CA Gen

Distributed Processing - Overview Guide

Release 8.5

Second Edition

CA Technologies Product References

This document references the following CA Technologies products:

- CA Gen
- Advantage™ Gen

Contact CA Technologies

Contact CA Support

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

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

Providing Feedback About Product Documentation

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

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

Documentation Changes

The following documentation update have been made since the last release of this documentation:

- Layer 1: Transaction Security (see page 45) and C—Transaction Enabler, Server Manager Runtime Exits (see page 143) - Removed the instances of aefsecex.from these topics to address the STAR issue:15968423.

Contents

Chapter 1: Introduction

Computer programs use various implementation techniques to facilitate the interoperation of the pieces of software that make up their complete application. One such technique is client/server. A client/server application distributes an application's logic between code that operates as a client and code that operates a server. Clients and servers cooperate using a mutually understood *application protocol*. Clients typically request the services, or application logic, offered by a server.

Clients and servers can exist on the same physical machine or can be distributed across a computer network.This guide serves as an overview to the CA Gen Distributed Processing client/server application.

Audience

This guide is primarily intended for application developers using CA Gen to build all or part of a Distributed Processing application.

Prerequisite Knowledge

Users should be familiar with developing applications using the CA Gen Toolset and with characteristics common to client/server applications.

Related Documentation

For more information about Distributed Processing applications or related topics, see the following list of references:

- *Action Diagram User Guide*
- *ASP.NET User Guide*
- *Build Tool User Guide*
- *Client Server Design Guide*
- *Distributed Processing—Client Manager User Guide*
- *Distributed Processing—Communications Bridge User Guide*
- *Distributed Processing—WebSphere MQ User Guide*
- *Distributed Processing—Enterprise Java Beans User Guide*

- *Distributed Processing—Proxy User Guide*
- *Distributed Processing—. NET Server User Guide*
- *Tutorial*
- *Tuxedo User Guide*
- *User Exit Reference Guide*
- *Web Generation User Guide*

Distributed Processing Concepts and Terminology

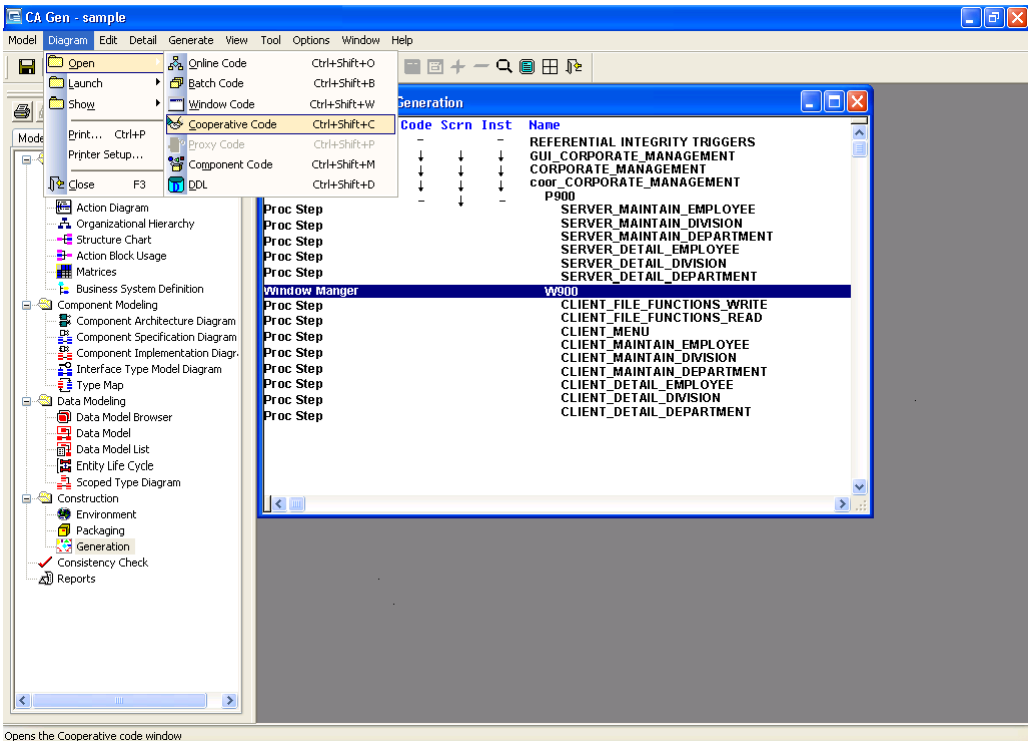
This chapter describes a set of concepts and terminology commonly used when discussing CA Gen Distributed Processing applications and the environments where they execute.

Distributed Processing Application

A CA Gen Distributed Processing (DP) client/server application is software that consists of two or more separate executables, where each executable performs a specific function for the overall application. Communication between the DP application components is accomplished using a mechanism that is known as *cooperative flow*.

Using the CA Gen Toolset or the Client Server Encyclopedia (CSE), a DP application is packaged and generated as a *Cooperative Application*. Cooperative Packaging in a model identifies those procedure steps that operate in a Server Manager and Window Manager.

The Server Manager and Window Manager procedure steps are shown in the following illustration:



Cooperative Flow

A cooperative flow is a generated set of instructions that provides the means for a PStep to pass control and data to a target PStep. For a cooperative flow to be generated, the target PStep must be packaged in a Server Manager. The generation of a cooperative flow occurs for each of the following modeling constructs:

- A Window Manager PStep containing a link type dialog flow, where the target PStep is packaged in a Server Manager
- A Procedure Step USE action language statement, where the target PStep is packaged in a Server Manager
- A USE ASYNC action language statement, where the target PStep is packaged in a Server Manager

The term cooperative flow is also used to describe a user-written application invoking the processing of a PStep packaged as part of a generated Server Manager.

Distributed Processing Server

Distributed Processing Server (DPS) is a term that is used to refer to the target PStep of a cooperative flow request. The DPS is a No Display procedure step that contains business logic. The DPS typically provides the interface for accessing database information. The DPS is packaged in a Server Manager. Server Managers contain one or more PSteps.

Note: In certain environments, the Server Manager is generated code (CICS, IMS, TE, Tuxedo). Other environments that themselves provide the Server Manager functionality (EJB and .NET servers). For the latter, the generated output for a Server Manager provides the context (deployment descriptors) in which the DPS is defined to the environment.

A generated Server Manager provides some functionality that is not available when the execution environment provides the Server Manager support. For example, a CICS generated Server Manager performs the setup processing that establishes part of the execution environment prior to invoking the target DPS. This setup processing includes invoking a set of user exits. These user exits are not implemented in the environments where the environment itself provides the functionality of the Server Manager.

More information:

Security (see page 41)

Distributed Processing Client

The application that initiates a processing request to a DPS is a Distributed Processing Client (DPC), and may be one of the following types:

- **Window Manager**—A CA Gen Window Manager can be either a GUI application or a Web client application. Window Manager clients present data to an end-user display terminal device. The GUI clients are Microsoft Window MFC applications, while Web clients provide a user interface using a Web browser and Web application. The generated Web applications can either be a Java Web Generation client or an ASP .NET Web application.
- **Server Manager**—Certain DPS execution environments (MQSeries, Tuxedo, CICS, IMS, EJB, Web Services, .NET servers) support a DPS initiating a cooperative flow request of another DPS. These server-to-server flows are designated in the model using either the Procedure Step USE or USE ASYNC action language statement.

- **User-written Application**—An application that is not generated using CA Gen can also operate as a DPC application. A user-written application must provide the same calling interface to the DPS as a generated DPC application. To assist an application developer in developing user-written code, CA Gen offers some programming facilities that provide an interface to a DPS. These interfaces include:
  - **Proxy**—A proxy is a generated interface to a DPS. A generated proxy is not an application on its own. A proxy provides a coding interface that a non-CA Gen application can use to communicate with a specific DPS. A proxy does not provide facilities for data presentation. CA Gen can generate proxies usable by applications that execute C, COM, Java, or .NET code.

    Note: For more information, see the *Distributed Processing – Proxy User Guide*.
  - **User-Written EJB Clients**—A user can write a client application that invokes a target EJB without using a proxy.

    Note: For more information about invoking CA Gen EJB from a user-written application, see the *Distributed Processing—Enterprise JavaBeans User Guide.*
  - **User-Written .NET Clients**—A user can write a client application that invokes a target .NET server without using a proxy.

    Note: For more information about invoking CA Gen .NET server from a user-written application, see the *Distributed Processing—.NET Server User Guide*.
  - **User-Written Web Service Clients**—A user can write a client application that invokes a target EJB or TE Web Service without using a proxy.

    Note: For more information about invoking CA Gen Web Service from a user-written application, see the *Distributed Processing— Enterprise JavaBeans User Guide*.

Request/Reply Application Protocol

The CA Gen DP applications communicate using a request/reply application protocol. The DPC invokes the DPS, providing the DPS with its Import View. With one exception, a DPS returns its Export View to the invoking DPC.

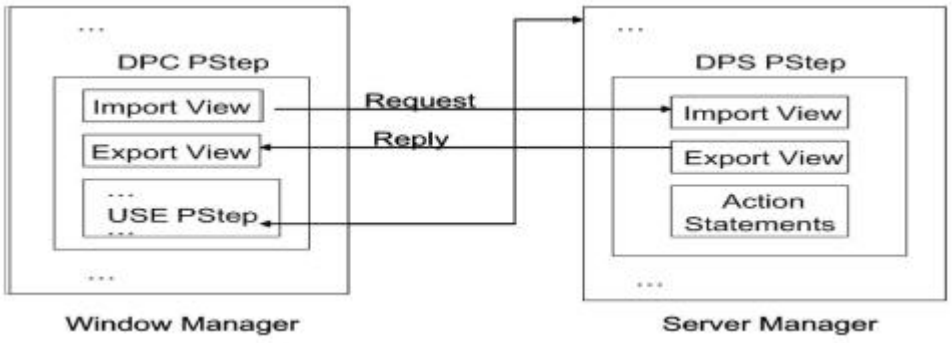
Note: The noted exception is covered as part of the USE ASYNC NO RESPONSE discussion.

From a Server Manager point of view, the request is a DPS Import View and the reply is a DPS Export View.

From a Window Manager point of view, the DPC populates data into a view matching the DPS Import View. The view then flows to the DPS when the DPC executes a cooperative flow (a Dialog Flow, Procedure Step USE or USE ASYNC). The inbound view data is placed into the DPS Import View on entry into the DPS code. The DPS executes and returns data to the DPC in the DPC view that was matched to the target DPS Export View.

Conceptually, server-to-server flows and user-written code behave the same as a Window Manager. They populate a view that matches the DPS Import View. They invoke the DPS with that view as input. The user-written code receives the DPS Export view as output on return from the DPS.

The following illustration depicts a generated Window Manager flowing to a target Server Manager using a Procedure Step USE action language statement. In this illustration, the DPC Import view has been matched to the DPS Import View, and the DPC Export View has been matched to the DPS Export View.



More information:

Asynchronous Cooperative Flows (see page 51)

Synchronous vs. Asynchronous Processing

CA Gen offers two styles of cooperative flow behavior. The first style (synchronous cooperative flows) synchronizes the execution of the DPC with the return of data and control from the DPS. The second style (asynchronous cooperative flows) lets a DPC continue its execution asynchronously to the execution of a DPS.

Synchronous Cooperative Flows

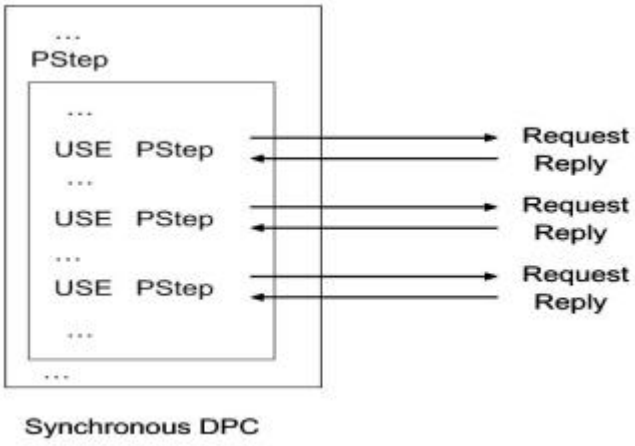
For synchronous cooperative flows, the client application blocks execution until the outstanding response is received. This blocking behavior prevents the requesting application from engaging in any processing that is concurrent to the execution of an outstanding cooperative flow request.

Using synchronous cooperative flows causes a DPC to serialize its processing when it has to flow to other target procedures. Processing within the DPC cannot continue until control is returned to the DPC from a previously executed cooperative flow.

Within a CA Gen model a synchronous cooperative flow is generated when using the following types:

- A link type Dialog Flow in a Window Manager to flow to a target PStep that is packaged in a Server Manager
- A Procedure Step USE (PStep USE) action language statement, where the target PStep is packaged in a Server Manager

The following illustration depicts a DPC that contains three synchronous cooperative flows. The processing within the DPC suspends its execution until the target Procedure Step returns control to the DPC. The synchronous cooperative flow behaves as a Remote Procedure Call (RPC).



A user-written client application that uses a proxy can perform a synchronous cooperative flow to a DPS by using the *execute mechanism* that is exposed in the proxy that is associated with the target PStep.

Asynchronous Cooperative Flows

For an asynchronous cooperative flow, the DPC does not wait for a response from the DPS before continuing its execution. The requesting DPC can continue with other processing while the target server is processing the outstanding request.

A DPC is not obligated to complete an outstanding asynchronous flow request prior to initiating a flow to another DPS. A DPC can have multiple outstanding flows.

The action language that supports an asynchronous cooperative flow consists of four action language statements; USE ASYNC, GET ASYNC, CHECK ASYNC, and IGNORE ASYNC. The USE ASYNC statement initiates a cooperative flow. The GET ASYNC retrieves a cooperative flow response. The other two statements allow the generated application to query the status of an outstanding cooperative flow, or direct that flow's response be ignored.

An asynchronous cooperative flow request is generated when the action language coded in a PStep contains a USE ASYNC action language statement. Similar to the Procedure Step USE statement, the target PStep must be packaged in a Server Manager. The calling PStep and the target PStep may not be packaged in the same load module.

A generated proxy provides an interface that lets a DPS be invoked in an asynchronous manner. This interface is optional and generates the user-written application which uses asynchronous proxy interface.

Using asynchronous cooperative flows is more complicated than using the more traditional synchronous cooperative flow. The logic that initiates the flow to a target DPS is separate from the logic that is involved in retrieving the response. The mechanics of initiating and completing the cooperative flow are the responsibility of the application developer. Additionally, the application developer is given the opportunity to provide error handling within the action diagram using WHEN clauses that can be specified on the asynchronous action language statements.

Note: Asynchronous support is not available to every execution environment and TP Monitor offered by CA Gen. Supported environments are documented in the Asynchronous Support section in the current release of the *CA Gen Technical Requirements* document.

Unit of Work

A unit of work is defined as a collection of operations within an application that must be completed in their entirety before that unit of work is considered complete.

Transaction Processing

Transaction processing is the set of programmatic controls that applications use to mark the start and end of code implementing a unit of work. Transactions are used to synchronize concurrent access to one or more shared resources. If a DPS, the shared resource is a database. A CA Gen DPS operates as a traditional database transaction application.

A DPS transforms data from one consistent state to another such that the modifications are considered to be Atomic, Consistent, Isolated, and Durable. Collectively they are known as the ACID properties and are described as follows:

- **Atomic**—Either all operations that are part of the unit of work that is associated with a transaction complete or none are performed.
- **Consistent**—Data that are modified by the operations of a transaction is transitioned to a consistent state.
- **Isolated**—Even though transactions may execute concurrently, no transaction has visibility into the work in progress of another transaction.
- **Durable**—After a transaction completes successfully, its change survives subsequent failures.

Transactions that complete successfully commit their associated work, leaving any modified data in a consistent state. The unsuccessful transactions abort the work in which they were engaged. Aborting a transaction undoes the operations that are considered to be part of the unit of work that is associated with the failed transaction. The results of any modified data are rolled back to the consistent state that existed prior to the execution of the aborted transaction.

Typical transaction processing for a CA Gen Distributed Processing application is limited to a Server Manager that contains the DPS. A DPS defines a complete unit of work. Upon the completion of the DPS, the DPS Server Manager is synchronized. This synchronization results in a commit or abort being performed, depending on the success or failure of the processing of the DPS.

Distributed Transaction

A distributed transaction is a transaction whose operations run in multiple processes on multiple machines. The Distributed transactions are typically used when the operations in a transaction can be split up for special processing needs or efficiencies, such as when the processing requires access to more than one resource or database.

In a distributed transaction, each process updates the data that is committed as a single operation. A mechanism that is known as *two-phase commit* (2PC) coordinates the completion of the processes that are involved in a distributed transaction.

A CA Gen DPS can issue a cooperative flow to another DPS for extending the unit of work. Some server environments support distributed transactions, such as Tuxedo, .NET servers, and EJBs. In such environments, the CA Gen Server Manager is the software that enables a DPS to participate in a distributed transaction.

More information:

Two-Phase Commit Processing (see page 127)

Distributed Transaction Participant

From a CA Gen modeling perspective, each Procedure Step that is packaged as part of a Server Manager has a set of properties. One of these properties identifies the PStep as being able to participate in a distributed transaction. If this property is set on a Procedure Step, the associated DPS is referred to as a Distributed Transaction Participant (DTP).

Note: The Distributed Transaction Participant check box on the Procedure Step Properties dialog is set in the model using either the Toolset or the CSE Construction Client.

The default value of a Procedure Step DTP property is off (not participating). Therefore, all types of cooperative flows (client-to-server or server-to-server) to the associated DPS result in a new transaction for each inbound request.

The target DPS participates in an existing distributed transaction when all of the following conditions are met:

- The DTP property of a Procedure Step is set.
- The Procedure Step is the target DPS of a server-to-server flow.
- The target environment supports the use of distributed transactions.

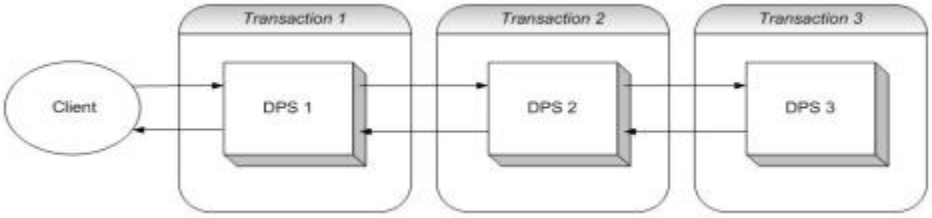
If a transaction context does not exist (no client-to-server cooperative flow), a new transaction context is created prior to giving control to the target DPS.

The commit or rollback operation for a distributed transaction is coordinated across all participating DPSs and is deferred until each DPS involved in the same unit of work has completed.

