program or accessing a VSAM file.
- Use Java facilities, such as manipulating XML, using the networking support provided by the java.net package, and so on.
- Receive and send over 32 KB of data from or to an EJB client, with the marshalling and unmarshalling of the data handled by the Java infrastructure.

Development

A tool such as Rational Application Developer enables you to easily develop and test Java programs and enterprise beans right inside the Integrated Development Environment (IDE). It includes a single WebSphere Application Server which has full functionality of the WebSphere Application Server and provides server runtime support for testing and debugging enterprise beans. It also includes a feature which will generate a test client to enable you to easily and quickly unit test your enterprise beans without having to write any client code.

Coding a session bean

A session bean is composed of the following:

- Home interface
- Remote interface
- Bean implementation
- Deployment descriptor

The home interface

The home interface specifies the life cycle events such as the creation and removal of a session bean instance.

```
public interface EJBoneHome extends javax.ejb.EJBHome {
                 EJBone create() throws javax.ejb.CreateException,
                                        java.rmi.RemoteException;}
```

The remote interface

The remote interface exposes the methods in the enterprise bean implementation that may be used by a client.

```
public interface EJBone extends javax.ejb.EJBObject {
                 public ResultsBean execute(
                             String funcName,
                             String encoding,
                             String commarea,
                             int commareaLength,
                             String username,
                             String password,
                             boolean managed,
                             String gatewayURL,
                             int gatewayPort,
                             String cicsServer,
                             String mirror,
                             int trace,
                             int iterations,
                             boolean appTrace)
                             throws ResourceException,
                                               Exception,
                                              java.rmi.RemoteException;
}
```

The bean implementation

The bean implementation contains the code that actually carries out both the business logic that will be called by the client and the life cycle methods that will be invoked by the container. By convention, the name given to this class is the name of the remote interface with the suffix *Bean* appended to it as shown in Example A-3.

Example: A-3 Sample EJB implementation

```
import com.ibm.connector2.cics.*;
import com.ibm.ctg.client.T;
import javax.resource.cci.*;
import javax.resource.*;
import javax.naming.InitialContext;
import javax.ejb.*;
import java.io.PrintWriter;

public class EJBoneBean implements SessionBean {
    private SessionContext mySessionCtx;
    transient private Connection eciConn;
    transient private Interaction eciInt;
    transient private ECIInteractionSpec eSpec;
    ...

//This is the main method that does the call to the CICS Server
public ResultsBean execute(
        String funcName,
        String encoding,
        String commarea,
        int commareaLength,
        String username,
        String password,
        boolean managed,
        String gatewayURL,
        int gatewayPort,
        String cicsServer,
        String mirror,
        int trace,
        int iterations,
        boolean appTrace)
        throws ResourceException, Exception {
        JavaStringRecord jsr = null;
          getConnection();
          ...
          //setup the interactionSpec
          eSpec.setFunctionName(funcName);
          eSpec.setCommareaLength(commareaLength);
          // set reply length to same size as commarea
          eSpec.setReplyLength(commareaLength);
          eSpec.setInteractionVerb(ECIInteractionSpec.SYNC_SEND_RECEIVE);
          ...

          //create a record for use
          jsr = new JavaStringRecord(encoding);
          // set input data if we have any
          if (commareaLength > 0){
             jsr.setText(commarea);
          }
          //make the call
          try{
             eciInt.execute(eSpec, jsr, jsr);
          }catch (ResourceException e){
          ...
       }
       ...
    }
    //Get a connection to the EIS using the resource adapter
    private void getConnection() throws Exception{
        ConnectionFactory cf = null;
        try{
            javax.naming.Context ic = new InitialContext();
            cf = (ConnectionFactory) ic.lookup("java:comp/env/ECI");
        }catch (Exception e){
             ...
          }
        try{
            eciConn = (Connection) cf.getConnection();
        }catch (Exception e){
             ...
        }

        try {
            eciInt = (Interaction) eciConn.createInteraction();
        }catch (Exception e){
             ...
        }
    }
          .........
}
```

We purposely omitted the ejbActivate(), ejbPassivate(), ejbRemove(), getNonManagedConnection(), and setters/getters for the Session Bean.

The deployment descriptor

The deployment descriptor provides declarative information to the container about the enterprise bean, such as transactional functionality and security. Typically it is generated by the application development tools (such as Rational Application Developer). A deployment descriptor is an XML document, and an example is shown (Example A-4).

Example: A-4 EJBone EJB deployment descriptor source

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar id="ejb-jar_ID">
   <display-name>EJBoneEJB</display-name>
   <enterprise-beans>
      <session id="EJBone">
         <ejb-name>EJBone</ejb-name>
         <home>itso.cics.eci.j2ee.testercci.EJBoneHome</home>
         <remote>itso.cics.eci.j2ee.testercci.EJBone</remote>
         <ejb-class>itso.cics.eci.j2ee.testercci.EJBoneBean</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
         <resource-ref id="ResourceRef_1">
            <description>CICS ECI Resource adapter</description>
            <res-ref-name>ECI</res-ref-name>
            <res-type>javax.resource.cci.ConnectionFactory</res-type>
            <res-auth>Container</res-auth>
         </resource-ref>
      </session>
   </enterprise-beans>
   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>EJBone</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>*</method-name>
         </method>
         <trans-attribute>Required</trans-attribute>
      </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

To see what the resource reference is bound to, see ibm-ejb-jar-bnd.xml as shown in Example A-5:

Example: A-5 Source of ibm-ejb-jar-bnd.xmi

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<ejbbnd:EJBJarBinding xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:ejbbnd="ejbbnd.xmi" xmlns:ejb="ejb.xmi" xmlns:commonbnd="commonbnd.xmi"
xmlns:common="common.xmi" xmi:id="ejb-jar_ID_Bnd">
  <ejbJar href="META-INF/ejb-jar.xml#ejb-jar_ID"/>
  <ejbBindings xmi:id="EJBone_Bnd" jndiName="ejbs/EJBone">
    <enterpriseBean xmi:type="ejb:Session" href="META-INF/ejb-jar.xml#EJBone"/>
    <resRefBindings xmi:id="ResourceRefBinding_1" jndiName="eis/CICSA">
      <bindingResourceRef href="META-INF/ejb-jar.xml#ResourceRef_1"/>
    </resRefBindings>
  </ejbBindings>
</ejbbnd:EJBJarBinding>
```

Coding a client application

Figure A-29 illustrates the steps a client program needs to perform to invoke the methods in an enterprise bean.

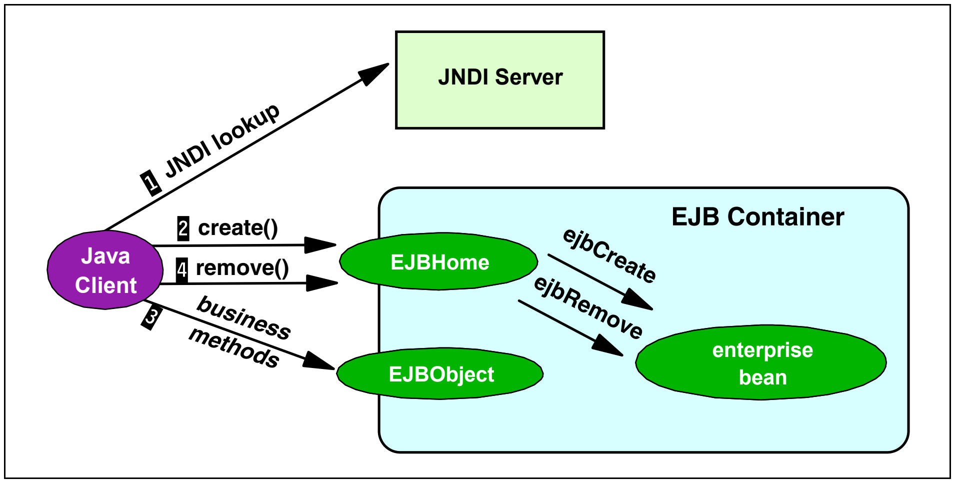


Figure A-29 EJB overview

These steps for accessing EJBs are as follows:

1. Locate the enterprise bean's home interface in a naming service, using the Java Naming and Directory Interface (JNDI).
2. Call the bean's home interface to gain access to the bean's remote interface (as known as EJBObject).
3. Make business method calls against the remote interface.
4. Call the bean's home interface to remove the bean.

There are many ways an EJB can be consumed. These are the most likely clients for a CICS enterprise bean:

- A requester of Web services if we turn the EJB into an Enterprise service.
- Another enterprise bean, running inside another EJB Server such as another CICS EJB Server or WebSphere Application Server.
- A servlet, running in an Web application server, such as WebSphere Application Server.
- A Java application, running in any available platform.

Sample Servlet, JavaBean, Web deployment descriptor, and JSP code is shown shown from Example A-6 to Example A-9 on page 374 as the clients of the EJB.

Example: A-6 A Servlet accessing an EJB

```
public class EJBoneServlet extends HttpServlet {
   ...
   // names of attributes set in the request
   private static final String attrCicsProgram = "funcName";
   private static final String attrCommareaLength = "commareaLength";
   private static final String attrCommarea = "commareaInput";
   private static final String attrEncoding = "encoding";

   ...
   public void init(ServletConfig sc) throws ServletException {
      // call HttpServlet init method
      super.init(sc);
      ...

   }

   ...
    //Common processing routine.
   public void processRequest(HttpServletRequest request, HttpServletResponse 
response)
      throws IOException, ServletException {

         ...
         // retrieve HTTP request parameters
         funcName = request.getParameter("funcName");
         encoding = request.getParameter("encoding");
         commareaInput = request.getParameter("commareaInput");
         commareaLength = request.getParameter("commareaLength");

         ...
         // set common attributes in the request
         request.setAttribute(attrCicsProgram, funcName);
         request.setAttribute(attrEncoding, encoding);
```

```
          request.setAttribute(attrCommarea, commareaInput);

          ...
          // execute the CICS request using the session bean
          try{
             // lookup our session bean
             Context ic = new InitialContext();
           Object or = ic.lookup("java:comp/env/ejb/EJBone");
           EJBone tester = null;
           if (or != null) {
                EJBoneHome home = (EJBoneHome)PortableRemoteObject.narrow(or, 
EJBoneHome.class);
                tester = home.create();
          }

          ResultsBean resultsB = tester.execute(funcName, encoding, 
commareaInput, commareaLengthInt, username, password, managedBool, gatewayURL, 
gatewayDaemonPort, cicsServer, mirror, traceInt, iterationsInt, appTraceBool);
          ...
          tester.remove();
       }
       ...
}
```

Example: A-7 ResultsBean

```
import java.io.Serializable;

//Data Bean which stores a results string and a byte array representing the
//data which created the string. Initially has an empty string and zero length
//byte array.
public class ResultsBean implements Serializable {
   private String resultsString = "";
   private byte[] resultsBytes = new byte[0];

   public void setResultsString(String data){
      resultsString = data;
   }
   public String getResultsString(){
      return resultsString;
   }
   public void setResultsBytes(byte[] data){
      resultsBytes = data;
   }
   public byte[] getResultsBytes(){
      return resultsBytes;
   }
}
```

Example: A-8   Web deployment descriptor

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app id="WebApp">
   <display-name>EJBoneWeb</display-name>
   <servlet>
      <servlet-name>EJBoneServlet</servlet-name>
      <display-name>EJBoneServlet</display-name>

<servlet-class>itso.cics.eci.j2ee.testercci.EJBoneServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>EJBoneServlet</servlet-name>
      <url-pattern>EJBoneServlet</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>
   <ejb-ref id="EjbRef_1">
      <description></description>
      <ejb-ref-name>ejb/EJBone</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <home>itso.cics.eci.j2ee.testercci.EJBoneHome</home>
      <remote>itso.cics.eci.j2ee.testercci.EJBone</remote>
   </ejb-ref>
</web-app>
```

Example: A-9   Results jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>CTGTesterCCI - Results</TITLE>
</HEAD>
<jsp:useBean class="java.lang.String" id="results" scope="request"/>
<jsp:useBean class="java.lang.String" id="commareaInput" scope="request"/>
<jsp:useBean class="java.lang.String" id="encoding" scope="request"/>
<jsp:useBean class="java.lang.String" id="defaultResults" scope="request"/>
<jsp:useBean class="java.lang.String" id="hexResults" scope="request"/>

<BODY>
<%@ include file="include.jsp" %>

<P> </P>
<TABLE BGCOLOR=white CELLPADDING=0><TBODY>
```

```
<TR ALIGN=LEFT><TH>Results </TH><TH> COMMAREA </TH></TR>
<TR><TD>Input:</TD><TD BGCOLOR=yellow><%= commareaInput %></TD></TR>

<TR><TD>Output using <%= encoding %>:</TD><TD BGCOLOR=yellow><%= results
%></TD></TR>

<TR><TD>Output using default encoding: </TD><TD BGCOLOR=yellow><%=
defaultResults %></TD></TR>

<TR><TD>Output in HEX: </TD><TD BGCOLOR=yellow><%= hexResults %></TD></TR>

</TBODY></TABLE>

<TABLE BGCOLOR=white><TBODY>
<TR><TH>Request succeeded.</TH></TR>
</TBODY></TABLE>

</BODY>
</HTML>
```

Deployment

Deployment is the process of taking a generic enterprise bean definition (consisting of home and remote interfaces, a bean class, and an XML deployment descriptor) and turning it into a server specific implementation, and installing it into an EJB server, and making it accessible to clients. CICS supports the use of two WebSphere products to generate the server specific implementation, and provides a tool to help installing enterprise beans in CICS.

Packaging a deployed JAR file

Once you have written your enterprise bean you should have a bean class, and a home and remote interface, along with an XML deployment descriptor. We now need to turn these into a server specific implementation by using a tool which will generate the stubs and ties and other classes needed to run it on the EJB server, and allow the client to access it. This can be done in two ways, as shown in Figure A-30 on page 376.

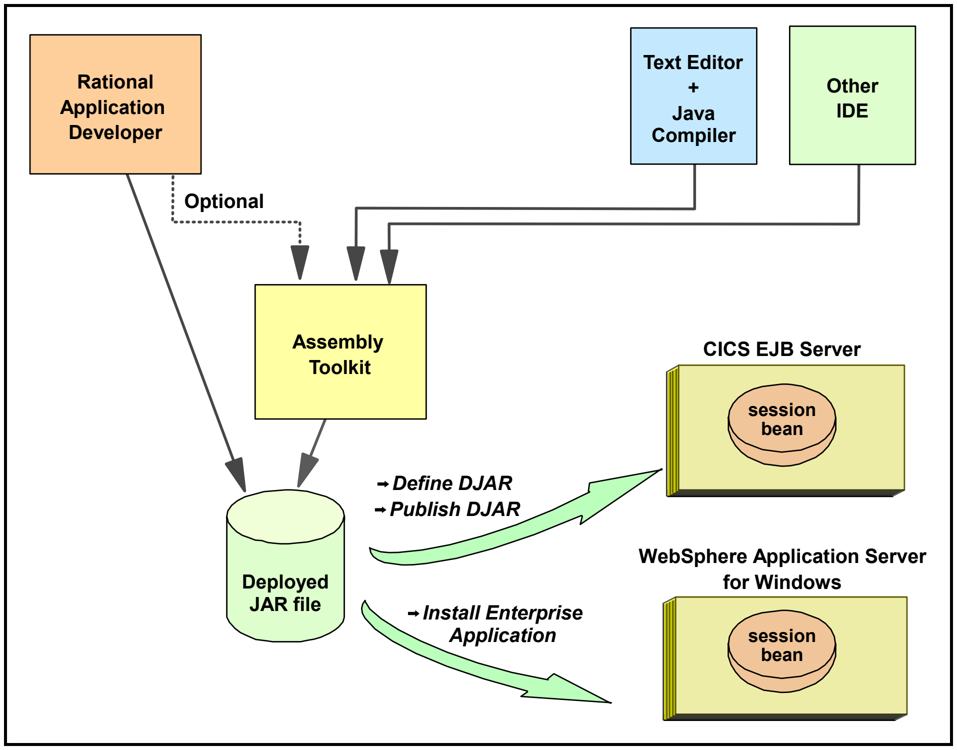
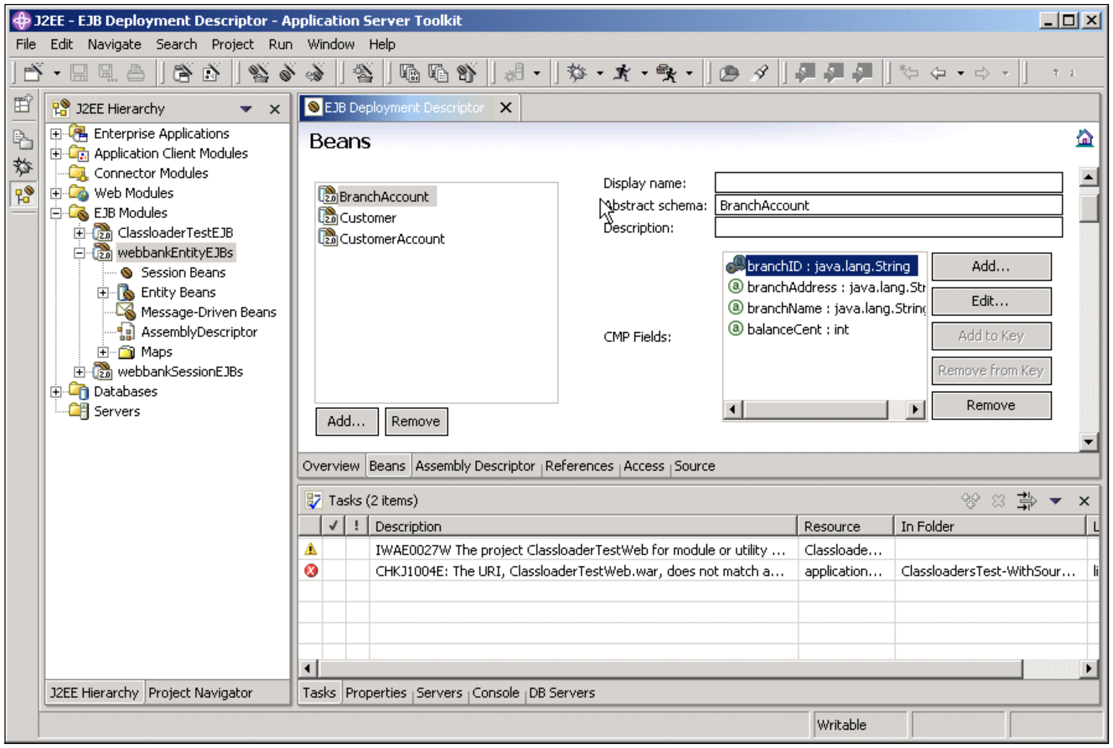

Rational Application Developer
Text Editor + Java Compiler
Other IDE
Optional
Assembly Toolkit
CICS EJB Server
session bean
Define DJAR
Publish DJAR
Deployed JAR file
WebSphere Application Server for Windows
Install Enterprise Application
session bean

Figure A-30 Creating a deployed JAR file for CICS

If you developed your bean in Rational Application Developer, you can use that tool to generate the deployed code. You can then export an EJB JAR from Rational Application Developer, and your bean will be deployed straight into CICS.

If you are using another IDE, WebSphere Application Server (with the exception of the Express configuration) provides the Assembly Toolkit in the Application Server Toolkit for packaging applications. The Assembly Toolkit is shown in Figure A-31 on page 377. For more information about how to use the Assembly Toolkit, refer to *IBM WebSphere Application Server V5.1 System Management and Configuration WebSphere Handbook Series*, SG24-6195.



Display name:

Abstract schema: BranchAccount

Description:

CMP Fields:

branchID : java.lang.String
branchAddress : java.lang.Str
branchName : java.lang.Strin
balanceCent : int

Tasks (2 items)

	✓	!	Description	Resource	In Folder	L
⚠			IWAE0027W The project ClassloaderTestWeb for module or utility ...	Classloade...		
⊗			CHKJ1004E: The URI, ClassloaderTestWeb.war, does not match a...	application...	ClassloadersTest-WithSour...	li

Figure A-31 Application Server Toolkit J2EE perspective

Deploying to CICS

Once you have generated the deployed JAR file, it must be installed into a CICS EJB server. This step involves the creation of CICS resource definitions, publishing bean references to an external namespace, and making the EJB-JAR file accessible to CICS. The following two deployment options are possible:

- Manual deployment

  This method involves manually creating and installing the required definitions, and then publishing your DJARs manually. This method is recommended for system programmers, and people who want to have more control over what is installed when and where.

- Deployment using the Resource Manager for Enterprise Beans (RMEB)

  The RMEB along with some new attributes on the CORBASERVER definition mean that it is very simple to install and update enterprise beans in a CICS system with very little knowledge of CICS. This method is recommended for systems where application developers will be writing and testing applications using enterprise beans.

Manual deployment

Manually performing the enterprise bean deployment allows you to test the technology and helps you understand the whole process. The steps in order to manually deploy an enterprise bean in CICS are as follows:

1. Create the CICS resource definition, namely a CORBASERVER, DJAR, REQUESTMODEL (optional) and TCPIPSERVICE.
2. Store the deployed JAR file in a HFS directory accessible to CICS.
3. Install the CICS resource definitions.
4. Publish the reference to the enterprise bean's home interface, also called the *interoperable object reference* (IOR) to the COS Naming Server, using the CEMT PERFORM DJAR() PUBLISH command.

For application developers who are not familiar with CICS, this is not a very easy option, and could slow down development. Therefore the following option is provided.

Deployment using the Resource Manager for Enterprise Beans

The RMEB is designed to speed up the development of EJB applications that run in CICS. It provides a simple Web based interface for the developers who can use it to install enterprise beans. If they want to make a change to the code, all they have to do is copy the new JAR to HFS, and click the **Scan** button. The enterprise beans will then be re-installed and re-published. Using this tool the developer does not need any knowledge of RDO or CICS commands. Using this tool requires that some initial setup has been done by the system programmer, such as correctly configuring a CORBASERVER (specifying a pickup directory and setting Autopublish to `Yes`). They must also have done the necessary tasks to setup RMEB (which uses the HTML interface provided by CICS Web support).

Data conversion

CICS TS V2.1 introduced support for enterprise beans running in a Java Virtual Machine (JVM) within CICS.

All `String` objects or `char` types are internally represented in Java as Unicode. However, when the Java program needs to interface with CICS resources (such as passing COMMAREAs to a COBOL program), all information must be in Java byte arrays, which do not use Unicode.

Data types

The z/OS JVM, by default, uses its native EBCDIC code page when converting strings or chars into byte-arrays. Figure A-32 shows the data flow between an EJB client and an enterprise bean in CICS.

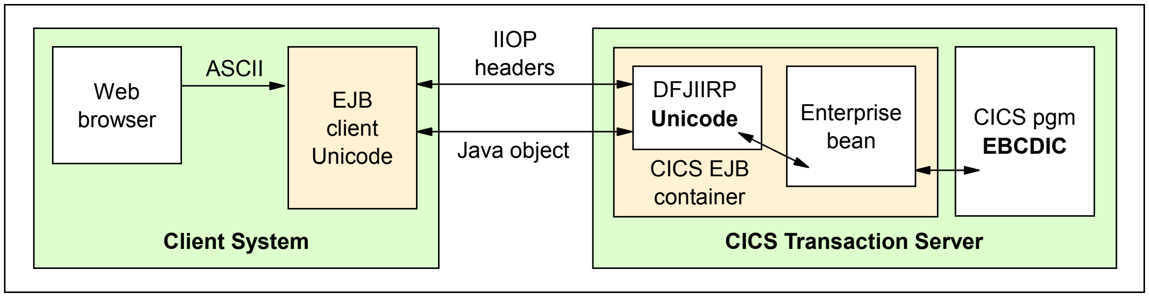


Figure A-32 EJB character set flow

In this flow of data there are basically five data types in use at different locations:

- The Web client application usually processes data in the Web browser's ASCII code page.
- The EJB client will use the ISO 8859-1 code page to build the IIOP header information. This is transparent to the Java programmer.
- The application data flowing between the EJB client and the EJB container will be Java objects, which are transported in a Unicode stream, unless the client marshals character data into a byte array. If this is the case, then you need to know the code page used by the sender to create the data.
- The CICS container will take care of proper code page conversion to invoke the enterprise bean. In fact, the request processor (DFJIIRP) program is a Java program and will interface with the user enterprise bean in Unicode.
- The server platform's native code page; for z/OS this is an EBCDIC code page, and conversion will be required if data needs to be passed from String

objects (Unicode) to the byte arrays, which are required for all access to CICS resources.

Accessing CICS resources

An enterprise bean running inside CICS can be designed to access CICS resources using two different sets of classes:

- JCICS classes
- CCI connector, supported by the new *CICS connector for CICS TS*

JCICS classes

Support for Java programs was first introduced by CICS TS V1.3 using the *JCICS* classes. The JCICS classes, as provided in dfjcics.jar, is the Java equivalent of the EXEC CICS application programming interface (API).

A CICS Java program or CICS enterprise bean can use the JCICS classes to link to an existing CICS program or to perform a variety of CICS operations, such as wiring to VSAM files or temporary storage queues.

Using JCICS

Figure A-33 shows an example of a COBOL program writing a record in a VSAM file, using the traditional CICS API.