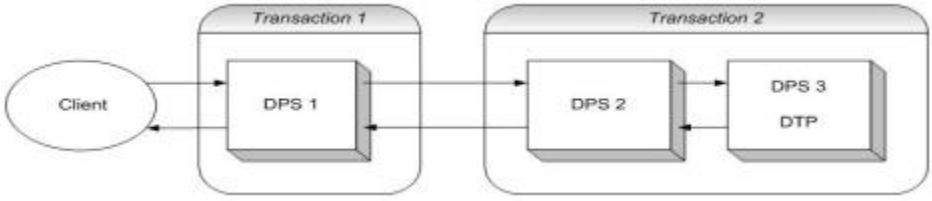
For the clients created using CA Gen, the first DPS in a flow from the originating client always results in the creation of a new transaction. A new transaction is created regardless of whether the DPS has been defined as a DTP.

Java and .NET can support user-written clients that are able to participate in client initiated distributed transactions. In such cases, the user-written application is responsible for creating the transaction context that it extends to the first DPS invoked by the user-written client.

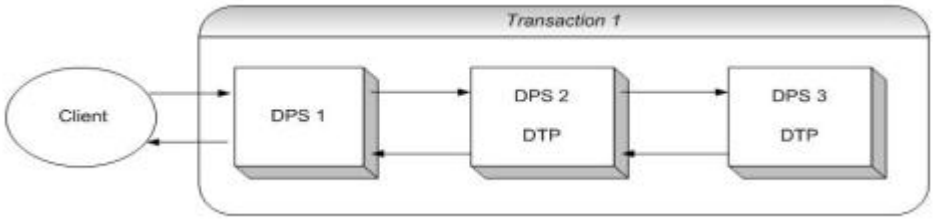
In the following illustration, none of the DPS applications extend an existing transaction by participating in a distributed transaction. Neither of the Procedure Steps associated with DPS 2 or DPS 3 are set as being a Distributed Transaction Participant. Therefore, flows to those DPSs results in the creation of a new transaction.



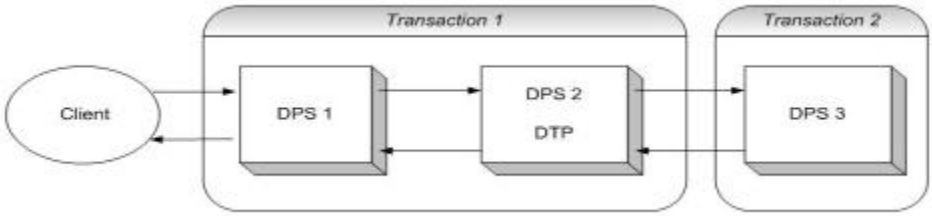
In the next illustration, DPS 2 is not a Distributed Transaction Participant (DTP). Therefore, a new transaction, *Transaction 2*, is created when DPS 1 flows to DPS 2. The flow from DPS 2 to DPS 3 occurs within the same transaction, *Transaction 2*. The synchronization of the work that is performed in DPS 2 and DPS 3 does not occur until DPS 2 returns to DPS 1. The work that is performed as part of *Transaction 2* (DPS 2 and DPS 3) is committed independent to the work performed by *Transaction 1* (DPS 1).



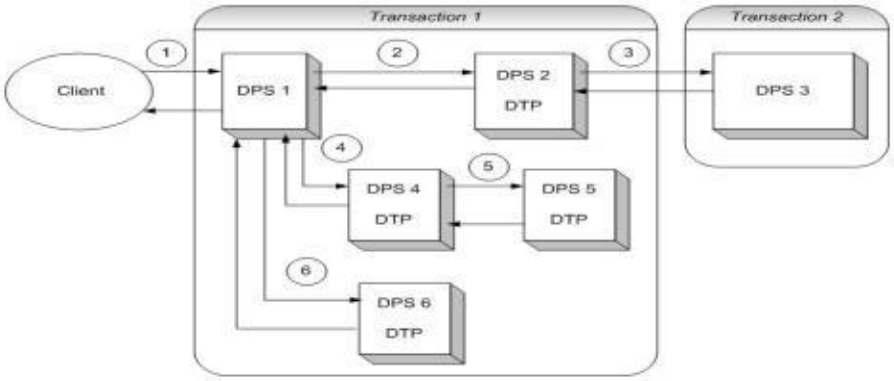
It is possible to extend a distributed transaction to more than one other DPS. The scope of the transaction continues as it includes other DPS applications as each of the participating DPS applications flows to other Distributed Transaction Participants. The following illustration shows three DPS applications participating in a single transaction.



An application may involve both stand-alone transactions and distributed transactions. Each transaction boundary represents its own unique sync-point. The following illustration shows two transactions. *Transaction 1* is a distributed transaction, and *Transaction 2* is a stand-alone transaction. The work that is done as part of DPS 3 is committed (or rolled back) as part of *Transaction 2.* The work that is done within the scope of *Transaction 1* has its own transaction control and is committed (or rolled back) independent of how *Transaction 2* completed.



This illustration has two transactions. *Transaction 1* is composed of five DPS applications. *Transaction 2* is composed of a stand-alone DPS application. The numbers in the circles identify the calling sequence.



Data Representation

A cooperative flow provides the mechanism for a PStep (the DPC) to pass control and data to a remote PStep (the DPS). The data in the Export View of the Pstep is passed to the Remote PStep and received into the Import View of the Remote Pstep.

A DPC and DPS can execute in different environments and on different machine architectures, for example, where the DPC is an ASCII machine and the DPS is an EBCDIC machine. Numeric data would be represented differently in the environments that execute the DPC and DPS.

To accommodate the various environments that support cooperative flows, view data is transmitted in a platform-independent encoded buffer. This proprietary buffer format is known as a *Common Format Buffer* (CFB). The encoded CFB is used to exchange view data that is transmitted between a DPC and DPS. For more information, see Common Format Buffer (see page 21).

CA Gen offers support for cooperative flows where the DPC and DPS operate in the same execution environment. These environments provide transport mechanisms that allow the data to be transmitted without using the CFB. These environments include Tuxedo clients flowing to Tuxedo servers, Java clients flowing to EJB servers, and .NET clients flowing to .NET servers. Each of these environments use a data format or transmission technique that is native to its respective environment.

CA Gen transmits view data in the format that best suits the execution environments of the DPC and DPS involved in the cooperative flow.

General Limitation on Data Passed Between DPC and DPS Applications

When data is being passed between a CA Gen DPC and DPS, cooperative flows do not support nested repeating group views.

Common Format Buffer (CFB)

The Common Format Buffer (CFB) is an encoded byte stream that CA Gen uses to exchange view data during the processing of a cooperative flow. In addition to the import and export view, a CFB contains various pieces of control data that is used to process the cooperative flow.

The format and encoding rules of a CFB are proprietary. Therefore, the explicit details of how a CFB is structured are not discussed. However, here are some characteristics concerning the encoding of a CFB to provide a general sense of its structure and content.

- Applications that target IMS using TCP/IP as the transport require the CFB to include the IMS Request Message (IRM) and a 4-byte trailer. CA Gen uses a 32-byte IRM; therefore, the maximum CFB size is reduced by 36 bytes.

- Many data types are placed into a CFB in a compressed form.
- Numeric data is encoded using a two's complement form with significant ending byte order.
- Null or non-populated fields are encoded as being skipped.
- The uncompressed strings are encoded as fixed-length strings or as null terminated.

The following table gives an overview of the CFB structure. The two types of areas within a CFB are the fixed portion and the relocatable portion. Some relocatable portions are optional depending on the type of CFB being constructed. An offset references the relocatable items and can appear anywhere in the buffer following the header.

Area	Type	Notes
CFB Header	Fixed	Structured data area containing control information primarily used by the runtimes involved in processing a cooperative flow.
User Action (UA)	relocatable	Structured data area containing data detailing the action performed by the user resulting in the cooperative flow.
Next Location	relocatable	Null terminated string containing Next Location system variable.
Model Name	relocatable	Null terminated string containing the model name.
Dialect	relocatable	A compressed string containing the dialect in effect on the client.
Exit States	relocatable	A compressed string containing a dialog flow exit state set by the DPC for use by the DPS.
View Data	relocatable encryptable	An encoded order set of fields containing the import or export view data being exchanged between the DPC and DPS.
Security	relocatable encryptable	Structured data area containing fields used for CA Gen Enhanced Security.

Tuxedo View32 Data Buffer

For a CA Gen distributed application to exploit features that Tuxedo provides the View32 data format of Tuxedo is used to exchange data.

The Tuxedo View32 buffers are C language structures that contain 32-bit field identifiers that are associated with the various elements of the view structure. CA Gen generates a View32 buffer definition when a DPC containing a cooperative flow targets the Tuxedo TP monitor, using Tuxedo as its communications type.

A View32 buffers allow CA Gen applications to use the data-dependent routing feature of Tuxedo. The Tuxedo TP Monitor environment is configured to route data to different servers based on the content of data that is contained in the View32 buffer. Data-dependent routing is achieved by using the capability that Tuxedo provides without modifying the generated application code.

Note: For more information about CA Gen use of View32 data buffers for Tuxedo cooperative flows, see the *Tuxedo User Guide*.

NonStop View Data Exchange

CA Gen generates the buffer definition for NonStop when a DPC containing a cooperative flow targets the Pathway TP monitor. View data is CFB-based and can use the Client Manager or the Communications Bridge as the transport mechanism.

Note: For more information about CA Gen use of view data buffers for NonStop cooperative flows, see the *NonStop Implementation Toolset User Guide.*

.NET Serializable Import and Export Objects

CA Gen generates the view data of the components of a .NET cooperative application in Import and Export Objects. A DPC exchanges the view data with a .NET DPS using a remote procedure call mechanism known as .NET Remoting. The exchange of view data occurs by serializing the Import and Export objects over the .NET Remoting transport.

Note: For more information about CA Gen .NET View Objects, see the *Distributed Processing – Proxy User Guide.*

Java Serializable Import and Export Objects

CA Gen generates the view data of the components of a Java cooperative application in Import and Export Objects. A DPC exchanges the view data with a Java EJB DPS using a remote procedure call mechanism known as Remote Method Invocation (RMI). The exchange of view data occurs by serializing the Import and Export objects over the RMI transport.

Note: For more information about CA Gen Java View Objects, see the *Distributed Processing – Proxy User Guide.*

Web Service Import and Export Objects

CA Gen supports import and export data that are transmitted in a Web Service format for some client and server environments. The Web Service format uses HTTP protocols transmitting SOAP messages with the actual data stored in XML.

Note: For more information about CA Gen web service formats, see the *Web Service Wizard User Guide*.

Data Validation

The CA Gen models capture information relating to view data that is sometimes used to validate the view data. The following list describes the type of data validation checks that occur during a cooperative flow:

- Data Length
- Data Precision (numeric data only)
- Permitted Values
- Mandatory Data

Not all CA Gen cooperative flow environments support data validation. The following list indicates which cooperative flows perform data validation:

- All DPCs invoking an EJB DPS
- All DPCs invoking a .NET DPS

Chapter 2: Anatomy of a DP Application

Distributed Processing Application

A CA Gen Distributed Processing application is made up of many pieces of software, each contributing to the overall processing of a cooperative flow from a DPC to a DPS. Some software is generated from a CA Gen model. Some pieces of software consist of the runtime environments that are licensed as part of CA Gen, while third-party vendors offers other software.

In addition, an application developer can optionally provide certain parts of their DP application as user-written code. For example:

- Clients that are not created with CA Gen
- External Action Blocks (EAB) invoked from a generated application
- User exits invoked during the execution of a DP application

CA Gen DP application developers have many options when determining how to combine the pieces of their overall application. These choices are driven and are often limited by the execution environments that they select to host their DPC and DPS applications.

A DP application is made up of client software, server software, and communications or middleware software to connect the client to the server. The following table identifies the major areas of software that make up a CA Gen Distributed Processing application.

The following table displays an abstract view of a Distributed Processing application:

<table>
<tr><th colspan="6">Distributed Processing Applications</th></tr>
<tr><th colspan="2">Client Execution Environments</th><th colspan="2"></th><th colspan="2">Server Execution Environments</th></tr>
<tr><th colspan="2">DPC application</th><th colspan="2">Communications and Middleware</th><th colspan="2">DPS application</th></tr>
<tr><td>Generated Clients</td><td rowspan="2">Client Runtime</td><td rowspan="2">Utilities and Transport Protocols</td><td rowspan="2">TP Monitors</td><td rowspan="2">Server Runtime</td><td rowspan="2">Generated Servers</td></tr>
<tr><td>Proxy Clients</td></tr>
<tr><td>EABs</td><td>User Exits</td><td>User Exits</td><td>User Exits</td><td>User Exits</td><td>EABs</td></tr>
</table>

Client Execution Environments

The CA Gen clients are executed in many environments. This discussion focuses on the types of Window Managers and user-written applications that can operate as a CA Gen Distributed Processing client. Each of these client environments relies on third-party vendor software to host the client application.

Generated Window Managers

Generated Window Managers can be either a Microsoft Foundation Class (MFC) GUI application or an Internet application. A CA Gen Window Manager implements the following functionality for a Distributed Processing application:

- User interface
- Data validation
- Client workflow processing

Generated Windows GUI Clients

A GUI Window Manager is generated as a C++ MFC Windows application. A generated Microsoft Foundation Class Windows GUI Client can include user-written external action blocks (EABs).

CA Gen provides an MFC Window GUI Runtime that supports the operation of a generated GUI Client. The GUI Runtime offers several user exit functions that provide the opportunity to customize GUI Runtime behavior. For more information about the user exit functions that are invoked from the GUI Runtime.

GUI Clients can flow to a DPS application using one of the CA Gen provided communication runtimes. For more information about the CA Gen provided communication runtimes, see Communications and Middleware (see page 29).

For more information:

User Exits (see page 131)
Overriding Communications Support at Execution Time (see page 57)

Generated Internet Clients

An internet client is a web application that is accessible to an end user through a web browser. An internet client application is generated as either a Java client or an ASP .NET client.

An Internet Client is a combination of one of the following pages:

- Java Server Pages (JSPs), Hyper Text Markup Language (HTML), Java Script, and Cascading Style Sheets (CSS) that are deployed to a Java EE web server and sent to a supported internet browser
- ASP.NET Pages that execute on a Windows server within an Internet Information Service (IIS) and are composed of generated HTML, Java Script, and Cascading Style Sheets (CSS) and are sent to an Internet Explorer browser

An internet client application can include user-written code as external action blocks (EABs).

CA Gen provides separate runtimes supporting Java web clients and ASP .NET web clients. Both the Java and ASP.NET runtimes offer user exits that provide the opportunity to customize runtime behavior.

Internet clients can flow to a DPS application using one of the CA Gen provided communication runtimes. For more information about the CA Gen provided communication runtimes, see Communications and Middleware (see page 29).

More information:

User Exits (see page 131)
Overriding Communications Support at Execution Time (see page 57)

User-written Client Applications

Most user-written client applications use generated proxy code. However, for certain execution environments, user-written code can be written without the use of a proxy.

User-written Clients That Use a Generated Proxy

A user-written client application can use a generated proxy to facilitate a flow to a target DPS. Each proxy is generated for a specific DPS. CA Gen provides support for the following types of generated proxies:

- C Proxy
- COM Proxy
- Java Proxy
- Java Proxy (Classic Style)
- . NET Proxy

Each proxy exposes an interface, an API, which a user-written application uses to invoke the associated DPS. Each type of proxy has its own runtime.

Each proxy runtime offers one or more user exit functions that support customizing the proxy runtime. For more information about the user exit functions available for each of the proxy runtimes, see the *User Exit Reference Guide*.

Each proxy runtime provides an interface to one or more communication runtimes. For more information about the CA Gen provided communications support available for use each type of generated proxy, see Communications and Middleware (see page 29).

For more information about how proxy clients select a communications runtime, see Overriding Communications Support at Execution Time (see page 57).

User-written Clients That Do Not Use a Generated Proxy

In Java and .NET client execution environments, user-written clients can be implemented without the use of a proxy. For example:

- A user-written Java client flowing to an EJB target server
- A user-written .NET client flowing to a .NET target server
- A user-written Web Service client flowing to an EJB or TE Web Service target server

Notes:

- For more information about writing a user-written .NET client without the use of a .NET proxy, see the *Distributed Processing—.NET Server User* Guide.
- For more information about how to write a user-written Java client without the use of a Java proxy, see the *Distributed Processing—Enterprise JavaBeans User Guide*.

Client Runtimes

For each client execution environment, CA Gen provides a client runtime that supports the execution of a generated Window Manager. This support includes functions to manage and control the presentation of the user interface. CA Gen provides the following client runtimes:

- MFC GUI Runtime
- Java Web Generation Runtime
- Java Web View Runtime
- ASP .NET Runtime
- C Proxy Runtime
- COM Proxy Runtime

- . NET Proxy Runtime
- Java Proxy Runtime

Each of the client runtimes supports the use of one or more communication protocols and middleware. Portions of the communications and middleware exist in the client runtime, some in the middleware, and some in the server runtime. The following section describes CA Gen communication and middleware runtime.

Communications and Middleware

Middleware is a collection of software that serves the following purposes:

- A client uses middleware to communicate with a target server.
- Middleware hosts the server and is known as the server execution environment.

CA Gen supports the following server execution environments:

- z/OS CICS
- z/OS IMS
- Windows Transaction Enabler
- Windows .NET Servers
- Windows MQSeries
- UNIX Transaction Enabler
- UNIX Tuxedo
- UNIX MQSeries
- Java EE EJB
- NonStop Pathway

A third-party vendor such as IBM, Microsoft, Oracle, or HP provides all but the CA Gen Transaction Enabler software. The third-party vendor middleware defines the underlying transport protocol that is used for the exchange of a request and a reply between a client and a server. Regardless of which middleware an application developer chooses, the application protocol between a DPC and a DPS remains the same. For more information, see Request/Reply Application Protocol (see page 13).

CA Gen provides the communications runtime for each server execution environment that is supported but only supports the minimum functionality that is required for the execution of a DPS application. The third-party middleware offers some features that are not available. The CA Gen communication runtimes supports only features required to support the execution of a DPS.

CA Gen supports three major communication and middleware components. The first major component is the Transaction Processing Monitor (TP Monitor). The TP Monitor provides the execution environment for the servers.

The next two major components exist external to the server execution environment. They provide the mechanisms that use client applications to communicate with a specific server. These communications and middleware components are as follows:

- Communication Utility programs
- Communications Runtimes

Communications Utility Programs

Not all client environments offer communications support for every server execution environment. Sometimes it is necessary to employ the use of a CA Gen provided communication utility program. These utility programs are stand-alone executables and receive cooperative flow requests from one or more clients.

These utility programs act as gateways to the target server environments they service. They often accept the client input using one protocol and communicate with their servers using another protocol.

In addition to the previous communications protocol conversion processing, these utility programs facilitate cross environment flows.