```
MOVE "00010" TO REC-KEY.
MOVE "Writing a record to VSAM" TO REC-TEXT.

EXEC CICS WRITE FILE("MYFILE")
          LENGTH(LENGTH OF RECORD-DATA)
          RIDFLD(REC-KEY)  RESP(RESP-FLD)
          FROM(RECORD-DATA) END-EXEC.
IF (RESP-FLD NOT = DFHRESP(NORMAL))
* ---> Error checking...
END-IF.
```

Figure A-33 Writing a VSAM record using COBOL

The same access is shown in Figure A-34 on page 381, using the JCICS classes.

```
String sKey = "00010";
String sText = "Writing a record to VSAM      ";
String sRec=sKey+sText;
KSDS myFile = new KSDS();
myFile.setName("MYFILE");
try {
     myFile.write(sKey.getBytes(),sRec.getBytes());
     System.out.println("Record write successful");}
catch (Throwable t) { // Error handling...}
```

Figure A-34   Writing a VSAM record using JCICS

Note that programs using the JCICS API do not implicitly call the CICS conversion program DFHCCNV, so all conversion from Unicode into EBCDIC byte arrays must be done inside the Java code, before and after invoking CICS services. Since JCICS programs can only run in the EBCDIC CICS environment, we do not have to be concerned about portability of code.

The method `getBytes()` will convert a Unicode string of character data into a byte array, using the default code page (EBCDIC) for the CICS JVM environment. But what if our data is composed of other data types such as 16 or 32-bit integers or packed decimal fields? In this case you will need to develop more complex data marshalling logic either within your Java application or within CICS. For further details on converting numeric data within Java refer to the IBM Redbook *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401.

CCI Connector for CICS TS

CICS TS V2.3 introduced the *CCI Connector for CICS TS*. This enables a Java program running within CICS to link to an existing program using the facilities of Common Client Interface (CCI). The CCI is defined by the JCA and is the same interface as provided by the CICS ECI resource adapter from the CICS Transaction Gateway. This means that enterprise beans that invoke existing CICS applications can be made portable, since the CCI can be used by any Java application, including servlets and other enterprise beans running in a different Enterprise JavaServer™.

The function is equivalent to that provided by the JCICS `link()` method of the `Program` object, but does not tie the Java program to running in a CICS environment.

Data conversion

When *using the CCI Connector for CICS TS*, you can handle data conversion using one of two methods:

1. Using the services of the CICS conversion program, DFHCCNV. First you have to convert the data from Unicode to ASCII. Then you must provide a conversion template in the DFHCNV conversion table. The connector invokes DFHCCNV before and after the program link call.
2. Using your own data conversion routines either within CICS or within your Java application.

More information concerning usage of the CCI Connector for CICS TS, restrictions and recommendations is available in the *CICS Information Center*.

B

SOAP for CICS feature

This appendix covers the SOAP for CICS feature available with CICS TS V2.2 and V2.3.

Note: The SOAP for CICS feature has been replaced in CICS TS V3.1 with the Web services support feature described throughout this redbook. We strongly recommend using the Web services support in CICS TS V3.1 in place of the SOAP for CICS feature.

This appendix contains the following sections:

- Introduction
- Security
- Transactional scope
- Performance and scalability
- Application development
- Data conversion

Introduction

The SOAP for CICS feature allows CICS to operate as a service requester or service provider within a Web services architecture.

Using the SOAP for CICS feature CICS business logic programs can interact securely and reliably with other Web services, exchanging XML based messages, independently of platform, or application language. Within the Service Orientated Architecture (SOA), standard definitions are used to describe the message formats than can be sent. Thus developers can rapidly build open standards based applications independently of the CICS business logic program they will interact with. This is the essence of the service orientated architecture, and the reason that is deemed to be loosely coupled.

The SOAP for CICS functionality was originally introduced as supportpac CA1M which has now been replaced by an optional feature for CICS Transaction Server V2.2 or V2.3.

Note: In CICS TS V3.1 the SOAP for CICS feature has been replaced by Web services support.

The service provider and service requester functions within CICS are run as *pipelines* utilizing CICS Business Transaction Services (BTS) functionality. User application code running as part of the pipeline needs to be aware that it will be within a BTS activity so that it correctly handles events and containers.

SOAP messages are sent as XML documents, for these messages to be understood by an existing CICS application, they must be parsed, and so user application code will need to be added to the pipeline to parse the XML. Enterprise COBOL V3 and Enterprise PL/I V3 both provide built-in XML parsing functionality for the PL/I and COBOL languages, and this can simplify the development of parsing code. Alternatively, parsing code can be generated automatically using a tool such as WebSphere Studio Enterprise Developer.

The rest of this section describes the supported transports and the inbound and outbound pipeline structure.

Supported transports

The SOAP for CICS feature supports two transports, HTTP and WebSphere MQ.

HTTP

The HTTP transport is supplied by CICS Web support. The standard format URL, if using the default analyzer program DFHWBADX, for use with SOAP for CICS is `/cics/cwba/dfhwsdsh/prog` where *dfhwsdsh* is the HTTP dispatcher program that will start the service provider pipeline and *prog* is the target user written message adapter within the service provider pipeline.

Note: You can use a different format URL (with a custom analyzer program) but you should ensure that the message adapter program name appears after the final slash in the URL. If it does not then the application mapper (DFHWSAMX) within the service provider pipeline will have to be used to specify the program name.

WebSphere MQ

In addition to HTTP you can use WebSphere MQ as the transport for the SOAP for CICS feature. The service provider pipeline will be started by the MQ trigger monitor task (CKTI) starting a transaction in response to receiving a new message. By default the transaction is CWSQ and it will run program DFHWSDSQ which is the MQ dispatcher program. The TRIGDATA parameter on the queue definition specifies the target message adapter program. Figure B-1 illustrates the flow of a message between MQ and CICS and shows the queues and tasks involved.

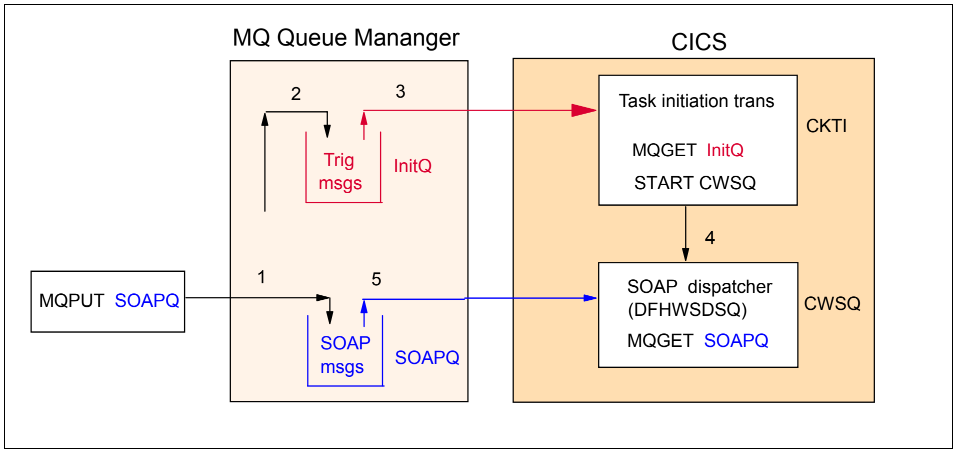


Figure B-1 Message flow between WebSphere MQ and CICS

Inbound pipeline: service provider

The inbound pipeline is shown in Figure B-2 and identifies the three different types of program involved.

CICS supplied	CICS code that is not changeable by the user.
User replaceable	CICS supplies a default version that can be modified.
User supplied	User written application code.

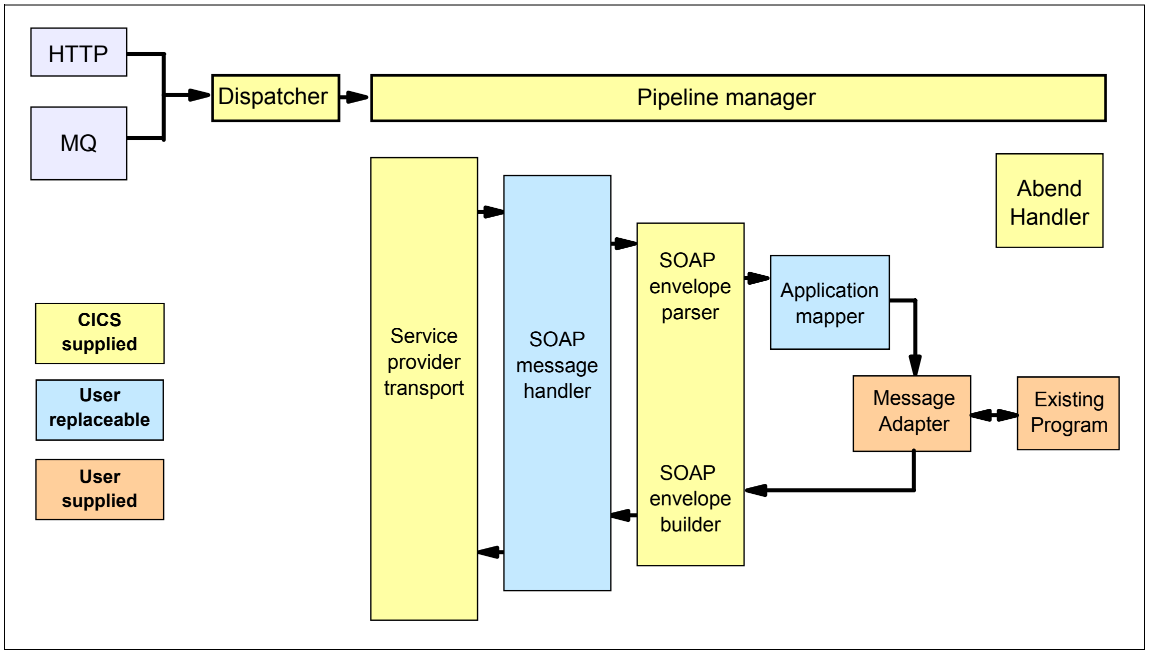


Figure B-2 SOAP inbound pipeline

Dispatcher

The main role of the dispatcher program is to create the BTS process that the pipeline will run in and then link to the pipeline manager activity. There are two different dispatcher programs supplied by CICS. DFHWSDSH is for HTTP and DFHWSDSQ is for MQ. In addition to starting the pipeline, DFHWSDSQ also receives the message from WebSphere MQ.

Pipeline manager

The inbound pipeline manager is a CICS-supplied module called DFHWSPMI. The pipeline manager ensures that each stage of the pipeline is executed in turn. It also manages the movement of data between the various stages of the pipeline. That data includes the SOAP request received from the client, the SOAP response created by the application, and any user application specific data. The pipeline manager establishes an abend handler so that any errors are processed correctly and a suitable error response is returned to the client.

Service provider transport

The service provider transport program is provided by CICS. There is one for each supported transport. DFHWSTIH is for HTTP and DFHWSTIQ is for MQ.

The service provider transport program handles the sending and receiving of the SOAP message using the appropriate protocol (HTTP or MQ). It is called twice during the pipeline. The first time it is called, it has to receive the SOAP request and make it available to the next stage of the pipeline. The second time it is called, it takes the completed SOAP response and sends it back to the client.

SOAP message handler

The SOAP message handler is a user replaceable program called DFHWSHDX. If it is changed it must still be called DFHWSHDX which means you can only have one per CICS region.

The SOAP message handler is called twice in the inbound pipeline: immediately after the service provider transport program has received the SOAP request and immediately before the service provider transport program sends the SOAP response. The SOAP message handler is given as input the entire SOAP message and has to supply as output a complete SOAP message. This is the only user replaceable program in the pipeline that gets to see the entire SOAP message.

The SOAP message handler can be used to directly manipulate and change the SOAP message. A possible reason to change the message is to add information into the SOAP envelope header section of the SOAP response. The SOAP message handler can also parse part or all of the message and extract information for use in later stages of the pipeline. This could be useful if there is information in the SOAP envelope header needed later, for example, some form of authentication such as a user ID and password.

The SOAP message handler could also be a good point in the pipeline to perform logging or auditing functions.

CICS supplies a default version of DFHWSHDX which does not perform any processing of the SOAP message.

SOAP envelope parser

The SOAP envelope parser is a CICS-supplied module called DFHWSSH. It parses the SOAP request using an XML parser. The purpose of parsing the SOAP request is to extract the body of the request, which is the data required by the message adapter, and to extract the XML namespace definitions.

The SOAP body and the namespaces are then made available to the message adapter. The namespaces are required to allow the SOAP body to be correctly parsed by the message adapter.

Application mapper

The application mapper is called DFHWSAMX. It is a user replaceable module responsible for determining the security context for the message adapter and target business logic application. Even if the program is modified, it must still be called DFHWSAMX. This means that you can only have one version per CICS region.

The application mapper can set the user ID, transaction ID, and program name for the activity used to run the message adapter program. It is given as input the values that the pipeline manager has determined will be used and can accept them or change any or all of them. The CICS-supplied version of DFHWSAMX accepts the values chosen by the pipeline manager.

Message adapter

This is a user-written program that needs to:

1. Convert the body of the SOAP request that was received into a correctly formatted COMMAREA structure for input to the existing program
2. Link to the existing program to perform the required business logic
3. Convert the COMMAREA returned by the business logic into the body of a SOAP response.

The program needs to understand XML and will need some means of parsing the XML it has been passed. The program can be written by hand or can be generated using appropriate tooling such as WebSphere Studio Enterprise Developer.

Existing program

The existing COMMAREA based program. This implements the required business logic and will be linked to from the message adapter program.

SOAP envelope builder

The SOAP envelope builder is DFHWSSH which was also the SOAP envelope parser. When called to build the SOAP envelope it will wrapper the XML produced by the message adapter with the required tags (Envelope and Body) for the SOAP envelope. It will also add any application-supplied namespaces to the SOAP envelope tag.

Abend handler

The pipeline manager has established DFHWSABE as the HANDLE ABEND program. If an abend should occur at any stage in the pipeline then DFHWSABE will get control to handle it assuming that a lower level user written abend handler has not already handled it. DFHWSABE will create a SOAP fault message to be returned to the service requester. The message is sent and any outstanding activities within the pipeline are cancelled. Finally, the pipeline is abended if using MQ as the transport or a SYNCPOINT ROLLBACK is issued if using HTTP. Example B-1 shows the SOAP fault that gets returned to the service requester. *abcode* would be replaced by the actual abend code and *abprog* would be replaced with the name of the program that abended.

Example: B-1 Format of the SOAP fault produced by DFHWSABE

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <env:Fault>
      <faultcode>env:Server</faultcode>
      <faultstring>The Server suffered an Internal Error</faultstring>
      <detail>
        <abend xmlns="http://cts.software.ibm.com/cicsts/soap">
          <abcode>abcode</abcode>
          <program>abprog</program>
        </abend>
      </detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Outbound pipeline

The outbound pipeline is shown in Figure B-3 on page 390 and identifies the three different types of program involved.

CICS supplied	CICS code that is not changeable by the user.
User replaceable	CICS supplies a default version that can be modified.
User supplied	User written application code.

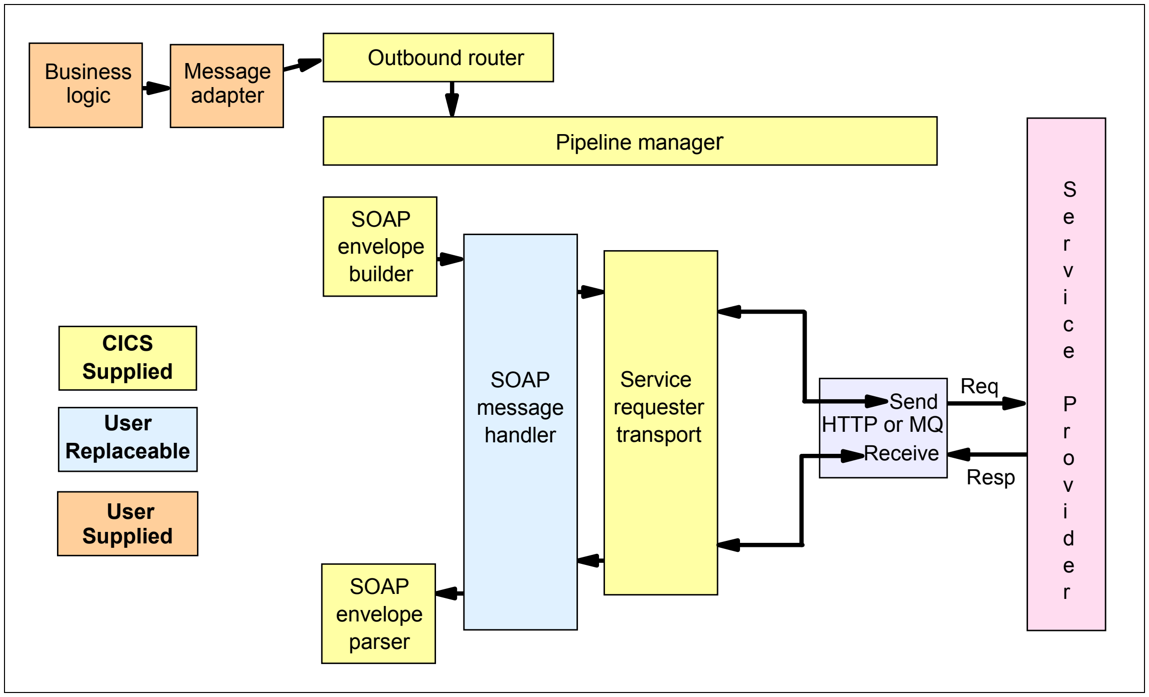


Figure B-3 SOAP outbound pipeline

Business logic

This user written program performs the necessary business logic before and after linking to the message adapter. This is unlikely to be an existing application as it would need to be performing a client function rather than a server function.

Message adapter

This user written program will be linked to from a business logic program. It has to perform several functions.

- Convert the COMMAREA into the body of a SOAP request.
- Specify the URI of the remote service provider.
- Invoke the outbound SOAP pipeline using the outbound router.
- Convert the response from the outbound router into a COMMAREA and return to the business logic program.

Outbound router

The outbound router is a CICS-supplied module called DFHWSRT. It has to determine which transport protocol is being used and start the pipeline manager to use that protocol. It is also responsible for passing the data from the message adapter to the pipeline and for returning the response from the pipeline manager back to the message adapter.

Pipeline manager

The outbound pipeline manager is a CICS supplied module called DFHWSPMO. The pipeline manager ensures that each stage of the pipeline is executed in turn. It also manages the movement of data between the various stages of the pipeline. That data includes the SOAP request received from the client, the SOAP response created by the application, and any user application specific data.

SOAP envelope builder

The SOAP envelope builder and the SOAP envelope parser in the outbound pipeline is the same program that was used in the inbound pipeline which is DFHWSSH. See "SOAP envelope parser" on page 388 and "SOAP envelope builder" on page 389. DFHWSSH performs exactly the same functionality in the outbound pipeline as it did in the inbound pipeline except the order of the calls is reversed. In the outbound pipeline, the first call is to build the SOAP envelope.

SOAP message handler

The SOAP message handler is DFHWSHDX. This is the same program that was used in the inbound pipeline and details can be found in "SOAP message handler" on page 387. It is still called twice but in the outbound pipeline, it is called before the service requester transport program sends the request and after the service requester program receives the reply.

Service requester transport

The service requester transport program is supplied by CICS. There is one program for each supported transport. DFHWSTOH is for HTTP and DFHWSTOQ for MQ. The service requester transport is responsible for sending the SOAP request to the remote service provider, receiving the SOAP response and making it available to the rest of the pipeline.

Security

In this section, we look at the security issues that you will need to consider when implementing a SOAP service provider or service requester using the SOAP for CICS feature.

Service provider

There are two separate areas to consider when implementing a SOAP service provider.

- Security for the transport mechanism used, which is either HTTP (using CICS Web support) or WebSphere MQ.
- Security within the SOAP inbound pipeline.

Transport security

The two supported transports will be discussed separately.

HTTP

The HTTP transport is supplied by CICS Web support. This means that all the security features available with CICS Web support are automatically available for use with the SOAP for CICS feature.

For example, to use SSL to encrypt the SOAP request you would define a TCPIPSERVICE with the SSL parameter set to either YES. You would also optionally provide the name of a server certificate to be used. If you are using the well-known port numbers, then remember to specify the value of the PORTNUMBER parameter as 443 for SSL.

If you require a means to authenticate the client then you can use either of the following:

- Basic authentication

  This will request a user ID and password.
- SSL client authentication

  To use this, define a TCPIPSERVICE with the SSL parameter set to `CLIENTAUTH`. This will request that the client provide a certificate which can then be authenticated by RACF and mapped to a user ID.

WebSphere MQ

When using WebSphere MQ as the transport, you will need to consider the following points.

The transport transaction (CWSQ) will be started by the trigger monitor using the same user ID as the trigger monitor. This user ID must have UPDATE authority to the input queue, the backout queue (if this is specified), and the BTS repository.

If `AUTH=IDENTIFY` is specified in the USERDATA parameter of the WebSphere MQ PROCESS definition for CWSQ, then the user ID under which CWSQ runs must have surrogate authority to allow it to start transactions on behalf of the user IDs in the MQ message descriptors (MQMDs) of the messages.

The process user IDs must have UPDATE authority to the reply queue, and to the BTS repository.

More information about security for WebSphere MQ can be found in *WebSphere MQ Security*, SC34-6079, and in the redbook *WebSphere MQ Security in an Enterprise Environment*, SG24-6814.

Pipeline security

The main security concern within the SOAP pipeline is in securing access to the target service provider application. The pipeline runs as a BTS process and is managed by the inbound pipeline manager program which is DFHWSPMI. DFHWSPMI runs each stage of the pipeline as a separate activity using EXEC CICS LINK ACTIVITY. The message adapter and business logic application may be executed using **`EXEC CICS RUN ACTIVITY SYNCHRONOUS`** which allows them to use a different user ID or transaction ID.

The inbound pipeline provides a user replaceable module, known as the *application mapper*, which is able to set the security context (program name, transaction ID and the user ID) for the requested service provider application. The module is called DFHWSAMX and a default version is supplied.

The position of the application mapper in the inbound pipeline is shown in Figure B-4 on page 394.

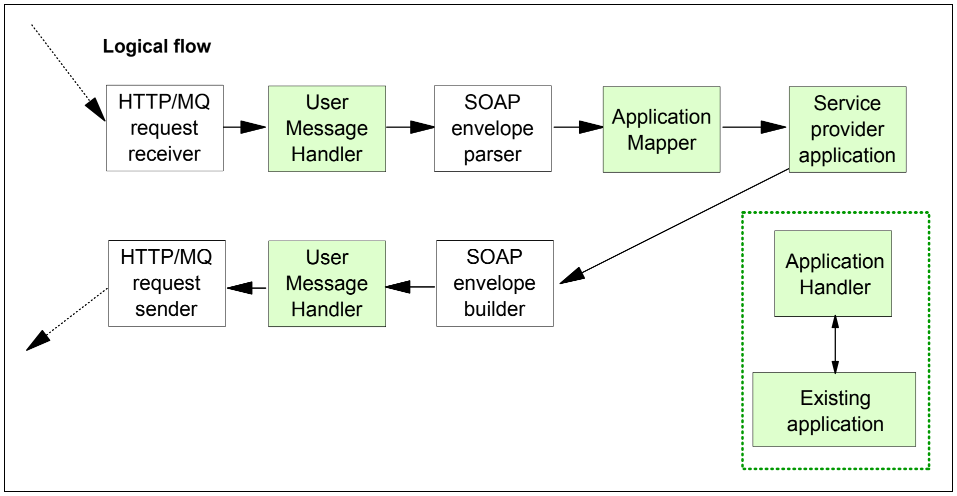


Figure B-4 SOAP inbound pipeline

To allow DFHWSAMX to determine the security context, CICS supplies it with several pieces of information in a set of input containers. These containers are:

- TARGET-TRANSID: contains the transaction ID CSAC.
- TARGET-USERID: contains the user ID that the pipeline is currently running under. This is either the user ID under which the CWS alias transaction is running or the user ID under which CWSQ is running.
- APP-HANDLER: contains the name of the service provider application as determined by the pipeline manager.
- TARGET-URI: contains the URI used by the client to request this service.
- USER-CONTAINERS: optional list of user-defined containers.

DFHWSAMX may need to use part of the SOAP request to determine the user ID, transaction ID, and program name. The only way to achieve this is to use the user message handler program (DFHWSHDX) to save any relevant part of the SOAP request into a user defined container. The name of that container will be passed to DFHWSAMX in the USER-CONTAINERS container. The container can then be accessed using an **`EXEC CICS GET CONTAINER`** call. DFHWSAMX can set the transaction ID, user ID, and program name by using **`EXEC CICS PUT CONTAINER`** calls specifying the container and the required value.

Depending on the transport security used you may or may not need to change the user ID.

If the URI used follows the default format for the SOAP for CICS feature then program name will not need to be changed. If the URI is generic with the name of the service provider supplied within the SOAP headers then the user message handler program will need to have extracted this information earlier in the pipeline and saved it in a user defined container.

Tip: If you use the application mapper to specify the user ID then you should also specify the transaction ID. If you specify just the user ID then the transaction ID will default to CSAC.

Service requester

The service requester runs as a standard CICS transaction so normal CICS resource security will be in place. There may be some security constraints placed upon you by the remote service provider.

When using HTTP as the transport, you may be required to use SSL (with or without a client certificate). This is achieved by specifying `HTTPS://` instead of `HTTP://` at the beginning of the URI in the TARGET-URI container. If the remote server requests a client certificate, then CICS will send the default certificate from the keyring. This means that the remote server can identify the CICS region that sent the request but it cannot determine the individual user who sent it. Another form of identification would have to be used in combination with SSL and a client certificate.

Web Services Security (WS-Security)

WS-Security V1.0 was announced by IBM, Microsoft, and Verisign on April 11, 2002. It presents a strategy for addressing security issues within a Web services environment.

WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

WS-Security also provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required by WS-Security. It is designed to be extensible (for example, support multiple security token formats). For example, a client might provide proof of identity and proof that they have a particular business certification.

Additionally, WS-Security describes how to encode binary security tokens. Specifically, the specification describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys. It also includes extensibility mechanisms that can be used to further describe the characteristics of the credentials that are included with a message.

Composable architecture

By using the SOAP extensibility model, SOAP-based specifications are designed to be composed with each other to provide a rich messaging environment. By itself, WS-Security does not ensure security nor does it provide a complete security solution. WS-Security is a building block that is used in conjunction with other Web service and application-specific protocols to accommodate a wide variety of security models and encryption technologies. Implementing WS-Security does not mean that an application cannot be attacked or that the security cannot be compromised.

Further details on WS-Security can be found at:

http://www-106.ibm.com/developerworks/Webservices/library/ws-secure/

WS-Security and the SOAP pipeline

The SOAP for CICS feature does not provide any built-in support for WS-Security. However, the user message handler, DFHWSHDX, has access to the entire SOAP request so it can be modified to implement some of the WS-Security techniques. For example, ID assertion (where the service requester is trusted and allowed to specify only the user ID for the service provider) can be implemented. Typically, the user ID would be sent in the SOAP headers as shown in Example B-2.

Example: B-2 Passing user ID in SOAP headers