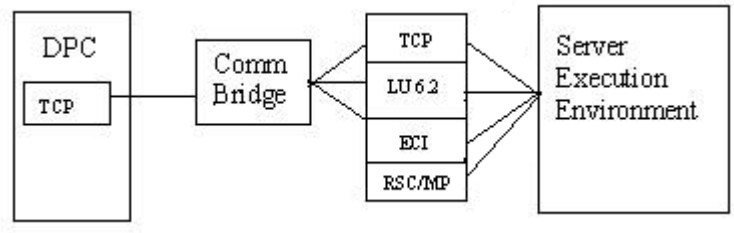
CA Gen offers the following communications utility programs:

- Communications Bridge
- CFB Server

Communications Bridge

The Communications Bridge (Comm. Bridge) supports the TCP/IP Sockets communications from various distributed processing clients and is able to communicate to its target server environment using LU 6.2, TCP/IP, ECI, or NonStop RSC/MP.

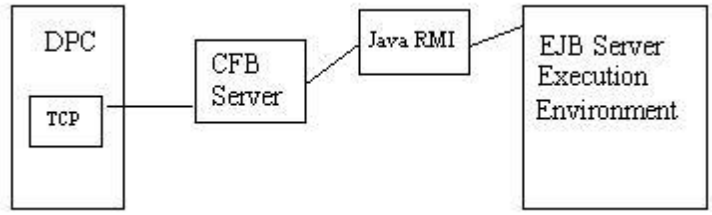
Note: For more information about the CA Gen Communications Bridge, see the *Distributed Processing—Communications Bridge User Guide*.



CFB Server

The CFB Server is a Java program that supports the TCP/IP Sockets communication from distributed processing clients and communicates to EJB distributed processing servers using Java RMI.

Note: For more information about the CA Gen CFB Server, see the *Distributed Processing—Enterprise JavaBeans User Guide*.



Communications Runtime

Clients use a communications runtime to facilitate the processing of cooperative flow to a given target server execution environment. A server execution environment dictates, which transport protocols, a given client can use. Some server execution environments support only one transport protocol, while others support more than one.

CA Gen supports the following transport protocols corresponding to each server execution environment:

Server Execution Environment	Supported Transport Protocols			
z/OS CICS	LU6.2	TCP/IP	MQSeries	ECI
z/OS IMS	LU6.2	TCP/IP	MQSeries	
Windows Transaction Enabler	TCP/IP	Web Services		
Windows .NET Component Services	.Net Remoting			
Windows MQSeries	MQSeries			
UNIX Transaction Enabler	TCP/IP	*Web Services		
UNIX Tuxedo	Tuxedo ATMI	TCP/IP		
UNIX MQSeries	MQSeries			
Java EE EJB	Java RMI			
Java EE EJB Web Services	Web Services			

Server Execution Environment	Supported Transport Protocols
NonStop Pathway	RSC/MP

Note: * Transaction Enabler Web Services is not supported in Linux.

Some client execution environments limit which communications runtimes they offer.

Note: For more information about determining which communication runtime is offered for a particular client execution environment, see the current version of *CA Gen Technical Requirements* document.

For more information about how a CA Gen client selects a communications runtime, see Overriding Communications Support at Execution Time (see page 57).

The following sections describe each type of communication runtime that CA Gen offers:

- Provide a brief description of the communications runtime.
- List the clients that use the runtime.
- Identify more CA Gen documentation that provides a more detailed description of the runtime.

Client Manager

The Client Manager is a CA Gen-provided Windows workstation application that provides an end user with concurrent connectivity to one or more target servers using either CPI/C (LU 6.2) or Sockets (TCP/IP) as the transport. MFC GUI is the client for this service.

Note: For more information about the Client Manager, see the *Distributed Processing—Client Manager User Guide*.

TCP/IP Sockets

The CA Gen TCP/IP communication runtime provides a connection-oriented stream transport that connects a DPC application to the server execution environment that hosts the target DPS application. Additionally, the TCP/IP runtime can also be used to connect a DPC to a CA Gen provided communications utility application such as the Communications Bridge or CFB Server.

Clients: MFC GUI clients, Java internet clients, ASP.NET internet clients, EJB DPS clients (server-to-server), .NET Server clients (server-to-server), User-written clients using C Proxy, COM Proxy, Java Proxy, or .NET Proxy.

Target Server Environment Communication Styles

The Client Manager supports three distinct styles of communications to target servers using the TCP/IP Sockets protocol. You can configure the three styles using the check boxes on the Sockets Server Configuration – Details dialog. The state of the check boxes depends on the target server environment requirements.

All the three styles use the same features of the underlying TCP/IP protocol. Two of the styles differ in the length of the life span of the connection, the third is how the Common Format Buffer (CFB) data is packaged.

Connection Life Time

CA Gen supports two different connection life time modes. The mode that is chosen, persistent compared to non-persistent, depends the needs of the target server environment.

Persistent socket connections are those connections that are long lived. After a connection is established, the same connection object is used for multiple client to server transactions. The connection is maintained until the client or server application performs a connection disconnect or exits. Transaction Enabler (TE) uses this type of connection on UNIX and Windows, z/OS IMS TCP/IP Direct Connect, z/OS CICS TCP/IP Direct Connect, EJB CFB Server, Windows Communications Bridge, and the Tuxedo Proxy Client on UNIX.

Non-persistent socket connections are maintained during a single client to server transaction. After the server returns a response buffer back to the client the server close its connection. A subsequent client request to the same server requires that a new connection is established. This type of connection is supported only when connecting with the z/OS CICS Socket Listener.

Common Format Buffer Packaging

The third TCP/IP communication style concerns how the Common Format Buffer data is delivered to the target server. There are two choices, wrapped for IMS support or unwrapped.

Communications to z/OS IMS TCP/IP Direct Connect require the CFB to be wrapped inside of IMS header data before it is sent to the IMS target server environment. Only z/OS IMS Direct Connect has this requirement. All the other TCP/IP target server environments require a normal or non-wrapped CFB.

WebSphere MQ

The CA Gen WebSphere communication runtime provides the interface mechanisms that a CA Gen DPC application uses to communicate with IBM WebSphere MQ client software. The client software connects a DPC application to an MQ queue manager. A WebSphere MQ queue manager manages and owns the request/reply messages flow between a DPC and DPS using message queues.

Clients: MFC GUI clients, Java internet clients, WebSphere MQ DPS clients (server-to-server), User-written clients using C Proxy, COM Proxy, or Java Proxy.

Note: For more information about how CA Gen uses MQ to support cooperative flows to CA Gen Distributed Processing server applications, see the *Distributed Processing—WebSphere MQ User Guide.*

ECI

The CA Gen ECI communications runtime provides the interface mechanisms that CA Gen DPC applications can use to externally call a CICS DPS application using a CICS Distributed Program Link (DPL). The target DPS appears as if it were called using an EXEC CICS LINK with the COMMAREA option.

Clients: MFC GUI clients, Java internet clients, EJB DPS clients (server-to-server), User-written clients using C Proxy, COM Proxy, or Java Proxy.

Tuxedo

The CA Gen Tuxedo communications runtime uses an Oracle-provided communications library that is collectively known as ATMI. ATMI is a Tuxedo acronym for Application-to-Transaction-Monitor-Interface. An application that uses the ATMI library is a Tuxedo client.

Clients: MFC GUI clients, Java internet clients, Tuxedo DPS clients (server-to-server), User-written clients using C Proxy, COM Proxy, or Java Proxy.

Note: For more information about how CA Gen uses Tuxedo ATMI to support cooperative flows from clients to CA Gen Distributed Processing Tuxedo server applications, see the *Tuxedo User Guide*.

NonStop Pathway

The CA Gen NonStop Pathway communications runtime relies on the Guardian Application and Execution Facility (GAEF) to provide native NonStop services and other services that generated application servers run on HP NonStop. The GAEF Server Interface (IEFSERVS) provided with the Implementation Toolset for NonStop binds to the CA Gen runtime and your CA Gen–generated server code.

Clients: Distributed processing for NonStop Pathway communications is supported only by using CA Gen Client Manager or CA Gen Communications Bridge as intermediate clients.

Note: For more information about how CA Gen uses RSC/MP to support cooperative flows from clients to CA Gen NonStop server applications, see the *NonStop Implementation Toolset User Guide*.

Java RMI

The CA Gen Java RMI communications runtime provides a client application with the ability to invoke an EJB server application. Java RMI is a remote procedure call (RPC) mechanism that is native to Java. Java RMI serializes the import and export view objects of the target EJB. These serialized objects serve as the input and output data of the invoked EJB.

Clients: MFC GUI clients, Java internet clients, EJB DPS clients (server-to-server), User-written clients using C Proxy, COM Proxy, or Java Proxy.

Note: For more information about how CA Gen uses Java RMI to support cooperative flows from clients to CA Gen Distributed Processing EJB server applications, see the *Distributed Processing—Enterprise JavaBeans User Guide*.

Web Services

A Web Service is a software system that is designed to support interoperable machine-to-machine interaction over a network. CA Gen Web Services Middleware runtime provides a client application with the ability to invoke an EJB or TE Web Service server application.

Clients:

- Java Web Generation clients, Java Web View clients, Java EJB servers flowing to other CA Gen servers and user-written clients using Java Proxy use the Java Web Services middleware.
- MFC GUI clients and user-written clients using COM Proxy and C Proxy use the C Web Services middleware.
- ASP.NET internet clients, .NET Servers flowing to other CA Gen servers and user-written clients using .NET Proxy use the C# Web Services middleware.

Note: For more information about how CA Gen uses Web Services middleware to support cooperative flows from clients to CA Gen Distributed Processing EJB Web Service server applications, see the *Distributed Processing—Enterprise JavaBeans User Guide*.

.NET Remoting

CA Gen .NET Remoting provides a C# client application with the ability to invoke a .NET Server application. .NET Remoting is a remote procedure call (RPC) mechanism that is native to the .NET environment. .NET Remoting serializes the import and export view objects of the target .NET Server. These serialized objects serve as the input and output data of the invoked .NET Server. ASP.NET internet clients and .NET Proxy are the clients for this service.

Note: For more information about how CA Gen uses .NET Remoting to support cooperative flows to CA Gen Distributed Processing .NET Server applications, see the *Distributed Processing—.NET Server User Guide*.

.NET Cross-Environment Communication Runtimes

CA Gen provides support to flow from a non-C# execution environment to a .NET Server. For example, GUI clients can flow to .NET Servers.

The C to C# (C2CS) cooperative flow runtime supports flows between CA Gen applications executing in C/C++ and .NET Servers.

Note: For more information about the way CA Gen applications use the C2CS runtime to support cooperative flows to CA Gen Distributed Processing .NET Server application, see the *Distributed Processing—.NET Server User Guide*.

Behavior Differences

Differences exist in the communication processing CA Gen can use to support a DPC to DPS cooperative flow. These differences are typically a result of capabilities that are provided as part of a given communications protocol and not available in others. Here are some illustrations.

- CA Gen uses a conversational protocol as provided by LU6.2. This protocol provides synchronization processing that CICS DPS and IMS DPS applications use to confirm deallocation status. The CA Gen TCP/IP implementation does not provide a conversational protocol and as such does not provide a comparable synchronization process.
- MQSeries provides a timeout capability for both send and receive processing. The CA Gen TCP/IP runtime does not provide a timeout capability.
- Some communication runtimes integrate security into their execution environments by the use of a User ID and Password as part of their communications protocol. Others use user exits to provide similar capability.
- Not all communication and middleware components support the use of asynchronous cooperative flows.

- The z/OS CICS TCP/IP product includes two connection implementations:
  - CICS Socket Listener—This implementation uses the CICS Socket Listener program TISRVLIS. TISRVLIS passes the connection socket to the CA Gen server application. The application server manages the socket and closes the socket when execution completes. This is known as a non-persistent socket connection as the socket connection is not maintained from one client request to the next.
  - Direct Connect for CICS—This implementation uses a persistent socket connection where the single socket connection is maintained throughout the life of the application.

  Note: Direct Connect for CICS support is no longer supported. Earlier versions of this implementation are supported as long as the software release in which it was delivered is supported.

Server Execution Environments

CA Gen supports various target environments capable of executing a CA Gen server application. Within a CA Gen model, each server manager can be generated and built for one or more server execution environments.

A server execution environment is comprised of the following items:

- Generated Servers
- Server Runtime
- TP Monitor

Generated Servers

The generated code of a CA Gen server application consists of a server manager and a set of one or more server Procedure Steps. The server manager provides the entry point into the server application. The server manager establishes the necessary execution environment before giving control to the target server procedure step.

The individual procedure steps that are contained in a server manager implement an elementary process as defined in the CA Gen model. The procedure steps implement the business rules of the application.

Server Runtime

CA Gen provides a server runtime for each server execution environment. These server runtimes support the execution of the server manager within their associated execution environment. This support includes code that provides interfaces the server manager uses to manage resources (databases), manage transactions, and control flows to and from other server applications.

The server runtime also provides a set of commonly used subroutines the generated server code can use for the following operations:

- Obtaining input data
- Return output data
- Trace
- NLS
- Date/Time formatting

Transaction Processing Monitor (TP Monitor)

The TP Monitor that hosts the server application usually defines a server execution environment. A TP Monitor is the execution environment that hosts the server application. The TP Monitor provides the capacity, control, and management support necessary to concurrently execute one or more transaction applications.

A CA Gen DPS is generated for a specific TP Monitor. The following list describes the TP Monitors supported by CA Gen:

- CA Gen Transaction Enabler (TE)
- IBM CICS
- IBM IMS
- Oracle Tuxedo
- Java Enterprise JavaBeans (EJBs)
- Microsoft .NET Component Services
- NonStop Pathway using TMF
- EJB Web Services

In addition to the TP Monitors mentioned in the previous list, IBM WebSphere MQ can also be considered a TP Monitor when it is used separate from CICS and IMS. WebSphere MQ triggers the target DPS's execution where the server executes as a stand-alone process operating directly on the host system.

The choice of TP Monitor limits other characteristics that shape a server execution environment.

Example:

- Operating System
  - CICS and IMS are TP Monitors only available on z/OS
  - Tuxedo is only available on UNIX
  - EJB is only available on a Java Platform JVM
  - Transaction Enabler is only available on Windows and UNIX
  - NonStop Pathway is only available on NonStop systems
- Language
  - CICS and IMS servers are generated in COBOL
  - Transaction Enabler and Tuxedo servers are generated in C
  - EJB servers are generated in Java
  - .NET servers are generated in C# (C sharp)
  - NonStop Pathway servers are generated in C
- Database

  For information about databases that are offered within a particular server execution environment, see the current version of *CA Gen Technical Requirements* document.

Behavior Differences

The generated Server Procedure Steps execute the same regardless of the server execution environment. However, the execution environments can impose operational differences.

Example:

- Distributed Transaction Support is only available when deploying DPS applications as a Tuxedo Server, a Java EJB, or .NET Server.
- Asynchronous cooperative flows between servers are only available when the DPSs are deployed as MQSeries servers.
- Obtaining a user ID context for a DPS that populate the USER_ID system attribute is not available when the DPS operates as a Java EJB or .NET Server.

- EJB and .NET Servers allow a DPS to have multiple database connections.
- Only certain environments support coding non-CA Gen user-written clients without the use of a generated Proxy.

Note: For more information about third-party software that is supported for a particular execution on environment, see the current version of *CA Gen Technical Requirements* document.

Chapter 3: Security

Security Overview

A CA Gen DP application incorporates security in multiple ways. Each portion of the security implementation depends the execution environment hosting the DPC and DPS.

Implementing a security solution for a CA Gen Distributed Processing application can involve processing that occurs at multiple points along the execution path of a cooperative flow.

Example:

- Security processing is invoked from within the DPC and DPS application programs.
- The security processing that executes as part of or on behalf of a DPC application typically gathers the security data that is supplied by the end user, such as a user ID and password.
- The security processing that executes as part of, or on behalf of a DPS application validates the security data that is associated with a client cooperative flow request. Depending on the particular DPS security implementation, additional security data can be provided. For information about how additional security data can be used, see Server Security Processing (see page 44).
- Security processing can be integrated as part of the vendor products. This security processing is invoked as part of the third-party communications products or the TP Monitor that supplies the execution environment for the DPS.
- Security processing can be invoked from a CA Gen provided communications program such as the Client Manager or Communications Bridge.

A security solution can involve one or more security processing choices which are as follows:

- The CA Gen applications can use security processing that the execution environment provides. This security processing is external to the application itself. For example, Java clients flowing to EJBs use Java EE roll based security.
- CA Gen provides programmatic support that application developers can use to implement security such as the USER_ID, CLIENT_USER_ID, and CLIENT_PASSWORD system attributes. This security implementation provides a tight interaction between the CA Gen Distributed Processing application and its execution environment.

The Application developers can also implement a *create-your-own security* implemented internal to the Distributed Processing application itself. In this case, the security processing is coded as part of the DPC and DPS.

Client Security Processing

Each DPC application type that originates a user request (GUI, proxies, or internet client) provides some facility for placing data into the CA Gen variables CLIENT_USER_ID and CLIENT_PASSWORD. For generated the DPC applications these variables are read/write system attributes. For the proxies, they are application data variables with corresponding get and set methods.

For the DPC applications that flow to a non-EJB and a non-. NET Server, the supporting runtime invokes a client security user exit (such as WRSECTOKEN). This client security user exit indicates whether the CLIENT_USER_ID and CLIENT_PASSWORD variables are used as security data associated with a cooperative flow request. That same user exit also influences how the runtime incorporates the security data into the cooperative flow request.

For the flows between a DPC and DPS that operate in the same execution environment and have processing that does not use a CFB or a Tuxedo View32 buffer (Java clients flowing to EJBs and .NET clients flowing to .NET Server), the CLIENT_USER_ID and CLIENT_PASSWORD system attributes are always transmitted (serialized) as part of the Import data that are exchanged between the DPC and the DPS.

For the DPC applications that flow to a non-EJB and non-. NET Server, the client security user exit influences how the security data is processed, as follows:

- The security user exit can indicate that the CLIENT_USER_ID and CLIENT_PASSWORD variables should NOT BE USED to populate any part of the cooperative flow (Security None).
- If a DPC client security user exit returns *Security None*, it is possible for the GUI DPC application to have security data that is provided by CA Gen, which is associated with the cooperative flow request. The Client Manager can specify the security data. This security implementation requires that you specify CA Gen as the transport mechanism for the cooperative flow. In this case, CA Gen indicates that the DPC uses the Client Manager.

The client security user exit can indicate that *Standard Security* be used, as follows:

- For the flows that use a Common Format buffer (CFB), the data in the CLIENT_USER_ID and CLIENT_PASSWORD variables are populated into corresponding fields that are located in the header portion of the Common Format Buffer (CFB). For more information, see Standard Security.
- For the flows using Tuxedo View32 data structure, the data in the USERID and PASSWORD fields is populated in corresponding fields in the View32 data structure.

The client security user exit can indicate that *Enhanced Security* is used, as follows.

- For the flows that use a Common Format Buffer, the CLIENT_USER_ID and CLIENT_PASSWORD variables are populated into fields that are located in the optional security-offset portion of the CFB. In addition, when using Enhanced Security, the CLIENT_USER_ID is populated into the header portion of the CFB.
- If the client security user exit indicates that Enhanced Security is used, the exit can also indicate that the data that is placed into the optional security-offset field be used by the Client Manager or Communications Bridge.
- Lastly, if the client security user exit indicates that Enhanced Security is used, the user exit can also supply a security token. This token is added to the security portion of the CFB. The security token can be used for passing security token data to an external authenticating process such as Kerberos.

Note: For the flows using Tuxedo View32 data structure, the data in the USERID, PASSWORD, and security token fields is populated in corresponding fields in the flows View32 data structure.

For a DPC application that flows to an EJB or .NET Server, the client security user exit provides the DPC application the opportunity to provide a user-defined security object. This object is similar in function to the security token described earlier.

The DPC GUI clients can include security processing that a CA Gen communications program provides such as the Client Manager. Other DPC client types can involve the use of a Communications Bridge.

The GUI DPC applications use the Client Manager as part of their security solution by having the Client Manager supply a user ID and password for a cooperative flow.

Note: For more information about security processing specific to the Client Manager, see the *Distributed Processing—Client Manager User Guide*.

Customers of CA Gen can optionally use a Communication Bridge to service cooperative flows from many clients. The Communications Bridge can be involved in security processing by utilizing the security data that is transmitted in the CFB as part of the cooperative flow.

Note: For more information about the security processing specific to the Communications Bridge, see the *Distributed Processing– Communications Bridge User Guide*.

Client-side security processing can also involve security processing that is provided as part of a third-party vendor product, as follows:

- IBM MQSeries can carry out authorization checks for each resource that is accessed during the processing of a cooperative flow.
- Flows to CICS that uses ECI require IBM Universal Client software. The Universal Client can optionally supply a User ID and password for flows that do not specify any security data.
- Flows to EJBs can use the Java EE roll base security mechanism.

For specific details about how to incorporate security processing that is external to CA Gen cooperative processing, see the relevant vendor documentation.

For more information:

z/OS Security (see page 167)

Server Security Processing

Regardless of how the client-side security data is provided and later associated with a cooperative flow, security data serves two primary objectives, as follows:

- Security data is used to authorize the execution of a DPS application. Server-side security processing uses the associated security data to grant execution access to those users it deems authorized.
- In certain execution environments, the security data is used to establish a user context under which the target DPS execute. In these execution environments, the CA Gen USER_ID system attribute is populated with the value of the user ID under which the server execute.

In other execution environments accessing the USER_ID is not allowed by the environment and would therefore use the USER_ID system attribute unavailable (EJBs and .NET Servers). More specifically, accessing the USER ID is not feasible; therefore, the TIRUSRID user exit does not exist in these server execution environments.

In those environments where the user ID context is accessible, the TIRUSRID user exit is invoked to obtain the user ID from the execution environment. This user exit is called soon after the Server Manager begins executing and before control is passed to the target DPS. The value of the USER_ID system attribute is populated as a result of executing TIRUSRID user exit.

Granting the execution access is security processing that can occur in more than one location along the execution path to a DPS. There can be up to as many as five layers of security processing that is incorporated when executing a DPS. These layers include the following security:

- Layer 1: Transaction Security
- Layer 2: Server Manager Security
- Layer 3: Application Security
- Layer 4: Procedure Step Security
- Layer 5: Function Security

Each layer refines the granularity of what is being protected. The user that is associated with a cooperative request must be granted access at each layer that is implemented for processing to proceed. Depending on the execution environment, not all layers of security within a DPS is implemented.

Layer 1: Transaction Security

Transaction security authorizes the activation of the transaction and lets control be passed to the DPS application. Layer 1 is comprised of security processing that is considered to be external to the DPS application.

Transaction security can validate that the user associated with a cooperative flow request is authorized to execute in the target environment. This processing can be considered similar to that of a logon operation. If the user associated with the cooperative request is not validated, the transaction does not get started and the Server Manager associated with the request does not receive control.

Additional transaction security validation can occur after the user associated with the cooperative request is validated as being an authorized user of the target system. The additional security processing validates the user is authorized to execute the specific transaction that is associated with the target DPS.

In certain execution environments, a DPS is identified using a transaction code (trancode). Users are granted access to execute certain trancodes. If access to a trancode is denied for a given user, the Server Manager that is associated with the specified trancode never receives control.

Layer 1, Transaction Security, is only available in certain execution environments (TE, Tuxedo Proxy Client, CICS, TMF, IMS, EJBs, and .NET Servers).

Third-party execution environments such as CICS, IMS, EJB, and .NET Servers offer transaction security processing that is unique to their respective environments. For example, if your execution environment is Pathway, an application can use Remote Server Call (RSC/MP) to invoke NonStop TMF servers to protect the integrity of server transactions. See the vendor documentation for specific details about how these environments provide transaction security.

For those environments that do not provide external transaction security, a comparable opportunity to validate user access and authorization can be accomplished in Layer 2.

Note: Beginning with Layer 2, the application load module, or Server Manager containing the target DPS, receives control and begins its execution. The security processing from Layer 2 through Layer 5 is implemented in application logic that is provided by the customer. This customer logic is implemented in the form of user exits and code that is implemented in, or invoked from, the server application itself (implemented in action language or an external action block).

Layer 2: Server Manager Security

In certain environments that do not provide external transaction security (Layer 1), it is possible for the Server Manager to validate authorization of the user of a cooperative flow request. The generated Server Manager invokes the TIRSECR user exit as part of preparing to pass control to the target DPS.

The Server Manager provides TIRSECR with the user ID and transcode that is associated with the cooperative request. The TIRSECR user exit is used to validate the user ID. In addition, TIRSECR can ensure that the user is authorized to execute the specified transcode.

The user ID value that is passed to the TIRSECR user exit is obtained from the TIRUSRID user exit. The user ID is made available in the Server Manager system attribute USER_ID.

In most cases the use of TIRSECR is not necessary if transaction security (Layer 1) is incorporated before launching the Server Manager containing the target DPS.

If EJB and .NET Servers, the USER_ID system attributes and user exits comparable to the TIRUSRID and TIRSECR user exits do not exist.

Layer 3: Application Security

As part of the processing that prepares to pass control to the target DPS, the generated Server Manager code invokes a second security user exit, TIRSECV.

For the non-EJB and non-.NET Server DPS applications:

- The TIRSECV user exit is invoked shortly after TIRSECR (see Layer 2: Server Manager Security).
- TIRSECV is invoked to process data contained in a cooperative request's enhanced security data area (see the Common Format Buffer (CFB) (see page 21) discussion occurring earlier in this chapter).
- TIRSECV provides the following:
  - The security data that is passed in the enhanced security data area contains both the CLIENT_USER_ID and CLIENT_PASSWORD values. The DPC sets the CLIENT_USER_ID and CLIENT_PASSWORD values.These values can be used in TIRSECV to determine if the user is valid and is authorized to execute the transcode that is associated with the cooperative flow request.
  - The enhanced security is located in an area of the CFB that is encrypted before the request buffer being transmitted to the DPS. TIRSECV is invoked after the Server Manager decrypts the buffer area containing the enhanced security data. See the Encryption and Decryption (see page 50) section for more details.
  - In addition to CLIENT_USER_ID and CLIENT_PASSWORD, the optional security token field is passed to the TIRSECV user exit. The client security user exit (for example, WRSECTOKEN) specifies the security token.
  - The security token can be used when the DP application incorporates an external authenticating process (for example, Kerberos).

For the EJB and .NET Server DPS applications, the CLIENT_USER_ID, CLIENT_PASSWORD, and optional security object are passed to the TIRSECV user exit.

Layer 4: Procedure Step Security

This layer of security is intended to validate access to the target procedure step. The application developer implements and validates the logic for access to the DPS and is coded as part of the action language the implements the DPS. The validation logic can be implemented in a common action block or can be external in one or more external action blocks.

The security processing that implements procedure step security can be accomplished by having one of the first few statements of the target Procedure Step call or can use a common or external action block. The called/used action block determines if access to the procedure step is granted. Typically, the called/used action block would import the user ID, transaction code, and other user-defined data that is needed to validate the request. The DPS itself grants access to its own processing using security data that provide by the DPC. The DPC provides this user-defined security data as part of the Import View of the DPS.

Layer 5: Function Security

Function security is intended to restrict access to specific processing of individual functions within a DPS. It can be implemented using a similar approach as described in Layer 4. One addition would be that an identifier that represents the secured function should also be imported to the called/used action block. The called/used action block would determine if access to the specified function should be granted.

Security Data

A cooperative flow request to a secure DPS must contain security data that is used to authorize execution access for the user that is associated with the cooperative flow request. CA Gen offers three ways in which security data can be provided as part of a cooperative flow:

- Standard Security
- Enhanced Security
- Roll Your Own Security

Standard Security

When using Standard Security, the security data (user ID and Password) is placed into fields that are located in the Common Format buffer (CFB) (see page 21) header.

The security data that is provided using this technique is used with Layer 1 and indirectly with Layer 2 (the USER_ID system attribute and transcode that is authorized by the TIRSECR user exit).

The User ID and Password values can be provided in one of two ways:

- In the implementations that do not use a Client Manager, a DPC application provides the User ID and Password using the CLIENT_USER_ID and CLIENT_PASSWORD variables. The client security user exit (WRSECTOKEN) indicates that these values be placed into the CFB header.
- In the implementations that use the Client Manager, the Client Manager provides the User ID and Password.

  Note: For more information about how the Client Manager can be configured for security processing, see the *Distributed Processing—Client Manager User Guide*.

The security data that is transmitted in the CFB header is not directly accessible to the logic of a DPS application. However, this security data, in most DPS execution environments, is used to establish the user context under which the DPS execute. For example, the supplied user ID and password are used to sign-on the user. The signed-on user establishes the context under which the transaction execute. The user ID value that is associated with the signed-on user is populated in CA Gen's USER_ID system attribute by way of the TIRUSRID user exit.

In certain execution environment the sign-on processing occurs by a third-party environment where the security data has to be passed as is. Therefore, when processing occurs in Layer 1 where the access validation occurs external to the DPS, the security data has to be available in a known form. To accommodate these execution environments, the header portion of the CFB cannot be encrypted.

Enhanced Security

When using Enhanced Security, the security data (user ID, Password, and optional security token) is placed into the security-offset portion of the CFB. Additionally, the user ID is placed into the UserID field that is located in the Common Format buffer (CFB) header.

As for a DPC application provides the User ID and Password using the CA Gen CLIENT_USER_ID and CLIENT_PASSWORD variables. The client security user exit (WRSECTOKEN) indicates that place these values into the security-offset. The client security user exit provides a return code indicating the flow use Enhanced Security.

As for the optional security token, the data that is used to populate the security-token within the security-offset is provided by the client security user exit. So, in addition to indicating that Enhanced Security is used, the client security user exit can optionally provide data that is used to populate the security token.

The security token is a byte array that serves as data that is passed in the CFB as part of the cooperative flow request. This data is provided as input to the server security Layer 3—Application Security user exit. The security token can be used as part of a Kerberos, or "Kerberos-like" implementation that is intended to authenticate the end user.

Create Your Own Security

Create Your Own Security provides security data that supports Layers 4-access and 5-access validation. The security data and related processing is built into the model definition. The DPS defines security data in its Import View. The invoking DPC populates the security data in the Import View of the DPS as part of preparing to perform the cooperative flow request. Assuming all other security processing lets the DPS execute, the DPS get control. The DPS can then validate the supplied security data using processing that is coded in its associated action diagram or external action block (EAB).

Encryption and Decryption

For those cooperative flows that use a CFB to exchange view data between a DPC and DPS, CA Gen offers the opportunity to have a portion of the request or the response that is transmitted in an encrypted form.

The use of encryption and decryption can occur regardless of the type of security processing being used (SECURITY_NONE, SECURITY_STANDARD or SECURITY_ENHANCED).

Two sets of user exits are invoked during the processing of a cooperative flow. One set lets the request be encrypted by the DPC and decrypted by the DPS. The other set lets the response be encrypted by the DPS and decrypted by the DPC. The two sets of encryption/decryption user exits can be used independently from the other. That is, the request can be encrypted without having to encrypt the response. The same is true for the response. The response buffer can be encrypted without having to encrypt the request.

Notes:

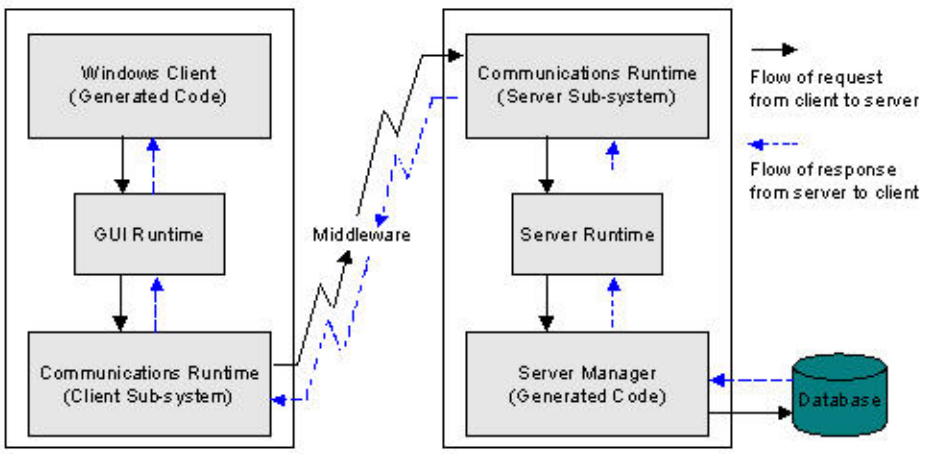
- For more information about deciding which section of the CFB is eligible to be encrypted, see Common Format Buffer (see page 21) in the appendix "Glossary of Distributed Processing Terminology."
- For more information about the user exits used by the DPC and DPS to encrypt and decrypt cooperative flow data, see "Distributed Processing User Exits (see page 131)."
- For more information about decrypting user exits available for use in these applications, see the *Distributed Processing—Client Manager User Guide* and *Distributed Processing—Communications Bridge User Guide*.

Chapter 4: Asynchronous Cooperative Flows

Synchronous Processing

A Distributed Processing application client (the DPC) executes a remote procedure (the DPS) as one atomic, blocked operation. The requesting application component cannot proceed with any other processing until the synchronous cooperative flow is complete and control is returned from the specified target server back to the client. A DPC expects that any request always with a corresponding response.

The following illustration shows the cooperative flow that takes place when a CA Gen GUI client issues a synchronous request to a Server Manager.



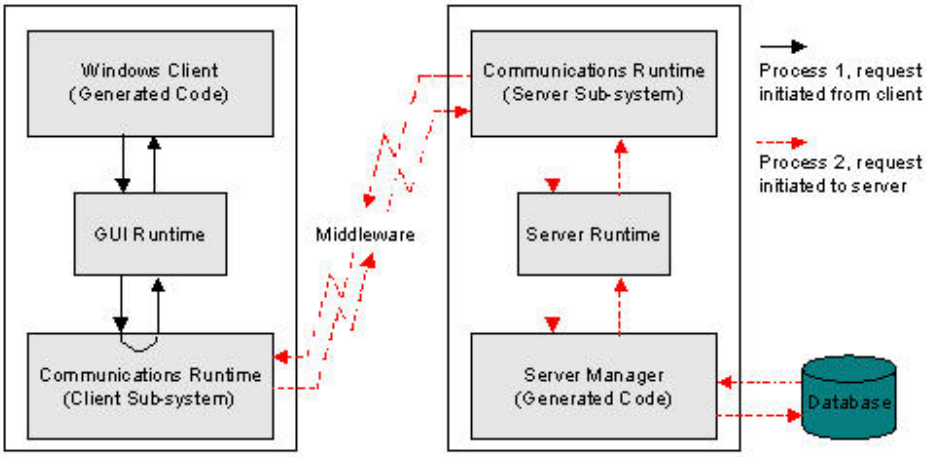
Synchronous cooperative flow is initiated using either a DialogFlow or a Procedure Step USE statement. The PStep USE is the preferred cooperative flow mechanism and is supported for both DPC-to-DPS and DPS-to-DPS requests.

Note: For more information about DialogFlow and Procedure Step USE, see the *Action Diagram User Guide*.

Asynchronous Processing with Generated Clients

In general, asynchronous processing is based on a similar concept of a cooperative flow (that is, request/response). However, asynchronous processing lets you decouple the handling of the request from the handling of the response. The effect of an asynchronous request is to initiate the server processing. The requesting application must then explicitly retrieve (or ignore) the associated asynchronous response of the request.

To allow the separation of request and response, an asynchronous cooperative flow involves several stages that are shown in the following illustration:



The request stage comprises the initiating request for the asynchronous flow (stage 1a) followed by a request to the server to process the request (stage 1b). The scope of the initiating request (stage 1a) depends the middleware being used, the middleware being the deciding factor on where the handover from stage 1a to stage 1b occurs. The retrieval stage (stage 2) comprises a request from the client to retrieve, or check the status of, the supplied response.

An asynchronous cooperative flow is initiated using a USE ASYNC statement. The options available with this command and the corresponding response commands are explained in the *Action Diagram User Guide*.

The processing of a USE ASYNC statement passes control to various runtime components and the supporting runtime code executes the code that actually initiates the flow to the target DPS. During the flow, the communications runtime code reaches a point in its processing where the request is considered accepted. This point can vary depending on the middleware environment being used. In most cases, the point of acceptance occurs when the communications code has sent the request to the target server environment. When the request is accepted, the runtime returns control to the initiating application, thus allowing processing to proceed. The supporting runtime continues to process the outstanding request and prepares to handle the associated response as directed by the initiating USE ASYNC statement.

A DPS application that is the target of an asynchronous cooperative flow behaves the same as if it were the target of a synchronous request. An exception is if the target DPS application is being invoked as a no response asynchronous request. In this case, the response buffer is not sent to the invoking DPC application.

Processing Options for an Asynchronous Request

When a DPC initiates an asynchronous cooperative flow, there are several options for retrieving the corresponding response. These options are specified as parameters when the request is initiated. The application component can choose the following options:

- Poll Response—in which the requesting application component can inquire if the response to a given outstanding request is available. After it is available, the response can be explicitly obtained by the application, thus completing the outstanding asynchronous request. Applications can implement either active or passive polling for a given response, where:
  - Active polling requires the application to request that control be returned to the application, even if the response is not immediately available. The application must either re-invoke the non-blocking request (GET ASYNC RESPONSE) to get the response periodically until the response becomes available, or use the check response statement (CHECK ASYNC RESPONSE).
  - Passive polling requires the application to request that the statement to get the response (GET ASYNC RESPONSE) should block until the response becomes available.
- Notify Event—This option is available only to windows clients, so is not available when performing server-to-server flows. The requesting application can request that an event handler be driven when the response to a given asynchronous request is available for processing. Because of the event action, the requesting application know that the response is available and can be obtained using the get response statement (GET ASYNC RESPONSE).

- No Response—The application can choose to only initiate a cooperative flow and not be concerned about receiving its corresponding response. The use of No Response frees the application from having to explicitly obtain or explicitly ignore the target server's response.