```
<S:Envelope xmlns:S=”http://www.w3.org/2001/12/soap-envelope"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
    <S:Header>
            ...
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Zoe</wsse:Username>
            </wsse:UsernameToken>
        </wsse:Security>
            ...
    </S:Header>
    ...
</S:Envelope>
```

DFHWSHDX could parse the headers and extract the user ID from the <wsse:Username> tag. This user ID could then be placed in a container using EXEC CICS PUT CONTAINER and the name of that container added to the list held within the USER-CONTAINERS container. This user ID is now available to be used by the application mapper (DFHWSAMX) to set the security context of the service provider application.

Transactional scope

In this section we will look at the current transactional capabilities in the SOAP for CICS feature and then provide a brief introduction to the Web services transaction standard, *WS-Transaction*.

SOAP for CICS transactional support

The SOAP for CICS feature has no support for participating in global transactions. The service provider and service requester pipelines run as separate CICS-transactions in their own units-of-work.

The pipelines run as activities within a BTS process. The original design of BTS meant that user syncpoints were not allowed. APAR PQ76073 for CICS TS V2.2 made enhancements to BTS processing to allow user syncpoints within an activity, but this support is only available with the SOAP for CICS feature if HTTP is used as the transport. If WebSphere MQ is used as the transport then user syncpoints are still not allowed.

Web services transaction standard (WS-Transaction)

Web services increasingly tie together a large number of participants forming large distributed applications. The resulting activities can be complex in structure, with complex relationships between their participants. It requires a flexible and extensible mechanism for controlling requests and outcomes in addition to the behavior offered by traditional distributed and database transaction models. So in August 2002, IBM, BEA, and Microsoft released WS-Transaction along with Business Process Execution Language for Web Services (BPEL4WS), and WS-Coordination specifications for transactional integrity, reliably choreographing Web services-based applications, providing business process management, and generic coordination facilities.

The WS-Transaction specification provides the definition of two coordination types:

- Atomic transaction (AT)

  AT is used to coordinate activities having a short duration and executing within limited trust domains. It has an *all or nothing* property.

- Business activity (BA)

  BA is used to coordinate activities that are long in duration and desire to apply business logic to handle business exceptions. It defines that actions are applied immediately and are permanent. Compensation logic is required to allow each participant to undo the operations it has performed within the conversation.

A Web services application can include both atomic transactions and business activities.

Note: The SOAP for CICS feature does not currently support the WS-Transaction specification.

For more information regarding WS-Transaction and WS-Coordination, refer to these Web sites:

http://www.ibm.com/developerworks/library/ws-transpec/
http://www.ibm.com/developerworks/library/ws-coor/

Performance and scalability

SOAP for CICS is now available as a fully supported feature of CICS TS V2.2 and V2.3. It provides support for both inbound and outbound SOAP messages. The SOAP for CICS SupportPac which became available for CICS TS V1.3 and V2.2 in March, 2003, only provided support for inbound SOAP messages.

In this section we will compare the performance and scalability of the SOAP for CICS feature, the SOAP for CICS SupportPac, and HTTP in a CICS TS V2.2 environment. We will also discuss briefly how application design and architecture choices will affect the performance.

The data in this section is based on a simple COBOL SOAP-aware application from the CICS zSeries Performance team at IBM Hursley. A sample of both the request and response message is shown in Example B-3 and Example B-4 on page 399.

Example: B-3 Sample request message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
  <SOAP-ENV:Body>
     <Length>00100</Length>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example: B-4 Sample response message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
   <SOAP-ENV:Body>
      <Commarea>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAX
      </Commarea>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The general observations based on the performance data are:

- Total CPU increases linearly as the transaction rate increases.
- CPU/transaction increases linearly as the return message size increases.
- CPU/transaction increases linearly as the number of elements in the XML message increases.
- HTTP message processing is faster than the SOAP for CICS feature.
- SOAP for CICS feature significantly out-performs the SOAP SupportPac.
- Persistent connections improve SOAP performance.

Throughput

As the transaction rate increases, the CPU usage increases as shown in Figure B-5 on page 400. The graph shows a comparison between the SOAP for CICS feature, the SOAP for CICS SupportPac, and a request being processed by a Web aware application that does not use SOAP for CICS. The SOAP for CICS feature outperforms the supportpac because the feature can function without using the BTS repository dataset. The effects on performance of this dataset are described in “BTS considerations” on page 401.

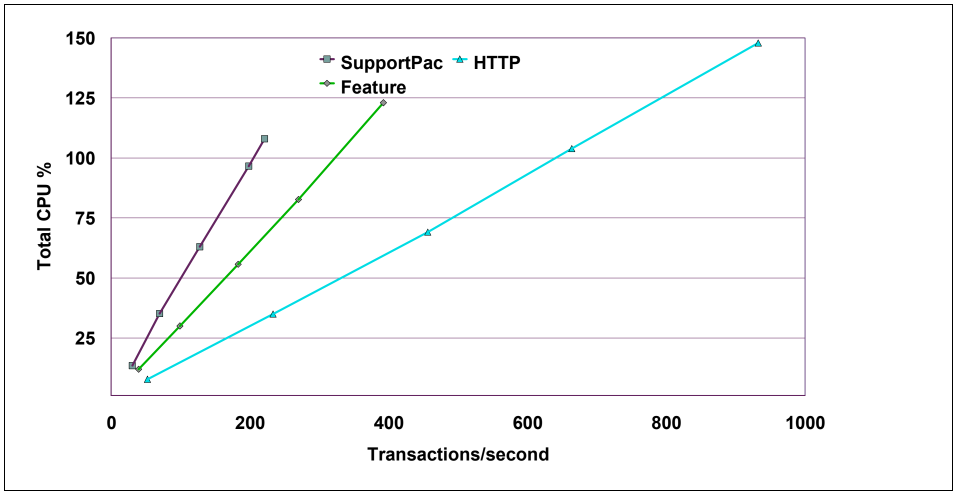


Figure B-5 CPU versus Transactions

Workload balancing

As the transaction rate increases it will become necessary to balance the workload across multiple CICS regions. For HTTP this can be achieved by using the Sysplex Distributor to route the incoming requests to different CICS regions. For WebSphere MQ a queue sharing group (described in *WebSphere MQ for z/OS Concepts and Planning Guide V5.3.1* GC34-6051) can be setup to allow multiple CICS regions to service messages on the same queue.

Once within CICS the existing business logic application that is linked to from the message adapter can be on an AOR and workload managed, for example, by CICSPlex SM. This is shown in Figure B-6.

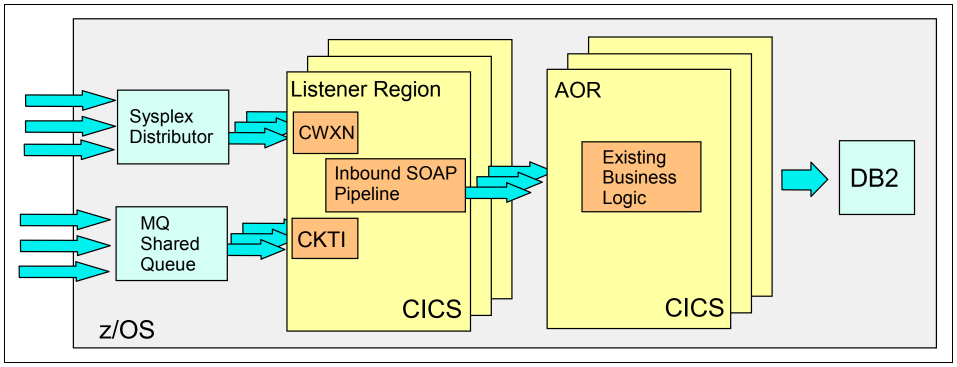


Figure B-6 Workload can be balanced across multiple listener regions and AORs

Transport considerations

There are two transports supported by the SOAP for CICS feature:

- HTTP
- WebSphere MQ

When HTTP is used as the transport, persistent connections outperform non-persistent connections. The reason for this is that, when using persistent connections, the client will reuse both the connection and the CWXN task in CICS on subsequent invocations of a service provider on the same CICS system. If the connection is not persistent then the client will have to establish a new connection and also cause a new CWXN transaction to be created for every invocation of a service provider.

When WebSphere MQ is used as the transport, WebSphere MQ storage requirements, address space storage, data storage, library storage, system LX usage, logging, and backup and recovery environments will have an impact on the performance. For more information, see *WebSphere MQ for z/OS Concepts and Planning Guide,* GC34-6051.

BTS considerations

A BTS repository dataset will be used if WebSphere MQ is the transport. The dataset will also be used if HTTP is the transport in a service requester pipeline and the DEFINE PROCESS command does not use the NOCHECK option.

Using the repository dataset will have the following impacts on performance:

- I/O to the dataset restricts throughput
- Control Interval lock contention on the dataset restricts throughput
- Use of the dataset causes additional MVS Logging

If the repository dataset is used then monitoring data produced by CICS can be used to improve the performance. For information about CICS monitoring, see the *CICS Performance Guide*, SC34-6247.

SSL considerations

Security comes with a performance cost. How much security is enough depends on the business requirements. The security between network nodes over the Internet is traditionally provided using SSL over HTTP (HTTPS). With HTTPS, you can perform mutual authentication of both the sender and receiver of messages and ensure message confidentiality.

If security is needed end to end through the application stack, or if security must be independent of the networking protocol, then other means must be considered. WS-Security specifies authentication and message integrity through XML Digital Signatures and message confidentiality through XML encryption using X.509 certificates in both instances. For details, see Chapter 5, “Security” on page 101. However, there are performance trades-offs that must be understood and evaluated against the overall set of requirements.

A common practice is to combine the two approaches by using SSL for encryption and then using XML Digital Signatures to authenticate the application end points and to ensure message integrity. Keep in mind that SSL will also involve at least one authentication of the server to whom a message is being sent, thus some redundancy is occurring.

Also different SSL key size and the encryption algorithms have different performance impact.

Design and architecture considerations

Designing and architecting a solution that meets the performance requirements has to be considered from the start, not afterwards.

The architecture should guarantee the behavior of the application and operate within a range of acceptable measurements; it must also ensure that the overall behavior of the solution is predictable. Here we will briefly look at the performance impact of message size, message structure, and WebSphere tooling.

Keep message size small

The obvious rule is to keep your payload small and simple. However, in the real world, you do not always have the luxury of adhering to this rule. Larger messages result in longer parsing times. It increases linearly as shown in Figure B-7 on page 403. You should be aware of these impacts and try to minimize the size of the XML message.

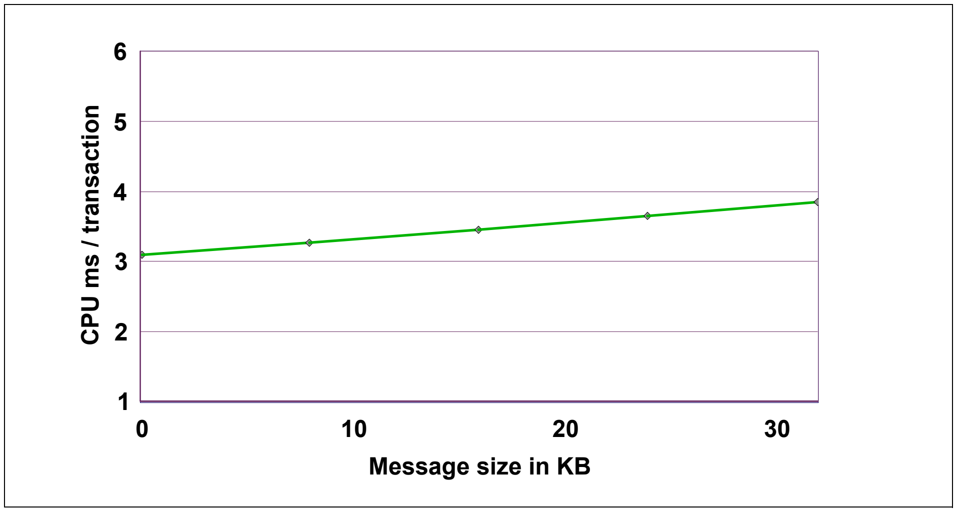


Figure B-7 CPU versus message size

Keep message structure simple

Complex XML structures with nested elements result in longer times for marshalling and un-marshalling the XML elements. XML Elements are extensible and they have relationships as shown in Example B-5. The book is the root element. Title, prod, and chapter are child elements of book. Book is the parent element of title, prod, and chapter. Title, prod, and chapter are siblings because they have the same parent.

Example: B-5 An XML example showing XML elements

```
<book>
   <title>Architecting e-business access to CICS</title>
   <prod id="SG245466" media="paper"></prod>
   <chapter>Introduction to e-business
      <para>what is e-business</para>
      <para>Product options</para>
   </chapter>

   <chapter>Performance and scalability
      <para>CICS Transaction Gateway</para>
      <para>SOAP for CICS features</para>
   </chapter>

</book>
```

There is a linear increase in CPU/transaction as the number of elements increases as shown in Figure B-8 on page 404.

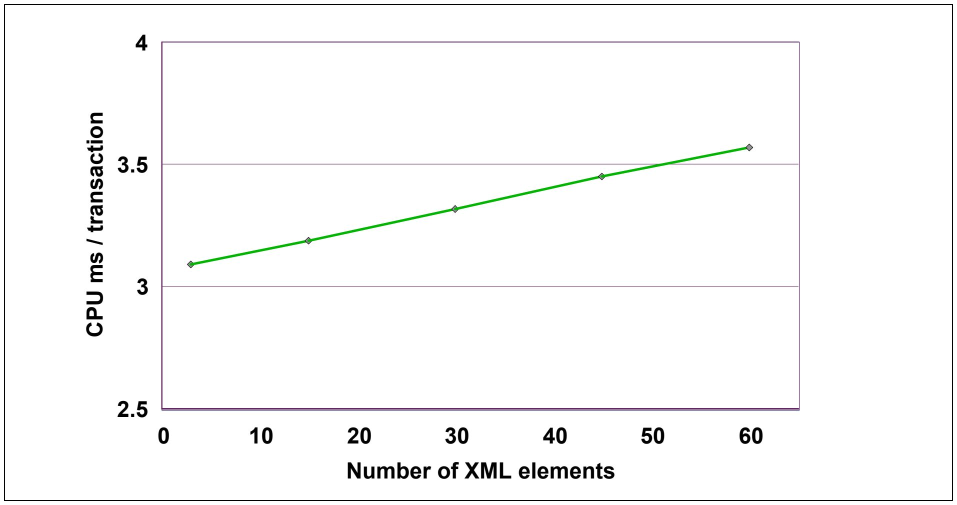


Figure B-8 CPU/Transaction versus number of elements

Other performance considerations

The majority of other runtime performance issues for a CICS SOAP solution will be no different than other CICS applications. As a previous section already mentioned, the following list needs to be considered:

- Hardware, capacity settings
- Operating system parameter settings
- Memory management
- LE runtime
- Transaction logs
- Server logs

Once the solution is operational, then an iterative process is applied to fine-tune a solution by capturing measurements from simulated loads, making adjustments and measuring again to understand their influence. You can iterate as many times as you need until the solution meets the business requirements.

Application development

The SOAP for CICS feature enables existing or new CICS applications, written in any supported programming language, to communicate outside of the CICS environment using the Simple Object Access Protocol (SOAP). The feature supports inbound and outbound SOAP requests. WebSphere Studio Enterprise Developer (WSED) provides a new generation of development tool sets to facilitate the development process.

In this section, we will discuss how to design, develop, and test the Web service of Service Oriented Architecture (SOA) style from an existing 3270 catalog management application named EXCATMAN, which is a greatly simplified COBOL application, as shown in Example B-6. It is a catalog management system for stationery orders and supports both inquiries and orders. In order to keep it simple, we will use the inquiry function only.

Example: B-6 CICS COMMAREA used between the 3270 presentation logic and the EXCATMAN business logic