  If the requesting application decides it no longer requires a response to an outstanding request, the response should be explicitly ignored for the asynchronous cooperative flow to be considered complete. This is important, because only the resources that are associated with completed requests can be returned to the system. A response that is queued, awaiting either a GET ASYNC RESPONSE or an IGNORE ASYNC RESPONSE statement, is known as a pending response. After the request has been obtained (or ignored), an asynchronous cooperative flow is considered complete.

Note: For detailed information about the set of asynchronous action language statements, see the *Action Diagram User Guide*.

Asynchronous Processing with CA Gen Proxies

Using the CA Gen proxies, a user-written client application can make asynchronous requests of a target server. The set of exposed proxy interfaces includes APIs that provide a user-written client application with the ability to communicate with a target DPS in an asynchronous fashion.

Similar to the action language described for generated client applications, asynchronous support for a proxy influences only the handling of a request's corresponding response. Initiating a request to a target DPS is no longer synchronously linked with obtaining its response. After a request is accepted for processing by the CA Gen Proxy Runtime, the client application is able to engage in other processing. The server processing proceeds asynchronously to the client processing. The client application is able to obtain (or ignore) a given outstanding response when it is best suited for the application logic.

CA Gen supports four types of proxies, as described in the following list:

- The ActiveX/COM Proxy is a generated COM object that provides an automation interface to CA Gen servers. Providing a COM automation interface lets web applications with Active Server Pages and any OLE-enabled client access to CA Gen servers.

  A set of common properties and methods supports Asynchronous behavior in the generated COM Proxy interface and the COM Proxy Runtime. A client application that wishes to take advantage of a given COM Proxy's asynchronous processing must invoke its exposed asynchronous interface method. For more information about generating and using an Active X/COM Proxy, see the *Distributed Processing – Proxy User Guide.*

- The Java Proxy is a Java-based interface that lets Java applets and Java applications access to CA Gen servers.

  A set of common properties and methods supports Asynchronous behavior in the generated Client Java Bean and Application Java Bean. An applet or application that wants to take advantage of the asynchronous processing ability of a Java Proxy must invoke the exposed asynchronous interface methods of the bean. For more information about generating and using a Java Proxy, see the *Distributed Processing-Proxy User Guide*.

- The .NET Proxy is a generated .NET object that provides an object-oriented interface to the CA Gen servers. The .NET Proxy and associated the CA Gen runtimes are 100 percent managed .NET code. Providing a .NET managed object interface lets any .NET aware application access to CA Gen servers. Typical user applications would be stand-alone programs (VB.NET or C#) and ASP.NET web applications.

  A set of common properties and methods supports Asynchronous behavior in the generated .NET object. A .NET aware application that wishes to take advantage of a given .NET Proxy's asynchronous processing, must invoke the respective object's exposed asynchronous interface methods. For more information about generating and using a .NET Proxy, see the *Distributed Processing – Proxy User Guide.*

- The C Proxy API is a generated C language header file that provides you with the flexibility to call CA Gen server components from a C language application, or applications that support a C language interface.

  The C Proxy Runtime contains a set of APIs that support asynchronous behavior. A client application wishing to use asynchronous processing must use the specific asynchronous function calls, as described in the *Distributed Processing—Proxy User Guide.*

Chapter 5: Overriding Communications Support at Execution Time

Generation Time Configuration

To designate which communications support is used to process a specific cooperative flow occurs when the client or proxy is generated.

Review the discussion in the Communications Runtime section of the chapter "Anatomy of a DP Application (see page 25)." It is a target DPS's execution environment that dictates which transport protocol a given client can use to facilitate the processing of a cooperative flow to a given target server. At Generation Time, the communications type of a specific cooperative flow determines the Server Environment settings that are associated with the Server Manager that contains the target DPS.

The communication type that can be designated for a Server Manager corresponds to one of the supported Communications Runtimes. The following list includes communication types that can be designated for one or more Server Manager execution environments:

- CA Gen
- JavaRMI
- NET Remoting
- MQSeries
- ECI
- TCP/IP
- Tuxedo

Note: CA Gen in the preceding list refers to the CA Gen Client Manager.

In addition to the communications type, any pertinent arguments that are required by the associated communications runtime can be included as part of the Server Environment settings that are associated with each Server Manager. In addition to designating the communications type, a CA Gen model contains a unique set of communications attributes that can be set for each Server Manager and its associated Procedure Steps.

A CA Gen application developer can use the Toolset or CSE Properties dialogs to specify the communications attributes for each transport.

Example:

- Host Name and Port Number can be set for TCP/IP
- Queue Manager and Queue Names can be set for MQSeries
- Name Manager URL and Initial Factory can be set for EJBs
- Tuxedo Service Name can be set for Tuxedo

The communications type and communication attributes to be used for a given flow is taken from the settings that are contained in the model and captured in generated code at the moment in time when a client application or proxy is generated. The current communication settings that are associated with a target DPS are provided to the generator. The resulting generated code contains the communications type and attribute values to be used at runtime when the client flows to the target DPS.

When the generated code is built, the communications type and associated attributes are built into the resulting application. The Generation Time Configuration provides the basis of how the application was designed to run by the application developer. The application developer generates an application to execute in a particular target execution environment.

CA Gen provides facilities to override the Generation Time Configuration at execution time. The override to the generation time configuration takes place in the client application early in the processing of a cooperative flow. The execution-time configuration is described in the following section. The circumstances in which using execution-time configuration is not possible appear in the following list:

- A client application that is generated to use a communications type of Tuxedo
- Server-to-Server flows that involve CICS servers
- Server-to-Server flows that involve IMS servers
- Server-to-Server flows that involve MQSeries Servers
- Server-to-Server flows that involve Tuxedo

Execution Time Configuration

For designating which communications type a client use to process a cooperative flow occurs when the client application initiates a flow to a target server. The ability to override the communication type at execution time eliminates the need to re-generate the client application to switch the type of transport to be used to process a given cooperative flow request.

In addition to the communication type, CA Gen lets the attributes that are associated with a selected communications type be set (or overridden) at execution time. The following bullets show two opportunities to influence the values of the communication attributes:

- An external configuration file that is known as the Comm Config file
- User exits invoked from each of the various Communication Runtimes

Proxies offer an API to override their Generation Time Configuration. A developer of a user-written application that use a generated CA Gen proxy can override the communications properties of the proxy by using the generated comCfg interface for that proxy.

Note: For more information about how to override a proxy communication configuration, see the *Distributed Processing—Proxy User Guide*.

Comm Config Files

Each client runtime that is provided by CA Gen accesses an external configuration file as part of its processing of a cooperative flow. This external file provides the user of a client application the opportunity to configure its communication processing at execution time. This external file is known as the commcfg (Comm Config) file.

Each of the CA Gen client runtimes uses the specific commcfg file that are as follows:

Client Runtime	commcfg File Name
MFC GUI Runtime	commcfg.ini
COM Proxy	commcfg.ini
C Proxy	commcfg.ini
Java Web Generation	commcfg.properties
Java Web View	commcfg.properties
Java Proxy	commcfg.properties
ASP .NET	commcfg.txt
.NET Proxy	commcfg.txt

About server-to-server flows, certain server execution environments support the use of the commcfg runtime override. In these cases, the server runtime uses the following commcfg file:

Server Runtime	commcfg file name
EJB Runtime	commcfg.properties
.NET Servers	commcfg.txt

Each commcfg file provides the means of controlling certain runtime behavior including:

- Transaction routing
- Enabling and disabling trace activity
- How the runtime cache the contents of the commcfg file

The CA Gen runtime code uses the following types of commcfg files, see the appendix "Comm Config Files (see page 153)":

- commcfg.ini
- commcfg.properties
- commcfg.txt

Transaction Routing

Transactions can be routed to a specified server execution environment by adding one or more TRANCODE lines to the commcfg file. Each TRANCODE line assigns a transaction code to a specific server execution environment.

- Transaction codes are specified using their full name or partial name. Partial names are designated using a trailing "*" (asterisk). A single asterisk indicates all transactions. For example:
  - ABCD is a fully qualified transaction name.
  - ABC* use a wildcard to indicate that transaction routing applies to all transaction codes that begin with the letters "ABC."
  - A* use a wildcard to indicate that transaction routing applies to all transactions codes that begin with the letter "A."

    * the wildcard that indicates transaction routing applies to all transactions.
- The arguments defining a destination for a given TRANCODE entry vary by communications type. Each communications type requires its own unique set of arguments. These arguments identify the target server execution environment. For example:
  - Host Name and Port Number for TCP/IP
  - Queue Manager and Queue Names for MQSeries

For a sample listing of the three types of commcfg files, see the appendix "Comm Config Files (see page 153)." The syntax that is used within each commcfg file type is followed when adding TRANCODE entries. The comments section of each sample commcfg file describes the format for each of the communication types offered and supported by the execution environments that use the given commcfg file (commcfg.ini, commcfg.properties, commcfg.txt).To determine which runtime uses which commcfg file, see Comm Config Files (see page 153).

The following format of a TRANCODE line is taken from a commcfg.ini file. The use of this file is intended to illustrate the general format of the TRANCODE line. The format of the TRANCODE line for the other files can be found in their respective files that are located in the "Comm Config Files" appendix. The format of a TRANCODE line in commcfg.ini is as follows:

```
TRANCODE COMMTYPE arguments
```

where TRANCODE and COMMTYPE are required and arguments depends on the value that is specified for COMMTYPE.

The following formats designate the type TRANCODE lines that can be entered in a commcfg.ini file:

where <...> and {...} are user supplied fields such that <...> are required fields, and {...} are optional.

<TRANCODE> TCP <host> <service/port>

<TRANCODE> ITP <host> <service/port>

<TRANCODE> MQS <Queue Name> <Queue Manager> {Reply Queue Name}

<TRANCODE> MQSC <Queue Name> <Queue Manager> {Reply Queue Name}

<TRANCODE> ECI {CICS system name}

<TRANCODE> CM

<TRANCODE> EJBRMI {Initial Factory Class}{Name Manager URL}

<TRANCODE> WS <baseURL> <contextType>

<TRANCODE> NET <hostName> <portNumber> <protocolCode>

Note: ITP is a version of the CA Gen TCP/IP cooperative flow-processing product that services flows to the CA Gen IMS TCP/IP Direct Connect product.

Some TRANCODE lines that could be coded in a commcfg.ini file are as follows:

```
ABCD ECI CICS22A

ABC* TCP myhost2 2008

A*   CM

*    TCP myhost4 2050
```

This illustration would route an exact trancode match of ABCD to the CICS region identified as CICS22A as defined within IBM Universal Client software. Transactions using ABCD as their trancode would use ECI as its transport mechanism. Trancodes starting with ABC (but not ABCD) would be routed to myhost2, port 2008 using TCP/IP as its transport mechanism. The Client Manager processes Trancodes that begin with A (but not ABC). All others trancodes is routed to myhost4, port 2050 using TCP/IP as its transport mechanism.

Enabling and Disabling Trace Activity

The commcfg file can be used to enable and disable tracing within the client runtimes. The CMIDEBUG statement is used to enable and disable the tracing. Usually CMIDEBUG controls tracing of the communications runtimes that is provided by CA Gen. The following CMIDEBUG statement turns on or turns off tracing activities respectively:

```
CMIDEBUG ON
CMIDEBUG OFF
```

The CMIDEBUG statement can appear multiple times within the file, but the last one encountered the setting that is in effect.

For a sample listing of the three types of commcfg files, see the appendix "Comm Config Files (see page 153)." The syntax that is used within each commcfg file type is followed when adding a CMIDEBUG statement. The comments that are provided in each commcfg file document the syntax to be used within that file. Also, the comments identify the location of any resulting trace file that is created as a result of adding a CMIDEBUG ON statement to the commcfg file. One of the CA Gen client runtimes uses the commcfg file.

Caching the Contents of the commcfg File

The commcfg file itself can be used to indicate to each client runtime how often the runtime should re-read the commcfg. To improve performance, each runtime creats a cache of the commcfg file. By default, caching is not performed and the file is re-read and re-parsed for each flow that is processed by the client process. Re-reading the commcfg for each flow lets any file changes be applied and used for the processing of the next flow that is serviced by the runtime. This default behavior cannot be ideal in all environments. Therefore, the CACHETIMEOUT statement can be used to control the number of seconds to wait between re-reading of the commcfg file. All flows that occur between the read and the timeout use the values that are located in the cached version of the commcfg file.

Note: Rereading of the commcfg takes place after the first flow that occurs after the specified timeout value has expired.

The following CACHETIMEOUT statements set the cache timeout value to 0 (the default behavior), 3 minutes, and never:

```
CACHETIMEOUT   0
CACHETIMEOUT   180
CACHETIMEOUT   NEVER
```

The specified cache timeout only applies to the client process from which a cooperative flow is initiated. If the client process terminates, the cache that is associated with that process goes away. The NEVER setting requires the client process be stopped and the communications runtime be unloaded before the commcfg file is re-read.

For a sample listing of the three types of commcfg files, see Comm Config Files (see page 153). The syntax that is used within each commcfg file type is followed when adding a CACHETIMEOUT statement. The comments that are provided in each commcfg file document the syntax to be used within that file.

Runtimes User Exits

To influence how a cooperative flow will be processed by a selected communication runtime is provided by a user exit invoked by that runtime.

User exits are invoked throughout the processing of a CA Gen Distributed Processing application. User exits are invoked from within the various client runtimes, communications runtimes, and server runtimes.

For information about CA Gen user exits, see the appendix "Distributed Processing User Exits (see page 131)." This appendix addresses the exits that are encountered during the processing of a CA Gen Distributed Processing application.

Appendix A: Glossary of Distributed Processing Terminology

This glossary defines terminology and acronyms commonly used when discussing CA Gen Distributed Processing applications. Additionally, this glossary contains terms and acronyms specific to those environments in which CA Gen Distributed Processing applications execute.

This compilation of terms and acronyms comes from many sources, including the vendors of third-party software. Some terms are specific to the vendor's products while others have a common usage. This glossary presents terms and acronyms grouped in sections that are associated with the context to which they would typically belong, as shown in the following list:

- General Distributed Processing
- CA Gen Distributed Processing
- CICS
- IMS
- EJB
- Tuxedo
- MQSeries
- .NET
- TCP/IP
- Web Services
- SNA
- ECI
- Java
- COM
- XML
- HP NonStop (Pathway, TMF, and GAEF)

General Distributed Processing Terminology

ACL—see Access Control List

Access Control List (ACL)

A list used by administration software for authorizing and validating access to controlled resources.

agent

A software module that provides a programmatic interface to the processing of a controlling process. In a client/server application the agent is responsible for preparing and exchanging information on behalf of either a client application or server application.

API—see Application Programming Interface

Application Programming Interface (API)

A set of related programmatic function definitions, exposing calling convention and format, used to facilitate access to software not directly implemented in the program.

application protocol

A mutually understood, predefined, sequence of messages exchanged between two application entities.

application resource

Data accessed using a Resource Manager.

asynchronous communication

A method of communication used between application entities in which the sending entity is able to proceed with its own processing without waiting for a reply to its request message. Contrast with synchronous communications.

atomicity

The "all-or-nothing" property of a transaction that insures that all of the transaction aware resources participating in the transaction either commit their associated changes or roll them back to their state prior to beginning the transaction.

authentication

A security mechanism used by a system to ensure that an application component has a valid identity.

authorization

A security mechanism, used by a system, to permit or deny access to a resource controlled by the system. See Access Control List.

blocking condition

A condition encountered in an application resulting in that application halting execution until something happens, such as the completion of a send operation, a response received, a timer pop, and so on.

bracket

A programmatic demarcation used to indicate the beginning and end of a transaction.

channel

A path established between two communicating end points over which communications transmission can take place.

Client

A software module that gathers and presents data to an application; it requests services and receives replies. This term can also be used to indicate the requesting role that a server module assumes when it requests services of another server module (see server-to-server flows).

client/server

A request/response programming style characterized by the application being divided into two or more cooperating components. The cooperating components are known as clients and servers. Clients request work of servers. In some cases a server takes on the role of a client when it requests the services of another server module (see server-to-server flows).

commit

A software action associated with making changes to associated resources permanent. Contrast with rollback. See two-phase commit.

daemon

A process that is actively running and does its processing in the background.

data-dependent routing

A process of selecting a server based on the contents of data contained in the request message being processed.

data independence

A characteristic of software processing that allows data to be accessed and updated without knowing the underlying storage format or relative structure of that data.

data marshalling

The process of rendering arguments and data values into a byte-stream format and packaging that data into a buffer so that it is suitable for transmitting between heterogeneous nodes connected using a communications network.

data unmarshalling

The process of rendering data to a node's native data representation. Typically, the data would have been previously marshalled into a format suitable for transmitting over a network. (see data marshalling).

decryption

The process of rendering an encrypted message into an intelligible format. Contrast with encryption.

deferred communications

Software processing that two modules communicate such that both applications need not be active simultaneously. Typically, deferred communications are performed using message queues. Also see asynchronous processing.

Directory Service

Software processing that maps names to locations.

distributed application, distributed computing

Software processing characterized by an application being composed of two or more software components that are physically located on more than one computer. Also see client/server.

domain

A component part of an architectural hierarchy. Typically, the domain is associated with an administrative partition of a complex distributed system, name space, or application.

durability

The property of a transaction that guarantees the permanence of modified data upon the completion of commit processing.

dynamic configuration

The ability to change the definition of an application while that application is active.

encryption

The process of converting a message rendering into a form where its contents are unintelligible to unauthorized parties. Contrast with decryption.

event

The occurrence of a condition, state change, or activity, of interest to one or more software components.

event notification

The process of signaling one or more software components that an event has taken place.

fault tolerance

A characteristic such that an application continues to run despite failures in its software components or communication interfaces.

forwarding

A processing technique by which pipeline parallelism is implemented, where a message is sent on to another service that will complete the work and reply to the original requesting application.

function shipping

A processing technique whereby the processing of an application is segmented such that functions are located near the data they need to access. Requests to access data are sent to the associated function and only the results of the processing are sent back to the requestor.

gateway

A software module capable of exchanging information between two different application environments and or networks.

global transaction

A transaction, which spans more than one Resource Manager.

heterogeneous computer platforms

A collection of computers utilizing different data formats, different operating systems, different internal data representation (byte ordering), and so on.

implementation transparency

A method whereby a requestor of a service need not be aware of how the service processing actually occurs, letting re-implementation of the service without impacting the requestor.

IDL—see Interface Definition Language

Interface Definition Language (IDL)

A language used to describe data types and functions, typically used in an implementation of a Remote Procedure call (RPC).

IPC - see Inter-Process Communications

Inter-Process Communication (IPC)

Various system provided mechanisms by which modules in separate address spaces (processes) communicate within a single computer. Examples of IPCs include shared memory, semaphores, message queues, mapped memory files, mailslots, domain sockets, atoms, and so on.

isolation

The behavior of a transaction that insures that concurrent operations on the underlying data do not interfere with each other.

load balancing

A software process that assigns inbound service requests to instances of servers capable of processing the request service, in a manner to optimize throughput and response time.

local client

A client application that executes on the same physical computer as the server application that has been selected to process a given cooperative request.

local transaction

A transaction, which is only active and known to a single Resource Manager.

location transparency

A software characteristic that lets a resource, or service, be identified in such a way that its name implies no particular network address or physical location.

message

One or more packets of data communicated between one or more programs.

middleware

Connectivity software that lets components of an application inter-operate across a network, despite differences in underlying communications protocols, system architectures, operating systems, databases, and other application services. Portions of middleware address communications while other portions provide an environment in which applications execute. Middleware can take on the following different forms:

- Transaction processing (TP) monitors which provide tools and an environment for deploying distributed applications.
- Remote Procedure Call (RPC) is an API that causes program logic located on the network to be executed as if it were a call to a local program.
- Message-Oriented Middleware (MOM) provides program-to-program data exchange between components of a distributed application. MOM supports synchronous and asynchronous operations. Messages exchanged using a MOM must be interpret and processed by the application receiving the message data.
- Object Request Brokers (ORBs) are object-oriented middleware that enable the objects that comprise an application to be distributed and shared across heterogeneous networks.

peer-to-peer

A cooperative processing environment where two communicating entities have similar processing capabilities.

phase one

The first phase of a two-phase commit. In this phase the resources modified during the transactions are prepared for phase two. Phase one culminates with the recording of the transactions decision to a log. This phase is under the control of the Transaction Manager. The Transaction Manager sends a prepare notice to the involved Resource Manager(s). The Resource Manager(s) write their updates to stable storage, and respond with an indication that it is prepared to commit or does not wish to commit.

phase two

The second phase of a two-phase commit which announces the transaction decision to the Resource Manager(s) involved in the transaction, triggering the Resource Manager(s) to release locks and making the results of the transaction visible to others desiring access to the managed resource(s).

post an event

A software operation that indicates that a processing condition of interest has occurred.

presentation service

A software layer that converts data being transmitted between heterogeneous computers. Presentation Service is usually associated with client applications allowing end user interaction with an associated software application or system.

proxy

A software component that exposes a simplified programmatic interface to a specific server application. A proxy presents a consistent, well-defined interface, used by other software modules to facilitate communications between the user of the proxy and its associated application. Using the interface exposed by a proxy helps to isolate and insulate its user from having to know explicit details of the protocol required to communicate with the associated application.

queue

A data structure that holds messages that will be processed by a software procedure.

recovery

Software processing that attempts to restore an application to a known state after a transaction, server, network, or computer has failed.

Remote Procedure Call (RPC)

A software programming technique that causes the execution of a non-local program to execute as if it were a call to a local program.

request buffer

A data buffer containing a request for a service originating from a client.

response buffer

A data buffer containing the results of a request for service.

request/response

A software processing style characterized by clients sending request buffers to obtain service from server modules. The requests result in the server modules replying in the form of a response buffer. Both the requesting clients and responding servers must comprehend their respective rolls to adhere to the application protocol.

Resource Manager (RM)

An administrative software component that maintains the state of application resources (the most common being databases). Most DBMS software assumes that the Resource Manager has responsibility for those databases it manages.

rollback

The activity that restores an application resource to its state prior to the initiation of a transaction involved in modifying the resource. The Resource Manager owning the modified application resource carries out-processing a rollback. In cases involving two-phase commit processing, the rollback of activities associated with a transaction are under the control of the Transaction Manager.

RPC—see Remote Procedure Call

self-describing buffer

A buffer that contains a description of its own format and contents; a message format in which the data type(s) and length(s) are known or can be determined without additional external information.

server

A software module, made up of a set of well-defined inputs, outputs and processing, capable of accepting requests from clients, proxies or other servers. In a request/response style of distributed processing, after a server completes its requested processing, the server returns a reply to the requesting client.

service

The generic name given to application processing available for request by a client, characterized as having well defined inputs, outputs, and processing. Servers provide one or more services to requesting clients.

stateless interaction

The interaction between a client and server where the service neither uses nor maintains any extra information about the client to process its request; all the data necessary to complete the work is in the client's request buffer. Stateless interactions are usually associated with the request/response style of distributed processing.

stateful interaction

The interaction between a client and server where the server keeps track of the processing progress for a given client interaction. Typically, the client would have more than one message exchange with the server to complete its required processing. Stateful interactions are usually associated with a conversation style distributed processing.

synchronous communication

A style of communication between application entities in which the sending entity waits for a reply to its message before resuming its own processing. Contrast with asynchronous processing.

time-independent communications

A style of communications whereby two software module communicate, however, both do not have to be active simultaneously. Typically deferred communications are performed using message queues. This description does not make any sense. Looks like you can have copied the description of "deferred communications."

thread

An execution unit that is independently dispatched or is the embodiment of a unique execution context. A thread is comprised of a uniquely identifiable execution sequence and the memory it manipulates. One or more threads run within the context of a process.

time-out

An event that occurs when processing takes longer than expected or configured. Time-out events are often used to satisfy blocked operations.

transaction

(1) a bracketed unit-of-work, comprised of one or more software requests, such that all operations must complete to consider the transaction complete.
(2) a discrete operation, dispatched on request from an initiating client.

transaction initiator

An application software component that begins a transaction.

transaction terminator

An application software component that ends a transaction.

Transaction Manager (TM)

An administrative software package that manages global transactions across multiple computers or Resource Managers.

Transaction Processing Monitor (TPM)

A service broker that uses a transaction request to dispatch associated services. The TPM interacts with the Transaction Manager and Resource Managers providing a framework of control and administration over the server-side application runtime environment.

Two-Phase Commit (2PC)

A software protocol used by a Transaction Manager to coordinate the commit processing of a Resource Manager.

unit-of-work

A collection of one or more transactions that are considered autonomous. The unit-of-work is considered to be complete only with the sum of its parts has been completed.

XA Interface

An X/Open standard defining the protocol for communicating between an external Transaction Manager and a Resource Manager.

2PC—see Two-Phase Commit

Terminology Unique to CA Gen Distributed Processing

action

A single Action Diagram statement.

action block

A stand-alone action diagram that other diagrams use. It defines the logic of an algorithm or specifies logic that is common to many action diagrams.

Action Diagram (AD)

A set of view definitions and procedural instructions that use the language and constructs provided by CA Gen for the purpose of implementing a procedure step, elementary process, or action block. An Action Diagram is an ordered collection of actions, which define the logic of an elementary process, procedure step, or action block.

ActiveX\COM Proxy—see COM Proxy

AD—see Action Diagram

AEF—see Application Execution Facility

AEFAD—see Application Execution Facility Asynchronous Daemon

AEFC—see Application Execution Facility Client

AEFUF—see Application Execution Facility User Funnel

Application Execution Facility (AEF)

A CA Gen application environment that supports interactively loading and execution of generated applications in a non-MVS (CICS, IMS or TSO) environment.

Application Execution Facility Client (AEFC)

A CA Gen application that provides presentation of IBM's 3270 data streams. AEFC is a 3270 emulator that interfaces with ASCII terminals. The ASCII terminals can either be directly connected to a host system or provided using a TCP/IP Telnet ASCII terminal emulator. The AEFC software handles the presentation of 3270 messages sent to or received from a CA Gen block mode application.

Application Execution Facility Asynchronous Daemon (AEFAD)

A CA Gen application that provides a facility to interactively load and execute generated applications. The AEFAD supports the execution of block mode and Distributed Processing Server (DPS) applications. The AEFAD is a TCP/IP socket application. It does not spawn a unique process for each connected user, rather it manages the messages coming and going to all connected users. The AEFAD multiplexes messages to/from multiple clients in an independent and concurrent manner.

Application Execution Facility User Funnel (AEFUF)

A CA Gen application that multiplexes AEFC and AEFAD connections onto one TCP/IP connection. AEFUF manages many client connections providing a critical scaling component funneling many user messages to a target AEFAD environment. Distributed Processing Client (DPC) applications must connect to an AEFUF.

CFB—see Common Format Buffer

Client Manager (CM)

A workstation application, offered by CA Gen, to service cooperative flow requests initiated by one or more CA Gen generated GUI applications operating on the desktop.

Common Format Buffer (CFB)

A dynamically constructed, in memory storage area use by CA Gen runtime during the processing of a cooperative flow. The CFB is an encoded buffer used to exchange the import and export views between a Distributed Processing Client (DPC) and Distributed Processing Server (DPS). The CFB is designed to provide a platform independent byte stream transferred between a DPC and a DPS. The CA Gen runtime will make use of a CFB based on the application's execution environment and selected transport type.

Communications Interface (COMMINF)

A set of "internal use" API's and associated dynamically loaded implementations, used by CA Gen runtime, to process a cooperative flow between a Distributed Processing Client (DPC) and Distributed Processing Server (DPS). Collectively COMMIF refers to the set of CoopFlow and MsgObj implementations offered by CA Gen.

Communications Interface (CIF)

A dynamically loaded software layer used by CA Gen Client Manager and Communications Bridge to handle the outbound communication with a target server environment. The Client Manager provides CPI/C (LU 6.2) and TCP/IP support, while the Communications Bridge provides CPI/C (LU 6.2), CICS ECI and TCP/IP.

Communications Bridge (Comm. Bridge)

A gateway application offered by CA Gen to concurrently service cooperative flow requests initiated from a variety of CA Gen distributed Processing Client (DPC) applications that are executing on one or more client workstations.

Common System Utilities (CSU)

A CA Gen runtime layer that exposes a common interface to a set of platform specific operations. For example, the operations provided within the CSU layer include tracing, locking, and dynamic loading of other runtime libraries. CSU is implemented and used within CA Gen's C/C++, Java, and .NET runtimes.

COM Proxy

(1) A CA Gen product that supports the generation of a COM object that can be used by non-CA Gen applications to flow to a procedure step that is packaged within a CA Gen generated Distributed Processing Server (DSP) application. (2) The generated COM code that provides a COM automation interface to a specific server procedure step. A COM proxy can be used by Web applications with Active Server Pages or any OLE-enabled client to access CA Gen generated Distributed Processing Server (DSP) applications.

cooperative processing

A term referring to client/server applications and their packaging. A cooperative application has a client component and a server component that work together across platforms. Cooperative processing is a synonym for Distributed Processing. See Distributed Processing (DP) Application.

cooperative flow

A type of flow used within CA Gen applications that facilitate the remote execution of a Distributed Processing Server (DPS) application. Cooperative flows result from modeling a dialog flow or PStep Use that targets a PStep that is packaged in a separately packaged server manager.

CoopFlow (Cooperative Flow)

A component of CA Gen's ODC architecture. CoopFlow is a generic term used to identify the software load modules that implement the various transport mechanisms supported within CA Gen (TCP/IP, MQSeries, Client Manager, ECI, Web Services, Tuxedo, JavaRMI). See Open Distributed Computing (ODC).

CSU—see Common System Utilities

dialog flows

A flow control technique used to transfer control, and possibly data, between procedure steps in the same CA Gen business system (internal flows) or between procedure steps within a different business system (external flows). There are two types of dialog flows; links and transfers.

Distributed Processing (DP) Application

A term used to describe a type of client/server application characterized by its processing being divided into two or more separate application parts. Each application part has a specific role in the overall application. The execution of the application is "distributed" amongst its various application parts. Some parts serve as Distributed Processing Clients (DPC), others serve as Distributed Processing Servers (DPS). Each application part is implemented in a procedure step and is physically packaged into a separate load module. In most cases each load module making up the distributed processing application is physically located on more than one computer.

Distributed Processing Client (DPC)

A generic term that describes the Distributed Processing application component that makes a request of the Distributed Processing Server (DPS) component. CA Gen supports many types of DPC applications (GUI Clients, Web Clients, hand written applications by way of a Gen Proxy Client, Distributed Processing Server (DPS)).

Distributed Processing Server (DPS)

A generic term that describes the generated Distributed Processing application component that processes cooperative flow requests initiated from Distributed Processing Client (DPC) applications. CA Gen supports many target environments capable of hosting a generated DPS application (for example, CICS, IMS, EJB, MQSeries, DCE, Tuxedo, Transaction Enabler).

DPC—see Distributed Processing Client

DPS—see Distributed Processing Server

elementary process

The smallest unit of processing activity that has meaning to a user such that when it is complete it leaves the associated business processing in a consistent state. For every elementary process, there is an associated action diagram.

export view

A type of view, within a distributed processing application, that contains the collection of attributes that are returned from an execution of a Distributed Processing Server (DPS).

Graphical User Interface (GUI)

A graphics based user interface that incorporates icons, pull-down menus and a mouse. GUI's have become the preferred way users interface with most computer systems. Some major GUI styles include Windows, Macintosh, Motif, and Web browser presentation of HTML. CA Gen supports Windows and Web Browser as graphics based user interfaces.

GUI—see Graphical User Interface

GUI client

A type of generated CA Gen Distributed Processing Client (DPC) application that operates under the control of a generated Window Manager. This type of application contains one or more cooperative flow requests that target a procedure step that are packaged in one or more generated Distributed Processing Server (DPS) applications.

import view

A view, within a distributed processing application, that contains the collection of attributes that are provided as input to an execution of a Distributed Processing Server (DPS).

Java Proxy

(1) A CA Gen product that supports the generation of two Java-based interfaces that can be use by non-CA Gen Java applets, applications, servlets, and EJB's to flow to a procedure step that is packaged within a CA Gen generated Distributed Processing Server (DSP) application. (2) The generated Java-based interface code that provides the Java interface to a specific server procedure step.

.jvf file

A Tuxedo Jolt file containing the CA Gen view definitions to be processed by its associated generated Tuxedo server. This file is the Java equivalent to the .tvf file used by the generated C Tuxedo servers.

Load Module

(1) a type of software file that is capable of being brought into, or "loaded" into, a computer's memory area. Load modules can be executed as programs. Others contain data. Executable load modules consist of an assembly of machine instructions and implement some logic. (2) In CA Gen, load module is a general term to indicate one or more procedure steps combined (packaged) into an identifiable unit. After being generated and installed, a load module is an executable file. The process for defining the packaging is known as load module packaging, or packaging for short.

Message Object (MsgObj)

A component of CA Gen's ODC architecture. The MsgObj is responsible for creating and parsing the message buffer being exchanged as part of the cooperative flow. MsgObj is a generic term used to identify the software load modules that implement the various types of data representation techniques employed as part of a cooperative flow (CA Gen's CFB, Tuxedo's View32,). See Open Distributed Computing (ODC).

MsgObj—see Message Object

Multi-Instance Client Manager

A user-modified version of CA Gen's Client Manager application. The user modification is in the form of a user exit. In a multi-user system, such as Windows 2000 with Terminal Service enabled, the user exit lets a customer implement a technique that distinguishes one user from another. Typically, the implementation techniques make use of the logon user-id or session-id associated with a specific instance of a user. After the Multi-Instance Client Manager has been customized, multiple copies of the Client Manager application can execute with a multi-user system. In a multi-user system, the Multi-Instance Client Manager for a given user behaves the same as the single user version of the Client Manager.

Open Distributed Computing (ODC)

A CA Gen architecture used to externalize, and normalize, the processing by which CA Gen runtimes service cooperative flows. ODC exposes a common API. This API is used by the various runtimes responsible for processing cooperative flows. Using this normalized API lets the runtimes perform cooperative flows without concern for the underlying middleware or communications mechanism being employed to service cooperative flows.

ODC is an object-oriented implementation that separates the processing of a cooperative flow into two main components. These components are known as the MsgObj (Message Object) and the CoopFlow (cooperative flow). The MsgObj is responsible for creating and parsing the message buffer being exchanged as part of the cooperative flow. The CoopFlow is responsible for handling the communications between a DPC and DPS. CA Gen supports a variety of middleware and transport mechanisms. Each cooperative flow determines which MsgObj and CoopFlow will be used to process a flow request. CA Gen supplies different MsgObj and CoopFlow implementations to support the variety of supported middleware and transport mechanisms.

Procedure

A method by which one or more elementary processes is carried out using a specific implementation technique.

PStep—see Procedure Step

Procedure Step (PStep)

A subdivision of a procedure that performs a discrete and definable amount of work necessary to complete a procedure. Each procedure step is contained within a procedure. Procedures can be implemented using multiple PSteps.

proxy

A CA Gen supported coding interface that enables a non-CA Gen application to flow to a CA Gen Distributed Processing Server (DPS). Proxies are generated interfaces and runtimes. CA Gen can generate proxies for use in the following implementation technologies; C/C++ programs, applications capable of using COM objects, applications that use client JavaBeans, applications that use Application JavaBeans, and applications that use .NET.

runtime

A term used to describe those software components that are logically part of a CA Gen application but are not part of the application's generated code. Some of these components are physically linked to the application, other are dynamically loaded depending on the execution characteristics of the application (middleware runtime). Runtime can also refer to the execution environments that support the execution of the application (Transaction Enabler, Tuxedo, CICS, IMS, MQSeries).

server manager

A CA Gen non-window load module that contains one or more PSteps. A Server Manager PStep is the target processing of a cooperative flow request. Server Manager is a synonym for Distributed Processing Server (DPS).

server-to-server flow

A server-to-server flow is a cooperative flow that is initiated from within a PStep that is packaged as part of a Distributed Processing Server (DPS) application.

services table

A generated table that is part of the ODC architecture. The Services table contains transport specific data for each TranEntry record.

target server

A term that refers to the Distributed Processing Server (DPS) applications involved in a given cooperative flow. It refers to the server portion of a client/server request. Target server can be used to refer to the physical hardware hosting the DPS application.

TE—see Transaction Enabler

T I T D—see CA Gen's CICS MQSeries Transaction Dispatcher

TranEntry Table

A generated table that is part of the ODC architecture. The TranEntry contains entries for each cooperative PStep understood by the application component being generated. Each record of a TranEntry table holds information needed to accomplish one side of a cooperative flow. A DPC will have a TranEntry table that contains a record for each cooperative PStep the DPC can execute. A DPS can have two TranEntry Tables. One table hold TranEntry records for each PStep available for use by a DPC. The other TranEntry table contains entries if the DPS executes any server-to-server flows.

Transaction Enabler (TE)

A collection of software components offered by CA Gen that are collectively known as the Transaction Enabler. The TE is made up of the AEFC, AEFUF, AEFAD, and the Tuxedo Proxy Client.

trancode

Within CA Gen, trancode, is used by the Dialog Manager to manage flows (links, transfers, and external flows) between PSteps.

transport

A generic term that refers to the layer of software that interfaces to the physical network infrastructure.

Tuxedo Proxy Client

A TCP/IP software component offered by CA Gen that serves as a gateway application for cooperative flows targeting Distributed Processing Server (DPS) applications executing under the control of the Tuxedo TP monitor.

User Funnel (UF)

A TCP/IP software component offered by CA Gen that serves as a gateway application for cooperative flows targeting Distributed Processing Server (DPS) applications executing under the control of CA Gen's Asynchronous Daemon. See Application Execution Facility User Funnel (AEFUF).

view

A collection of associated attributes from one or more entity types that are input or output from a business process.