```
*    Inquire Catalogue COMMAREA structure
           03 CA-REQUEST-ID           PIC X(6).
           03 CA-RETURN-CODE          PIC 9(2).
           03 CA-RESPONSE-MESSAGE     PIC X(79).
           03 CA-REQUEST-SPECIFIC     PIC X(911).
      *    Fields used in Inquire Catalog
           03 CA-INQUIRE-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
               05 CA-LIST-START-REF       PIC 9(4).
               05 CA-LAST-ITEM-REF        PIC 9(4).
               05 CA-ITEM-COUNT           PIC 9(3).
               05 CA-CAT-ITEM OCCURS 15.
                   07 CA-ITEM-REF         PIC 9(4).
                   07 CA-DESCRIPTION      PIC X(40).
                   07 CA-DEPARTMENT       PIC 9(3).
                   07 CA-COST             PIC ZZZ.99.
                   07 IN-STOCK            PIC 9(4).
                   07 ON-ORDER            PIC 9(3).
      *    Fields used in Place Order
           03 CA-ORDER-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
               05 CA-USERID               PIC 9(8).
               05 CA-CHARGE-DEPT          PIC 9(8).
               05 CA-ITEM-REF-NUMBER      PIC 9(4).
               05 CA-QUANTITY-REQ         PIC 9(3).
               05 FILLER                  PIC X(888).
```

Design

Companies need to keep up with today's fast moving marketplace, but traditional mainframe applications are very tightly coupled, and developers often cannot rapidly build applications that satisfy the needs of on demand business. With SOA and Web services existing requirements, enterprise application integration can be wrapped and exposed as services to provide loosely coupled accessibility to other functions, flows and applications, as well as provide a foundation for extending the platform to meet specific business demands as shown in Figure B-9.

We will take the simplest approach to demonstrate how to design and develop a Web service from EXCATMAN. This approach is often referred to as *bottom-up*.

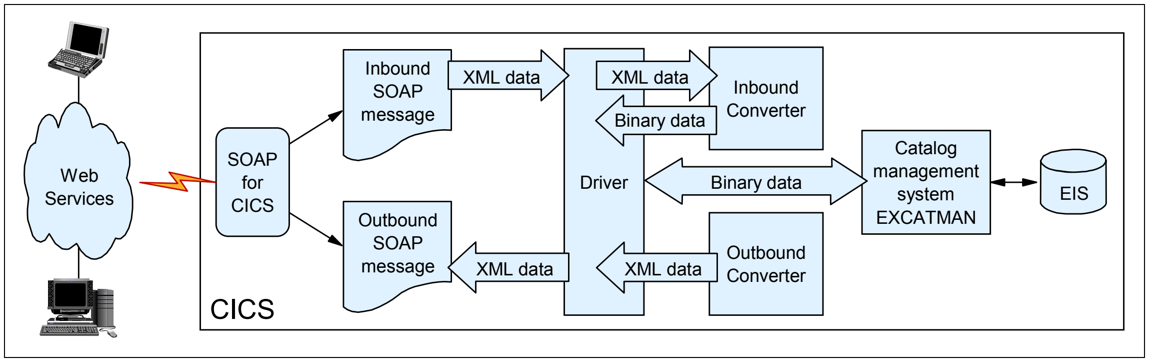


Figure B-9 The bottom-up approach for SOAP enablement

Development

As with any other development, you can write it by hand or take advantage of an IDE. We will list higher level steps of SOAP enabling the application by hand, then go through an IDE aided development approach for the same application and you will see for yourself why WebSphere Studio Enterprise Developer is the recommended tool for the job.

Hand-coded approach

In order to SOAP enable the EXCATMAN, we have to go through following steps to achieve the task. We will not go into detailed implementation here, only outlines the basic steps.

- From a COMMAREA to an XML document
- Implementing a CICS SOAP Web Service Provider
  - The message adapter
  - BTS Handling

  - XML parsing
  - XML generation
  - Building the SOAP body
- Implementing a CICS SOAP Web Service Requester
  - The message adapter
  - Building the SOAP body
  - BTS Handling
- Error Handling
  - Error handling in the Web Service provider
  - Error handling in the Web Service requester
- Handling Namespaces
- Web service test client

WebSphere Studio Enterprise Developer aided development

As you can see, there is a lot of tedious work to be done and many places could go wrong if you don't pay enough attention to the details. Now let's take a look at how the WebSphere Studio Enterprise Developer can help us simplify and speed up the SOAP enablement.

Locating your existing application

In order to locate the application, we need to define and connect to the remote z/OS system by using the tooling for z/OS in WebSphere Studio Enterprise Developer. Once we are connected, then locate the source for EXCATMAN.

XML conversion

1. Create a WebSphere Studio Enterprise Developer project

   In WebSphere Studio Enterprise Developer, switch to the Resource perspective and invoke the New Project wizard and follow the wizard to create a project called WSEDXML.

2. Moving your existing application to WebSphere Studio Enterprise Developer

   Use copy and paste operations on the source COBOL files for EXCATMAN into your local WSEDXML Project.

   If the source file is located in the local file system, then use the **File** -> **Import** operation provided in the Workbench to import the source files into the local project.

3. Invoke the XML Converter generator wizard

   Select EXCATMAN.cbl in the Navigator view and right-click and select **Enable XML** -> **Generate XML Converter** from the menu.

4. Specify input and output files for the wizard

   Use the first page of the XML Converter wizard to select input and output files for the Converter, Driver template, and the XML Schema files as follows:

   Note: The XML Schema files are automatically generated that contains the description of the names and types of XML elements. These elements can appear in an XML document that our program will process and generate.

   – Source file: specify where your existing COBOL program is located
   – Converter folder: specify folder(s) where the wizard will generate converter program(s)
   – Converter file name: specify the name you want to give your converter file.
   – Converter driver file name: specify the name you want to give your converter driver.
   – Generate converters and converter driver to the Input converter file: choose whether you want to generate the converters and converter driver to the file specified in the *Input converter file name* field. The driver will appear first in the file.
   – XSD file folder: specify where the wizard will generate the XML Schema file.
   – XSD file names: specify the name you want to give your XML Schema file(s)
   – Overwrite files without warning: select if you want to overwrite existing output files

5. Specify the generation options

   Use the second page of the XML Converter wizard to specify generation options for your Converter and Driver programs as follows:

   – Program name - specify the value for the program name.
   – Author name - specify the value for the AUTHOR paragraph
   – Driver type - specify the desired driver type. Choose **CICS**. The CICS driver type provides some conveniences to minimize the need for modifications.

– Maximum message size - specify the maximum size of the XML message that will need to be allocated when processing and generating the XML message.

– Code pages - specify code page(s) for the encoding of the inbound and outbound XML documents, and the code page for the host data.

– Inbound Namespace - specify the inbound namespace or accept the default. This is currently not validated.

– Outbound Namespace - specify the namespace container for messages created by the outbound converter.

6. Specify input and output data structures

   We will only use the INQUIRE function of our existing business logic. In order to optimize the flows, we select the CA-REQUEST-ID EXCATMANCOMMAREA field, and only this one. We do the same type of optimization for the output COMMAREA.

7. Generate code

   Click **Finish** to complete the generation process. After the wizard processing completes, you will notice that your Converter and Driver files are generated and displayed in the Navigator view.

8. Modify the CICS SOAP converter driver programs

   You can now modify the converter driver template that was generated by the tool. The section of the WSEDXMLD driver program should be modified as follows in Example B-7 (the necessary changes are highlighted in bold).

Example: B-7 WSEDXMLD driver program

```
* ************************************************************
*             Business Program Binary Interface
* ************************************************************
*       01 DFHCOMMAREA .
        01 EXCATMAN-COMMAREA.

* ------------------------------------------------------------
*              Execute Current Business Program
* ------------------------------------------------------------
           exec cics link
            program  ('EXCATMAN')
            commarea (EXCATMAN-COMMAREA)
           end-exec
```

9. Create the CICS SOAP Web Service provider

   Under **File -> Dataset -> Member selection** of Generate XML converter Wizard window, we will have WebSphere Studio Enterprise Developer generate a single COBOL program, and use the EXCATSOA prefix.

   Edit the EXCATSOAD.cbl file, and change the EXEC CICS LINK to the following as shown in Example B-8.

Example: B-8 Change the EXEC CICS LINK

```
* -------------------------------------------------------------
      *               Execute Current Business Program
      * -------------------------------------------------------------
           exec cics link
            program  ('EXCATMAN')
            commarea (EXCATMAN-COMMAREA)
          end-exec
```

10. Creating the Web Service WSDL file

    The input and output XML Schemas have been generated already. Now we will use them as input to our Web Service WSDL file generation, in order to expose our EXCATMAN Web service to the world by selecting **File** -> **New** -> **Other** -> **Web Services** -> **WSDL** and naming it `EXCATMAN_INQUIRY_Service`. When you click the **Finish** button, the new WSDL file will be generated and added to your project.

    Now we will switch to the **Graph** view of the WSDL source, and create a complete WSDL file as in the following steps:

    – Create a Port Type: EXCATMAN_PortType

    – Create an Operation: EXCATMAN_INQUIRY_Operation

    – Create request and response messages: EXCATMAN_INQUIRY_Request_Message, EXCATMAN_INQUIRY_Response_Message.

    – create parts for both request and response messages: Select the **Type of element** radio button, then browse EXCATSOAI.xsd and EXCATSOAO.xsd respectively.

    – Associate messages with the Port Type: click the **Port Types** tab, and click the **Browse** button, and select the EXCATMANWSDLFile.wsdl file on the pop-up window, then choose the request message for the request and the response message for the response.

    – Create a binding for the Port Type: click the **Bindings** tab, select a SOAP binding type with a CICSSOAPBinding name. Click the **Next** button. We will use the SOAP over HTTP binding type, with *document* style.

– Create the service: select the **Services** tab, name the Web service EXCATMAN_Service, create a new SOAP Port named EXCATMAN_Service_Port, and click the **Next** button, then specify the URL (`http://ipaddress:port/CICS/CWBA/DFHWSDSH/EXCATSOD`), and click the **Finish** button to generate the WSDL file as shown in Example B-9.

– Validate the WSDL file: Right-click the WSDL file and select the **Validate WSDL file** option.

Example: B-9 A complete WSDL file

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="EXCATMAN_INQUIRY_Service"
targetNamespace="http://example.com"
xmlns:tns="http://example.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsd1="http://www.EXCATSOAI.com/schemas/EXCATSOANIInterface"
xmlns:xsd2="http://www.EXCATSOAO.com/schemas/EXCATSOAOInterface">
  <wsdl:import namespace=http://www.EXCATSOAI.com/schemas/EXCATSOANIInterface
location="EXCATSOAI.xsd">
   </wsdl:import>
  <wsdl:import namespace="http://www.EXCATSOAO.com/schemas/EXCATSOAOInterface"
location="EXCATSOAO.xsd">
   </wsdl:import>
  <wsdl:message name="EXCATMAN_INQUIRY_Request_Message">
<wsdl:part name="EXCATMAN_INQUIRY_Request_Message_Part"
type="xsd1:EXCATMANCOMMAREA">
</wsdl:part>
   </wsdl:message>
  <wsdl:message name="EXCATMAN_INQUIRY_Response_Message">
   <wsdl:part name="EXCATMAN_INQUIRY_Response_Message_Part"
type="xsd2:EXCATMANCOMMAREA">
</wsdl:part>
   </wsdl:message>
  <wsdl:portType name="EXCATMAN_PortType">
<wsdl:operation name="EXCATMAN_INQUIRY_Operation">
      <wsdl:input name="EXCATMAN_INQUIRY_OperationRequest"
   message="tns:EXCATMAN_INQUIRY_Request_Message">
</wsdl:input>
<wsdl:output name="EXCATMAN_INQUIRY_OperationResponse"
   message="tns:EXCATMAN_INQUIRY_Response_Message">
</wsdl:output>
</wsdl:operation>
   </wsdl:portType>
<wsdl:binding name="CICSSOAPBinding" type="tns:EXCATMAN_PortType">
```

```
  <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="EXCATMAN_INQUIRY_Operation">
    <soap:operation soapAction="urn:EXCATMAN_PortType" style="document" />
    <wsdl:input name="EXCATMAN_INQUIRY_OperationRequest">
<soap:body parts="EXCATMAN_INQUIRY_Request_Message_Part" use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:EXCATMAN_PortType" />
     </wsdl:input>
    <wsdl:output name="EXCATMAN_INQUIRY_OperationResponse">
       <soap:body parts="EXCATMAN_INQUIRY_Response_Message_Part" use="encoded"
encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
namespace="urn:EXCATMAN_PortType" />
     </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="EXCATMAN_Service">
  <wsdl:port name="EXCATMAN_Service_Port" binding="tns:CICSSOAPBinding">
    <soap:address location="http://ipaddress:port/CICS/CWBA/DFHWSDSH/EXCATSOD"
/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Generating a Web Service test client with WSED

Now that the WSDL file is ready to be used to test the EXCATMAN Web service provider.

Right-click the WSDL file and select **Web Services** ->**Test** with Web Services Explorer. Navigate to EXCATMAN_INQUIRY_Operation in the Web Services Explorer navigator view, give `01INQC` as the value of the ca__request__id field and click the **Go** button in the invocation form; the results of executing the Web Service will be returned.

You can find the complete set of documentation about using the Web Services Explorer in the online WebSphere Studio Information Center.

Data conversion

This section discusses the code page considerations when using the SOAP for CICS feature. XML data is usually sent in UTF-8. CICS does not support code page conversion from UTF-8 to EBCDIC so the SOAP for CICS feature relies on the fact that the first 256 characters of UTF-8 are the same as ISO-8859-1 from which CICS does support conversion to EBCDIC.

Service provider: Inbound pipeline

The way data gets converted within the inbound pipeline differs depending on whether HTTP or MQ is being used as the transport mechanism.

HTTP

If HTTP is used as the transport then the SOAP request is assumed to be transmitted in code page ISO-8859-1. An **`EXEC CICS WEB RECEIVE`** is issued by the inbound transport handler DFHWSTIH to retrieve the data. This specifies that it should be converted to EBCDIC code page 037. The SOAP response sent back to the service requester is assumed to be in code page 037 and is converted to ISO-8859-1 when DFHWSTIH issues the **`WEB SEND`** command.

WebSphere MQ

If WebSphere MQ is used as the transport then the data is converted by MQ on the MQGET call issued by DFHWSDSQ. CICS specifies the default values for the conversion which is MQMD_ENCODING set to `MQENC_NATIVE` and MQMD_CODEDCHARSETID set to `MQCCSI_Q_MGR`.

MQENC_NATIVE and MQCCSI_Q_MGR are the default values set in the MQMD for these parameters. They are described in the *WebSphere MQ Application Programming Reference*, SC34-6062.

Service requester: Outbound pipeline

The different transport mechanisms used will affect how data is converted when CICS acts as a service requester.

HTTP

DFHWSTOH is the outbound transport handler for HTTP. It uses the outbound HTTP support provided by DFHWBCLI. The call to DFHWBCLI specifies that the SOAP request is in code page 037 and is to be converted to ISO-8859-1. When the SOAP response is received it is converted back to code page 037 from the code page of the remote service provider. If the data was UTF-8 then it is first converted to ISO-8859-1 and then that is converted to 037. The reason for this is that DFHCCNV, which handles code page conversion in CICS, does not support conversion with unicode code pages.

WebSphere MQ

DFHWSTOQ is the outbound transport handler for WebSphere MQ. It does not do any data conversion when sending the SOAP request to the remote service provider. It uses MQPUT which will associate the default server code page with the message. The remote service provider would need to convert the message it receives using normal MQ methods. The SOAP response is received using an MQGET call which specifies code page conversion using the same options as the inbound pipeline.

Abbreviations and acronyms

ADS	Application Data Structure
AES	Advanced Encryption Standard
AOR	application-owning region
API	application programming interface
AT	Atomic transaction
BA	Business activity
BLI	business logic interface
BMS	Basic Mapping Support
BTS	Business Transaction Services
CA	certificate authority
CCF	Common Connector Framework
CCI	Common Client Interface
CEX2	Crypto Express2
CEX2A	cryptographic accelerator
CEX2C	cryptographic coprocessor
CORBA	Common Object Request Broker Architecture
CPACF	CP Assist for Cryptographic Function
CPU	Central Processing Unit
CSD	CICS system definition file
DBCS	double-byte character set
DES	Data Encryption Standard
DMZ	De-militarized zone
DN	distinguished name
DNS	Domain Name System
DPL	distributed program link
EBCDIC	Extended Binary-Coded Decimal Interchange Code
ECI	External Call Interface
EIS	Enterprise Information Systems
EJB	Enterprise JavaBean
EJS	Enterprise Java Server
EPI	External Presentation Interface
ESB	Enterprise Service Bus
ESI	External Security Interface
ESM	external security manager
EXCI	External CICS Interface
GIOP	General Inter-ORB Protocol
GUI	graphical user interface
GWAPI	Go Webserver API
HATS	Host Access Transformation Services
HFS	Hierarchical File System
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
ICF	Internal Coupling Facility
ICSF	Integrated Cryptographic Service Facility
IDE	Integrated Development Environment
IFL	Integrated Facility for Linux
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
IRD	Intelligent Resource Director
ITSO	International Technical Support Organization
J2EE	Java 2 Platform Enterprise Edition

JAAS	Java Authentication and Authorization Services
JCA	J2EE Connector Architecture
JDBC	Java Database Connection
JIT	Just-In-Time
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JSEE	Java Secure Socket Extension
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
LE	Language Environment
LTC	Local Transaction Containment
LTPA	Lightweight Third Party Authentication
MAC	message authentication code
MD	message digest
ORB	Object Request Broker
OTS	Object Transaction Service
PEM	Password Expiration Management
QoS	Quality of Service
QR	Quasi Reentrant
RCP	relative cursor position
RDO	resource definition online
REL	Rights Expression Language
RII	redundant I/O interconnect
RLE	Run Length Encoding
RMEB	Resource Manager for Enterprise Beans
RMI	Remote Method Invocation
RMLT	resource manager local transaction
ROI	return on investment
RPC	remote procedure call
RRS	Resource Recovery Services
SBA	Start Buffer Address
SBCS	single-byte character sets
SCA	Service Component Architecture
SF	Start Field
SFM	Service Flow Modeler
SFR	Service Flow Runtime
SIT	System Initialization Table
SNA	System Network Architecture
SOA	service-oriented architecture
SPI	System Programming Interface
SSL	Secure Sockets Layer
SWAM	Simple WebSphere Authentication Mechanism
TCB	Task Control Block
TDES	Triple DES
TLS	Transport Layer Security
TOC	total operating cost
TOR	terminal-owning region
TRUE	Task Related User Exit
TS	temporary storage
UDDI	Universal Description, Discovery and Integration
UOW	unit-of-work
URL	Uniform Resource Locator
URM	user-replaceable module
USS	UNIX System Services
VIPA	virtual IP addressing
VPN	virtual private networks
WLM	Workload Manager
WSDL	Web Services Description Language
WSED	WebSphere Studio Enterprise Developer

XSD	XML schema definition
zIIP	z9 Integrated Information Processor

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 422. Note that some of the documents referenced here may be available in softcopy only.

- *Patterns: Connecting Self-Service Applications to the Enterprise*, SG24-6572
- *Patterns on z/OS: Connecting Self-Service Applications to the Enterprise*, SG24-6827
- *Patterns: Direct Connections for Intra- and Inter-enterprise*, SG24-6933
- *Patterns: Self-Service Application Solutions Using WebSphere for z/OS V5*, SG24-7092
- *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303
- *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133
- *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2*, SG24-6284
- *Host Access Client Package V4 Update*, SG24-6182
- *Using IBM WebSphere Host Access Transformation Services V5*, SG24-6099
- *CICS/ESA and TCP/IP for MVS Sockets Interface*, GG24-4026
- *WebSphere Application Server for z/OS V5 and J2EE 1.3 Security Handbook*, SG24-6086
- *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2*, SG24-6284
- *Securing Web Access to CICS*, SG24-5756
- *A Performance Study of Web Access to CICS*, SG24-5748
- *Revealed! CICS Transaction Gateway with More CICS Clients Unmasked*, SG24-5277
- *OSA-Express Implementation Guide*, SG24-5948

- *IBM WebSphere V5.1 Performance, Scalability, and High Availability WebSphere Handbook Series*, SG24-6198
- *Architecting High Availability Using WebSphere V6 on z/OS*, SG24-6850
- *Workload Managment for Web Access to CICS*, SG24-6118
- *TCP/IP in a Sysplex*, SG24-5235
- *Communications Server for z/OS V1R2 TCP/IP Implementation Guide Volume 5: Availability, Scalability, and Performance*, SG24-6517
- *Exploring WebSphere Studio Application Developer Integration Edition 5.0*, SG24-6200
- *CICS Transaction Server for OS/390 Version 1 Release 3: Web Support and 3270 Bridge*, SG24-5480

Other publications

These publications are also relevant as further information sources:

- *CICS Application Programming Guide*, SC34-6231
- *CICS Application Programming Reference*, SC34-6232
- *WebSphere MQ for z/OS V5.3 Concepts and Planning Guide*, GC34-6051
- *WebSphere MQ for z/OS V5.3 System Setup Guide*, SC34-6052
- *WebSphere MQ for z/OS V5.3 System Administration Guide*, SC34-6053
- *WebSphere MQ Application Programming Guide*, SC34-6064
- *IP CICS Sockets Guide,* SC31-8518
- *CICS Transaction Server V2.3 RACF Security Guide,* SC34-6249
- *CICS Transaction Server V2.3 Performance Guide*, SC34-6247
- *CICS Transaction Server for z/OS V2.3 CICS Operations and Utilities Guide*, SC34-6229
- *CICS External Interfaces Guide,* SC34-6006
- *CICS Intercommunication Guide*, SC34-6243
- *Communications Server for AIX Planning and Performance Guide,* SC31-8220
- *UNIX System Services Planning*, GA22-7800
- *IBM Developer Kit for z/OS Java 2 Technology Edition V1.4.2 Persistent Reusable Java Virtual Machine User's Guide,* SC34-6201
- *CICS TS V2.3 CICSPlex SM Managing Workloads,* SC34-6259

- *Java applications in CICS*, SC34-6000
- *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458
- *WebSphere MQ Security*, SC34-6079
- *WebSphere MQ Security*, SC34-6588
- *CICS Transaction Server for z/OS V3.1 CICS External Interfaces Guide*, SC34-6449
- *CICS Transaction Server for z/OS V3.1 RACF Security Guide*, SC34-6454
- *CICS Transaction Server for z/OS V3.1 Internet Guide*, SC34-6450
- *CICS Transaction Server for z/OS V2.3 CICS Customization Guide*, SC34-6227
- *WebSphere MQ Application Programming Reference*, SC34-6062

Online resources

These Web sites and URLs are also relevant as further information sources:

- IBM CICS SupportPac Library

  http://www.ibm.com/software/ts/cics/txppacs
- WS-Security

  http://www.ibm.com/developer%20works/webservices/library/ws-secure/
- WS-Transaction

  http://www.ibm.com/developerworks/library/ws-transpec/
- WS-Coordination

  http://www.ibm.com/developerworks/library/ws-coor/
- Sun Microsystem's Enterprise JavaBeans Specification V1.1

  http://www.javasoft.com/products/ejb
- Unicode

  http://www.unicode.org
- HTTP Header information RFC

  http://www.ietf.org/rfc/
- The Common Object Request Broker: Architecture and Specification

  http://www.omg.org/technology/documents/formal/index.htm
- XML details

  http://www.w3.org/TR/REC-xml/

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

Architecting Access to CICS within an SOA

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Architecting Access to CICS within an SOA

All the information you need to design your CICS SOA architecture

Fully updated for CICS TS V3.1 and CICS TG V6

Covers CICS Web services, CICS TG, CICS Web support, and CICS Service Flow Feature

With the emergence of service-oriented architecture (SOA), the options for accessing existing CICS assets have become more varied than ever. This IBM Redbook is intended for IT architects who select, plan, and design SOA solutions that make use of CICS assets.

First, we provide an introduction to SOA and the options for transforming CICS assets into SOA solutions. Then we introduce the different architectures and technologies on which a CICS SOA solution can be based.

Next we offer a broad overview of the different technologies available for SOA enablement of CICS applications. We also offer a comparison of some of the current strategic technologies that include the CICS Web services support in CICS TS 3.1, CICS Transaction Gateway, CICS Web support, Link3270 bridge, and the CICS Service Flow Feature.

We continue with an in-depth analysis of the issues pertaining to security, transaction management, performance, application development, and data conversion when designing and developing your SOA solutions.

Finally, we document different customer scenarios, where these technologies have been successfully used to integrate existing enterprise information systems into new SOA solutions.

SG24-5466-05 ISBN 0738496952