View Matching

A process used by a CA Gen user to map views of the source (action diagrams) to the import view of the destination (action diagram). View matching insures that the destination process or procedure (action diagram) has the data necessary to execute. It maps the export view of the action block to views in the source action diagram to return the results of the destination action block execution.

View Definition File (VDF)

A generated file that contains source code made up of data structures that externally describe the cooperative flows occurring within its associated distributed processing application component. A VDF file is created for each generated window manager (Distributed Processing Client (DPC)) and for each generated server manager (Distributed Processing Server (DPS)). The VDF file is compiled and linked into its respective application component. The VDF file contains a ViewDef table, TranEntry table(s) (for a DPS with server-to-server flows there are two TranEntry tables), and a Services table.

ViewDef Table

A table that is part of the ODC architecture. The ViewDef is a generated table that contains entries for each import view and export view associated with a component of a cooperative flow. The ViewDef is used by the ODC MsgObj to map the attribute values to and from the communications buffer and the associated view storage area within the associated distributed processing application component (either the DPC or DPS).

Window Manager

A generated CA Gen application that operates as a Windows GUI applications. Applications containing a window manager can be packaged as stand-alone Windows application or packaged as a cooperative application. For the latter, the application containing the window manager is capable of containing one or more cooperative flows. A window manager packaged as a cooperative application operates as the Distributed Processing Client (DPC) application when flowing to a remote PStep, where that PStep is located in a Distributed Processing Server (DPS).

.VDF—see View Definition File

CICS Terminology

AOR—see Application-owning region

Application-owning region (AOR)

A CICS system region that owns the transaction.

asynchronous processing

Asynchronous processing within CICS is an intercommunication function that lets a transaction executing on one CICS system start a transaction on another system. The two transactions execute independently of each other. Compare with *distributed transaction processing*.

ATI—see Automatic Transaction Initiation

Automatic transaction initiation (ATI)

The automatic act of initiating a CICS transaction by an internally generated request. For example, issuing an EXEC CICS START command or the reaching of a transient data trigger level. A transaction can be initiated immediately, at a specified time, after a specified time interval, or, if a terminal is required, as soon as that terminal is free.

auxiliary storage

Data storage other than main storage; for example, storage on magnetic tape or direct access devices.

auxiliary trace

An optional CICS function that causes trace entries to be recorded to the auxiliary trace data set. The auxiliary trace data set is a sequential data set on disk or tape.

CICS Complex (CICSplex)

(1) A set of interconnected CICS regions acting as Resource Managers, and are combined to provide a set of coherent services for a customer's business needs. In its simplest form, a CICSplex operates within a single MVS image. Within a Parallel Sysplex environment, a CICSplex can be configured across all the MVS images in the sysplex. The CICS regions in the CICSplex are generally linked through the CICS inter-region communication (IRC) facility, using either the XM or IRC access method (between regions in the same MVS image), or the XCF/MRO access method (between regions in different MVS images). (2) The largest set of CICS regions or systems to be manipulated by a single CICSPlex SM entity.

CICSplex—see CICS Complex

CICSPlex SM—see CICSPlex System Manager

CICSPlex System Manager (CICSPlex SM)

A system-management product that provides a single-system image and a single point of control for one or more CICSplexes.

CICS region userid

A userid assigned to a CICS region at CICS initialization. It is specified either in the RACF started procedures table when CICS is started as a started task, or on the USER parameter of the JOB statement when CICS is started as a job.

CICS-attachment facility

Part of the CICS product that provides a multithread connection to DB2 to let applications running under CICS execute DB2 commands.

CKTI

An MQSeries supplied CICS transaction that monitors an MQSeries initiator queue. CKTI starts a CICS transaction when an MQ event occurs on that queue.

COMMAREA—see Communication Area

Communication area (COMMAREA)

A CICS storage area used to pass data between tasks that communicate with a given terminal. The area can also be used to pass data between programs within a task.

connection

Defines a remote system with which your CICS system communicates using either intersystem communication or multi-region operation (MRO).

DOR—see Data-Owning Region

Data-Owning Region (DOR)

A CICS system region that owns the data files or databases. DORs are similar to file-owning region.

Distributed Program Link (DPL)

A programming mechanism that lets a CICS client program call a server program running in a remote CICS region, and pass and receive data using a COMMAREA.

Distributed Transaction Processing

A type of intercommunication in CICS, in which the processing is distributed between transactions that communicate synchronously with one another over intersystem or inter-region links.

DPL—see Distributed Program Link

DTP—see Distributed Transaction Processing

EDF—see Execution Diagnostic Facility

EIB—see EXEC Interface Block

EXEC

A keyword used in CICS command language. All CICS commands begin with the keywords EXEC CICS.

EXEC Interface block (EIB)

A control block associated with each task in a CICS command-level environment. The EIB contains information that is useful during the execution of an application program (such as the transaction identifiers) and information that is helpful when a dump is being used to debug a program.

Execution Diagnostic Facility

A CICS facility used for testing application programs interactively online, without making modifications to the source program or to the program preparation procedure. The facility intercepts execution of the program at various points and displays information about the program at these points. Also displayed are any screens sent by the user program, so that the programmer can converse with the application program during testing just as a user would do on the production system. There are two CICS-supplied transactions used to invoke this facility depending on application type: (1) CEDF is used for applications executing at a terminal, (2) CEDX is used for non-terminal tasks.

ECI—see External Call Interface

External Call Interface (ECI)

An API that lets a non-CICS program running on a CICS client call a CICS program located on a CICS server, and pass and retrieve data from it.

function shipping

A process, transparent to the application program, by which CICS accesses resources when those resources are actually held on another CICS system.

FOR—see File-Owning Region

File-owning region (FOR)

A CICS address space whose primary purpose is to manage files and databases. FOR is similar to data-owning region (DOR).

intercommunication facilities

A generic CICS term covering inter-system communication and multi-region operation.

Inter-Region Communication (IRC)

The method by which CICS implements multi-region operation.

Inter-System Communication (ISC)

A term that describes communication between separate systems using SNA networking facilities or using the application-to-application facilities of VTAM.

interval control

The CICS processing activity that provides time-dependent facilities.

IRC—see Inter-Region Communication

ISC—see Inter-System Communication

link

(1) A logical connection between two systems or applications. (2) A link is a programmatic request within one program to invoke another program. Upon completion of the invoked program, control returns to the invoking program at the point at which the link was executed.

mirror task

A task required to service incoming requests that specify one of the CICS mirror transactions (CSMI, CSM1, CSM2, CSM3, CSM5, CPMI, CVMI).

mirror transaction

A transaction initiated in CICS in response to a "function shipping" request from another CICS system. The mirror transaction recreates the original request and the request is issued. The mirror transaction returns the acquired data to the originating CICS system.

MRO—see Multi-Region Operation

MSGUSER

One of the DDNAMEs by CICS as a destination for log data.

multiprogramming

The concurrent execution of application programs across partitions.

Multi-Region Operation (MRO)

A way to communicate between CICS systems without the use of SNA networking facilities. The systems must be in the same operating system; or, if the XCF access method is used, in the same MVS sysplex.

Parallel Sysplex

An MVS Sysplex where all the MVS images are linked through a coupling facility.

principal facility

To a transaction, the principal facility is the session that activates it.

pseudoconversational

A type of CICS transaction that is designed to appear to the operator as a continuous conversation occurring as part of a single transaction.

quasi-reentrant

A characteristic of a CICS application that applies to programs that are serially reusable between entry and exit points because it does not modify itself or store data within itself between calls on CICS facilities.

queue

A collection of items formed by items in a system waiting for service; for example, tasks to be performed or messages to be transmitted in a message-switching system.

resource

Any facility of the computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs.

SDT—see Shared Data Table

Shared Data Table (SDT)

A copy of a data set that CICS loads into a MVS data space and is accessed by data table services like cross-memory instead of VSAM services.

start code

A 2-byte indicator showing how the transaction issuing the request was started. It can have a number of values, as described in the following table:

D	A distributed program link (DPL) request that did not specify the SYNCONRETURN option. The task cannot issue I/O requests against its principal facility, nor can it issue any syncpoint requests.
DS	A distributed program link (DPL) request that did specify the SYNCONRETURN option. The task can issue syncpoint requests.
S	START command without data.
SD	START command with data.
TD	Terminal input or permanent transid.

SIT—see System Initialization Table

System initialization table (SIT)

A CICS table of user-specified data that controls how CICS is initialized.

session

A logical link between two CICS systems that communicate using intersystem communication or MRO. In CICS intersystem communication, a session makes use of an SNA LU-LU session.

taskid

An identifier (number) given by CICS to a task.

task

The dispatching unit of execution in CICS corresponding to an invocation of a transaction for a particular user.

TCT—see Terminal Control Table

TCTTE—see Terminal Control Table Terminal Entry.

Temporary Storage (TS)

A CICS facility use to temporarily save data in the form of a sequential queue. A TS queue is held in main storage or on a VSAM data set on DASD. All queues not in main storage are in a single VSAM data set. A task can create a TS queue with a name selected by the task. The queue exists until deleted by a task (usually, but not necessarily, the task that created it).

Temporary Storage Queue (TSQ)

A storage are for data that is managed by CICS.

Temporary Storage Queue Owning Region (TSQOR)

A CICS region that owns and manages access to temporary storage queues. See TSQ.

Temporary Storage Table (TST)

An in storage table describing temporary-storage queues and queue prefixes for which CICS is to provide recovery.

terminal

In CICS, a device equipped with a keyboard and some kind of display, capable of sending and receiving information over a communication channel.

Terminal Control Table (TCT)

A table describing a configuration of terminals, logical units, or other CICS systems in a CICS network with which the CICS system can communicate.

terminal control table terminal entry (TCTTE TCTE).

In the TCT, an entry for each terminal known to CICS. TCTTEs are generated either during system initialization (for terminals predefined by resource definition) or when a terminal is auto installed. The TCTTE describes the terminal and addresses the corresponding TCTLE (RPL for VTAM terminals), the active TCA, and TIOAs; it also contains control information relating to terminal control requests issued by the CICS application program.

Terminal-owning region (TOR)

The CICS system region that owns the terminals.

TOR - see Terminal-Owning Region

transaction

A unit of processing (consisting of one or more programs) initiated by a single request, often from a terminal. A transaction can require the initiation of one or more tasks for its execution.

transaction routing

A type of CICS intercommunication in which users of a CICS system can execute transactions that exits in a CICS system other than the one in the user's terminal is connected.

transaction backout

The system activity that occurs because of a transaction failure. Updates to resources associated with the transaction are backed out.

transaction identifier

A synonym for transaction name or transaction code. For example, a transaction identifier is a one to four character code entered by an operator when selecting a transaction.

TS—see Temporary Storage

TST—see Temporary Storage Table

Two-Phase Commit

When a transaction is operating as part of a distributed Unit-of-Work (UOW), two-phase commit is the protocol used to coordinate and execute a syncpoint. At syncpoint, all updates to recoverable resources must either be committed or backed out. At this point, the coordinating recovery manager gives each subordinate participating in the UOW an opportunity to vote on whether its part of the UOW is in a consistent state and can be committed. If all participants vote yes, the distributed UOW is committed. If any vote no, all changes to the distributed UOW's resources are backed out.

This is called the two-phase commit protocol, because there is first a "voting" phase (the prepare phase), which is followed by the actual commit phase. This can be summarized as follows:

1. PREPARE

   Coordinator invokes each UOW participant, asking each one if it is prepared to commit.

2. COMMIT

   If all UOW participants acknowledge that they are prepared to commit (vote yes), the coordinator issues the commit request.

If any UOW participant is not prepared to commit (votes no), the coordinator issues a backout request to all.

trace

A facility for recording CICS activity. There are three destinations for trace entries: internal trace, auxiliary trace, and externally to the Generalized Trace Facility (GTF).

Transaction Work Area (TWA)

An option specified on the ADDRESS command. The TWA is an in memory storage area used to pass information between application programs, but only if they are in the same task. The pointer reference is set to the address of the TWA. If a TWA does not exist, the pointer reference is set to X'FF000000'.

TSQ—see Temporary Storage Queue

TSQOR—see Temporary Storage Queue Owning Region

A CICS region that owns temporary storage queues.

TWA—see Transaction work area

User-Maintained Shared Data Table

An MVS data table whose records are not automatically reflected in the source data set.

Unit of Work (UOW)

The processing activities associated with a sequence of processing actions (database changes, for example) that must be completed before any of the individual actions performed by a transaction can be regarded as committed. After changes are committed (by successful completion of the UOW and recording of the syncpoint on the system log) they become durable, and are not backed out in the event of a subsequent failure of the task or system.

The beginning and end of the sequence can be marked by the following activities:

- Start and end of transaction, when there are no intervening syncpoints
- Start of task and a syncpoint
- A syncpoint and end of task
- Two syncpoints

Thus a UOW is completed when a transaction takes a syncpoint, which occurs either when a transaction issues an explicit syncpoint request, or when CICS takes an implicit syncpoint at the end of the transaction. In the absence of user syncpoints explicitly taken within the transaction, the entire transaction is one UOW.

Unit of Work Identifier (UOW ID)
A unique identifier as known within the originating system. On MVS, a UOW ID can be assigned explicitly using an API call or is automatically assigned by the target system (CICS, WebSphere, and so on). It is a binary value derived from the originating system clock or a character value (hhmmss).

UOW—see Unit Of Work

UOW id—see Unit Of Work Identifier

user authentication
The process by which a service accurately establishes the authenticity of a user making a request.

user identification
The process by which the identity of a user is established. Typically, the term user ID is used to denote the user's identity. In Java terminology the term principal is used.

IMS Terminology

ACB—see Application Control Block

ACBGEN—see Application Control Block Generation

alternate program communication block
A TP PCB, defined by the user, that can be used to describe output message destinations other than the terminal that originated the input message. Where SAMETRM=YES is not implicitly or explicitly specified, an alternate PCBs destination can be either a logical terminal or an input transaction queue.

APPC—see Advanced Program-to-Program Communications

APPC/IMS
A part of the IMS Transaction Manager that uses the CPI communications interface to talk to application programs.

APPLCTN Macro
An IMS macro that defines the process resource requirements for an application that runs under the control of IMS/DC and those applications that access databases through DBCTL. An APPLCTN macro combined with one or more TRANSACT macros defines the scheduling and resource requirements for an IMS application program.

Application Control Block Generation (ACBGEN)
The process by which the application control blocks in IMS are generated.

Application Control Block (ACB)

An IMS control block created from the output of DBDGEN and PSBGEN and placed in the ACB library for use during online and DBB region type execution of IMS.

Base Primitive Environment (BPE)

A system-service-layer of IMS that provides a common set of system services (such as storage management, tracing, and dispatching) to various components such as the IMS Common Queue Server (CQS) and IMS Client Server Object Manager (IMS CSobject) and IMS Connect.

Batch Message Processing (BMP) program

An IMS batch processing program that has access to online databases and message queues. BMPs run online, but like programs in a batch environment, they are started with job control language (JCL).

batch-oriented BMP program

A BMP program that has access to online databases and message queues while performing batch-type processing. A batch-oriented BMP does not access the IMS message queues for input or output. It can access online databases, GSAM databases, and MVS files for both input and output. Contrast with transaction-oriented BMP.

batch processing program

An application that has access to databases and MVS data management facilities but does not have access to the IMS control region or its message queues. (*See also* batch message processing program and message processing program.)

BPE—see Base Primitive Environment

BMP—see Batch Message Processing Program

class

An attribute related to a transaction code and a message region that is used to determine scheduling. (*See also* message class and region class.)

cold start

An IMS system start type that results in IMS being initialized for the first time or when some error condition prevents a warm or emergency restart. (*See also* emergency restart and normal restart.)

control program (IMS)

An IMS program that initiates and controls the major IMS facilities, such as IMS database, telecommunications, and message scheduling.

control region

An MVS main storage region that contains the IMS control program.

conversation

A dialog between a terminal and a message-processing program that uses IMS conversational processing facilities. A conversation can also be a dialog between an LU 6.2 program and an IMS application program. (*See also* conversational processing.)

conversational processing

A programming style letting a user's application program accumulate information acquired through multiple interchanges with a terminal, even though the program terminates between interchanges.

CSQQTRMN

The MQSeries supplied IMS application that monitors an MQSeries initiation queue and starts an IMS transaction when an MQSeries event occurs on that queue.

Data Base Description (DBD)

A collection of macro statements that define the characteristics of a database, such as the database's organization and access method, the segments and fields in a database record, and the relationship between types of segments.

Data Base Description Generation (DBDGEN)

A DBDGEN is the process by which a DBD is created.

Data Base Program Communication Block (DBPCB)

A control block that describes an application program's interface to a database. One DBPCB is required for each database view used by the application program.

Data Communications Control (DCCTL)

A subsystem that lets the IMS Transaction Manager act as a stand-alone, full-function transaction manager that can connect to DB2 or other external subsystems.

DBD—see Data Base Description

DBDGEN—see Data Base Description Generation

DBPCB - see Data Base Program Communication Block

data store

An IMS Transaction Manager system that provides transaction and database processing.

dependent region

An MVS virtual storage region that contains MPPs, BMPs, MDPs, or online utilities.

emergency restart

A restart of IMS following an IMS or MVS failure.

group member

The name of an entity that joins an XCF group and communicates with IMS using the OTMA protocol. A member can be either a server (IMS) or a client.

IMS Connect

An MVS gateway application program that uses TCP/IP communications to TCP/IP clients or Web browsers and IMS OTMA communication to IMS.

IMS Connect Base Primitive Environment (IMS Connect BPE)

A system service component that underlies the address space of IMS Connect.

IMS DB

An IMS licensed program capable of processing concurrent data base calls and offers high performance for a variety of applications, ranging from those with moderate volume and complex data structures to those with extremely high volumes and simple data structures.

IMS—see Information Management System

IMS Open Transaction Manager Access (OTMA)

OTMA is connectionless client/server protocol used to communicate with IMS transaction programs.

IMS request message (IRM)

A structured IMS message that passes required IMS data and user message exit data to IMS Connect. Both the user message exits and IMS Connect use the IRM.

IMS Transaction Manager

The data communication system used within IMS to provide high-volume, high-performance, high-capacity, low-cost transaction processing for both IMS DB and DB2 databases.

Information Management System (IMS)

An IBM transaction processing product that consists of several system environments available with Database Manager and Transaction Manager, capable of managing complex databases and terminal networks.

Input/Output Program Communication Block (I/O PCB)

A DC-PCB provided automatically by IMS to an application program that executes in a communication system with the DC feature. The I/O PCB is the mechanism by which a program obtains an input message from a terminal and returns a reply to the terminal that originated the input message. (*See also* alternate program communication block.)

IRM—see IMS request message

line response mode

A variation of response mode where all operations on the communication line are suspended while the application program output message is being generated. *See also* response mode and terminal response mode.

LTERM—see Logical Terminal

Logical Terminal—LTERM

A symbolic name in IMS that is mapped to a VTAM LU or BTAM physical terminal.

message control information

The part of the OTMA message prefix that is not contiguous with the rest of the message prefix.The Message Control Information must be specified for every OTMA message, and contains such information as the transaction pipe name and the message type.

Message Processing Program (MPP)

An IMS application program that is driven by transactions and has access to online IMS databases and message queues. (*See also* batch message processing program and batch processing program.)

message class

A class, assigned to a transaction code, that determines within which message region an application program is to process that transaction. (*See also* region class.)

message queue

The data set on which messages are queued before being processed by an application program or sent to a terminal.

MPP—see Message Processing Program

MVS Extended Coupling Facility (XCF)

A component of MVS that provides functions to support cooperation between authorized programs running within a sysplex.

non-persistent socket connection

A connection between the client and IMS Connect that lasts for a single exchange of one input-and-output pair. IMS Connect terminates a non-persistent socket connection after sending each output message. IMS Connect requires the client application to reconnect before sending each input, and to terminate the connection after receiving each output. This socket connection was intended to support IMS Web connections.

normal restart

An IMS system start type that reinitializes and activates IMS. A normal restart can occur after a termination that was initiated by a /CHECKPOINT command.

Open Transaction Manager Access

A connectionless client/server protocol used to communicate with IMS transaction programs.

OTMA—see Open Transaction Manager Access

password security

IMS processing that makes use of system macros and security maintenance utility control statements to restrict the use of IMS resources (databases, application programs, physical and logical terminals, transactions, and commands) to a person or persons who can supply the correct password.

PCB—see Program Communication Block

persistent socket connection

A long-lived connection that exists between the client and IMS Connect and can last for multiple transactions. Either the client or IMS Connect can terminate a persistent socket connection.

primary request

In an MSC network, a message entered into a terminal before it is processed. *See also* secondary request and response.

Program Communication Block (PCB)

An IMS control block that describes an application program's interface to and view of an IMS database. For message processing and batch message processing programs, a PCB describes the source and destinations of messages. PCBs are defined during PSB generation. (*See also* data base program communication block and telecommunication-program program communication block.)

Program Specification Block (PSB)

An IMS control block that describes databases and logical message destinations used by an application program. A PSB consists of one or more PCBs.

Program Specification Block Generation (PSBGEN)

An IMS process by which a PSB is created.

PSB—see Program Specification Block

PSBGEN—see Program Specification Block Generation

region class

An IMS class that is assigned to a message region and indicates the message classes that can be processed within the region. (*See also* message class.)

reply

Synonymous with response in a non-VTAM environment.

response

A response is a message inserted to a logical terminal destination specified by an I/O PCB or an alternate response PCB. When VTAM is used, the term "reply" is substituted for "response" because response has a separate meaning in VTAM communications. (*See also* primary request, secondary request, and reply.)

Response Alternate PCB

An alternate PCB that is a synonym for alternate response PCB.

response mode

A mode of IMS terminal operation that synchronizes operations between the terminal operator and the application program. When IMS receives an input transaction that causes response mode to be entered, no more input is allowed until the application program response has been transmitted back to the terminal. (*See also* line response mode and terminal response mode.)

secondary request

In a multi-system environment, a message inserted to a transaction code destination by an application program. *See also* primary request and response.

server

A member of an XCF group that uses the OTMA protocol.

synchronization phase

An XRF phase, immediately after initialization, when the alternate builds the IMS control blocks to mirror those in the active.

telecommunication-program program communication block (TP PCB)

The PCB that supports communication between an application program and a terminal or other application program. There are two types of TP PCBs: I/O PCB and alternate PCB.

terminal response mode

The type of response mode that suspends all input operations from the terminal until the application program has generated the output message. *See* line response mode.

termination phase

An XRF phase in which a subsystem shuts down.

TM—see IMS Transaction Manager

Tmember

The name of a client that connects to an OTMA group.

TRANSACT macro

An IMS macro that is used during a PSBGEN to identify the transaction codes associated with an application program previously named in a preceding APPLCTN macro. The applications identified in TRANSACT macros are scheduled for execution in an IMS message processing region. The TRANSACT macro also provides the IMS control program with information that influences the application program scheduling algorithm. It can define a message editing routine. The following TRANSACT macro arguments specify attributes of the associated transaction:
EDIT= Specifies whether the input (ULC) data is to be translated to uppercase. The first parameter of this operand defines whether the transaction is uppercase/lowercase (ULC) as entered from the terminal, or if it is to be translated to uppercase (UC) before being presented to the processing program. The default is UC.
MSGTYPE Specifies the type of transaction code (single or multiple segment), and whether the communication line from which the transaction is entered is to be held until a response is received. The MSGTYPE operands are not position dependent.
MULTSEG Specifies that the incoming message can be more than one segment in length. It is not eligible for scheduling to an application program until an end-of-message indication is received, or a complete message is created by MFS.
SNGLSEG Specifies that the incoming message is one segment in length. It becomes eligible for scheduling when the terminal operator indicates end-of-segment.
NONRESPONSE Specifies that, for terminals specifying or accepting a default of OPTIONS=TRANRESP, input should not stop after this transaction is entered.
RESPONSE Specifies that, for terminals specifying or accepting a default of OPTIONS=TRANRESP, no additional messages are to be allowed after this transaction is entered until this transaction sends a response message back to the terminal. Response mode can be forced or negated by individual terminal definition.

transaction

A specific set of input data that triggers the execution of a specific process or job. A transaction is a message destined for an application program.

transaction code

A 1- to 8-character alphanumeric code associated with an IMS message-processing program. The transaction code is used to invoke the MPP.

transaction command security

IMS processing that makes use of system macros and security maintenance utility control statements to permit specific application programs to issue some of the IMS operator commands.

transaction load balancing

Optional processing within IMS that enables a transaction to be scheduled into more than one message, or batch message, region at the same time.

transaction-oriented BMP

A BMP that performs transaction-type processing in a batch environment. A transaction-oriented BMP gets its input from the IMS message queues and can use the message queues for output. Contrast with batch-oriented BMP.

transaction pipe

A named IMS process management resource. An OTMA client must specify this resource when submitting a transaction to IMS. A Tpipe is analogous to an LTERM.

transaction socket connection

A connection between the client and IMS Connect that lasts for a single transaction. A transaction socket connection is terminated at the end of the transaction.

Unit of Work (UOW)

(1) For IMS DB, unit of work encompasses all the input and output messages associated with a transaction. (2) For IMS TM, UOW is a single IMS message. (3) For CQS, UOW is a client-defined grouping of data objects. (4) In advanced-program-to-program communication (APPC), UOW is the amount of processing that is started directly or indirectly by a program on the source system.

user message exit

An IBM supplied or user-written program that converts client message format to OTMA message format before IMS Connect sends the message to IMS. The same program converts OTMA message format to client message format before sending the message back to IMS Connect to send to the client.

warm start

An IMS start type that is synonymous with a normal IMS restart.

XCF - see MVS Extended Coupling Facility

XCF group

A logical collection of XCF members.The datastore (an IMS MPP) and IMS Connect must join to the same XCF group.

XCF member

An MVS application that joins an XCF group.

EJB Terminology

application server

A integrated set of Java EE software that provides several functions including a Web Server, JSP Server, JNDI Server, and an EJB Server.

application assembly

The process of collecting the various parts of a Java EE application (Web pages, JSPs, EJBs, and so on.) and placing them into WAR and EAR files for deployment onto an applications server. The resulting collection of files can also be known as being an "application assembly."

application deployment

The process of distributing the various parts of Java EE application (Web pages, JSPs, EJBs, and so on.) to the appropriate server within an application server. Application deployment is a process performed by the application server.

byte code

The Java compiler generates byte codes. A byte code is a high-level, machine-independent code for a hypothetical machine that is implemented by the Java interpreter and run-time system.

class

A software construct that defines the data (state) and methods (behavior) of the specific objects that are subsequently constructed from that class. A class in and of itself is not an object. A class is a description of an object. A class describes how the object looks and behaves when the object is created or instantiated from the specification declared by the class.

class loader

An object that is responsible for loading classes. Given the name of a class, a class loader should attempt to locate or generate data that constitutes a definition for the class. A typical strategy is to transform the name into a file name and then read a "class file" of that name from a file system.

data source

An alternative to the Driver Manager facility, a Data Source object is the preferred means of getting a connection to a particular DBMS or some other data source, such as a file. Data Source objects can provide connection pooling and distributed transactions.

deployment

See application Deployment.

deployment descriptor

An XML file within a WAR, EJB-JAR, or EAR file that contains the description of how the contents of the parts of the associated Java EE application are to be deployed by the application server.

EAR—see Enterprise Archive

EJB—see Enterprise Java Bean

EJB-JAR File

A specialized JAR file that contains the portion of the application that will be deployed to an EJB server. The EJB-JAR file includes a deployment descriptor, plus all the files needed to implement the EJB(s).

Enterprise Archive (EAR)

A specialized type of JAR file that usually contains the entire Java EE application. An EAR includes a deployment descriptor, as well as a WAR and EJB-JAR. The deployment descriptor describes the WAR and EJB-EAR files, plus any security and database information specific to the application.

Enterprise Java Bean (EJB)

A component architecture that is described as part of the Java EE specifications. EJB is used for the development and deployment of object-oriented, distributed, enterprise-level applications. Applications written using the EJB architecture are scalable, transactional, and secure.

Extensible Markup Language (XML)

A tag oriented, self-defining, markup language that defines the set of rules for describing the content and context of data within a document. Unlike other markup languages, XML is easily extended letting the markup language itself be user defined. XML describes a document's structure. It does not describe the rules of how the document should be formatted, presented or rendered. XML is a meta-markup language used for describing other markup languages.

home interface

One of two interfaces for an enterprise bean. The home interface defines zero or more methods for managing an enterprise bean. The home interface of a session bean defines create and remove methods, whereas the home interface of an entity bean defines create, finder, and remove methods.

JAR—see Java Archive

Java Archive (JAR)

A single compressed file that usually contains many different files. The types of files included in a JAR can be different. JAR files are compressed to save disk space and network bandwidth when they are transmitted across a network.

Java Remote Method Invocation (Java RMI)

A "remote procedure call (RPC)" mechanism that is native to Java. Java RMI provides a client application with the ability to invoke an EJB server application. Java RMI serializes the import and export view objects of the target EJB. These serialized objects serve as the input and output data of the invoked EJB.

Java RMI—see Java Remote Method Invocation

Java Naming and Directory Interface (JNDI)

A Java EE API that provides naming and directory functionality.

Java Server Page (JSP)

An extensible Web technology that uses static data, JSP elements, and server-side Java objects to generate dynamic content for a client. Typically the static data is HTML or XML elements, and in many cases the client is a Web browser.

Java Transaction API (JTA)

A Java EE API that lets applications and Java EE servers access transactions.

Java Platform, Enterprise Edition (Java EE)

An environment for developing and deploying enterprise applications. The Java EE platform consists of a set of services, application-programming interfaces (APIs), and protocols that provide the functionality for developing transaction oriented, multi-tiered, Web-based applications.

JNDI—see Java Naming and Directory Interface

JSP—see Java Server Page

JTA—see Java Transaction API

JVM—see Java Virtual Machine

Java EE—see Java Platform, Enterprise Edition (Java EE)

remote interface

One of two interfaces for an enterprise bean. The remote interface defines the business methods callable by a client.

servlet

A Java program that extends the functionality of a Web server, generating dynamic content, and interacting with Web clients using a request-response paradigm.

serialization

The process of saving an object's state to a sequence of bytes, as well as the process of rebuilding those bytes into a live object at some future time. The Java Serialization API provides a standard mechanism for developers to handle object serialization.

WAR—see Web Archive

Web Archive (WAR)

A specialized JAR file that contains the portion of the application that will be deployed to a Web Server. The WAR includes a deployment descriptor, plus all the files to be placed onto the Web Server.

XML—see Extensible Markup Language

Tuxedo Terminology

application

Software and its associated resources (databases, files, RMs, and so on.) used to computerize a business function. In Tuxedo an application is defined by a TMIB, and is synonymous with a domain.

Application-to-Transaction Monitor Interface (ATMI)

The API Tuxedo applications use to communication with Tuxedo's Transaction Program Monitor (TPM).

ATMI—see Application-to-Transaction Monitor Interface

AUTOTRAN

A Tuxedo transaction configuration option that defines a service to automatically start in transaction mode if the incoming request is not already part of an existing transaction.

Bulletin Board

A memory resident data structure used to manage various aspects of the Tuxedo system. A Bulletin Board contains the runtime representation of the TMIB. The Tuxedo Bulletin Board distributes, and partially replicates, the in memory resident bulletin Board.

Jolt

A Java-based interface to the Tuxedo system that extends the functionality of existing Tuxedo applications to include Intranet and Internet-wide availability.

Jolt Server Listener (JSL)

A Jolt process that handles the initial Jolt client connection, and assigns a Jolt client to the Jolt Server Handler.

Jolt Server Handler (JSH)

A Jolt process that manages network connectivity, executes service requests on behalf of the client and translates Tuxedo buffer data into the Jolt buffer, as well as Jolt buffer data into the Tuxedo buffer.

Jolt Repository

A central repository contains definitions of Tuxedo services.

Jolt Repository Server (JREPSVR)

A Jolt process that retrieves Jolt service definitions from the Jolt Repository and provides the service definitions to the JSH. The JREPSVR also updates or adds Jolt service definitions.

JREPSVR—see Jolt Repository Server

JSH—see Jolt Server Handler

JSL—see Jolt Server Listener

TMS—see Transaction Manager Server

Transaction Manager Server (TMS)

A Tuxedo administrative server that manages the two-phase commit processing and recovery for global transactions.

TMIB—see Tuxedo Management Information Base

Tuxedo Management Information Base (TMIB)

A data structure definition of all Tuxedo administrative and application services, as well as resource definitions.

typed buffer

An area of memory, allocated from the TUXEDO system that has an associated data type and subtype or format.

view

A Tuxedo typed buffer in which the data is composed of one or more fields, similar to a C' structure or a COBOL record.

view file

An ASCII file containing a description of the layout of fields with an associated View.

viewc

The TUXEDO view compiler command which parses a view file and generates a binary file used at run-time to interpret fields in a VIEW typed buffer.

Workstation Client (WSC)

The Tuxedo software component that provides a client application with the ability to accesses the Tuxedo system using the network (contrast to local client).

Workstation Handler (WSH)

A surrogate client that executes operations on behalf of one or more workstation clients. The WSH is a WSC's interface into the Tuxedo system.

Workstation Listener (WSL)

A Oracle Tuxedo supplied server that enables access to native services by workstation clients.

.JBLK file

Jolt Bulk Loader file. Processing the .jvf using the scripts provided with Tuxedo Middleware Support creates the .jblk file.

.JVF File

Jolt View File containing the View Definitions expected by a Tuxedo Server. This file is a mirror image of the .tvf except for some data-item-name changes and data-type changes.

.TVF File

Tuxedo View File containing the View Definitions expected by the Tuxedo Server.

MQSeries Terminology

CKTI

An MQSeries supplied CICS transaction that monitors an MQSeries initiator queue. CKTI starts a CICS transaction when an MQ event occurs on that queue.

CSQQTRMN

An MQSeries supplied IMS application that monitors an MQSeries initiation queue and starts an IMS transaction when an MQSeries event occurs on that queue.

dynamic queue

A local queue that is created when a program opens a model queue object.

initiation queue

A local queue on which a queue manager places trigger messages.

local queue

A queue belonging to the local queue manager. A local queue can contain a collection of messages waiting to be processed. Contrast with remote queue.

local queue manager

The queue manager to which a program is connected and provides message queuing services to the program. Queue managers to which a program is not connected are called remote queue managers, even if they are running on the same system as the program.

local definition

An MQSeries object that belongs to a local queue manager.

local definition of a remote queue

An MQSeries object that belongs to a local queue manager. This object defines the attributes of a queue that is owned by another queue manager. In addition, it is used to provide an alias for a queue-manager or an alias to a reply-to-queue.

message channel

In distributed message queuing, a message channel is the mechanism through which messages are moved from one queue manager to another. A message channel comprises two message channel agents (a sender and a receiver) and a communication link. Contrast with MQI channel.

message queue

Synonym for queue within an MQSeries system.

message queue interface (MQI)

The programming interface provided by the MQSeries queue managers. This API lets application programs access message queuing services from an MQSeries system.

model queue object

A set of queue attributes that act as a template when a program creates a dynamic queue.

MQI—see Message Queue Interface

MQI channel

A type of MQSeries connection that connects an MQI client to a queue manager on a server system. The client transfers only MQI calls and responses in a bi-directional manner. Contrast with message channel.

MQI client

A part of the MQSeries product that can be installed on a system without installing the full queue manager. The MQI client accepts MQI calls from applications and communicates with a queue manager on a server system.

MQSeries

A family of IBM licensed programs that provide message queuing services.

non-persistent message

A message does not survive a restart of the queue manager.

object

A controlled item or resource within MQSeries. An object is a queue manager, a queue, a process definition, a name list, or a channel.

persistent message

A message that is able to survive a restart of the queue manager.

queue

An MQSeries object which applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages, they point to other queues, or can be used as models for dynamic queues.

queue manager

(1) A system program that provides queuing services to applications. It provides an application-programming interface so that programs can access messages on the queues that the queue manager owns. *See also* local queue manager and remote queue manager. (2) An MQSeries object that defines the attributes of a particular queue manager.

remote queue

A queue belonging to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues.

remote queue manager

To a program, a queue manager is remote if it is not the queue manager to which the program is connected.

remote queuing

In message queuing, the provision of services to enable applications to put messages on queues belonging to other queue managers.

reply-to queue

The identity of a queue onto which the program that issued an MQPUT call wants a reply message or report message sent.

transmission queue

A local queue on which messages are temporarily stored prior to being transmitted to its associated remote queue manager.

trigger event

An event (such as a message arriving on a queue) causes a queue manager to create a trigger message on an initiation queue.

triggering

In MQSeries, a facility that lets a queue manager start an application automatically when predetermined conditions on a queue are satisfied.

trigger message

A message containing information about the program that a trigger monitor is to start.

trigger monitor

An MQSeries application that serves one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

.NET Terminology

ADO .NET

A component of the Microsoft .NET Framework for providing dynamic access to data held in a database.

AppDomain—see Application Domain

Application Domain (AppDomain)

Known as a lightweight process. Prior to .NET, isolation was achieved through separate processes using the assistance of the OS and the supporting hardware. If one process was having problems it would not impact the entire machine, the impact would be isolated to just the involved process. Because types are tightly controlled with the .NET Framework, a mechanism exists to run several application domains (AppDomains) in a single process, the same level of isolation can occur within a single process, but without incurring the additional overhead of making cross-process calls or switching between processes.

ASP .NET

A component of the Microsoft .NET Framework for building, deploying and running Web applications and distributed applications.

assembly

The primary unit of deployment within the .NET Framework. Within the base class libraries is a class that encapsulates the features of an assembly, and is named Assembly. This class exists in the System namespace. An assembly can contain references to other assemblies and modules. Developers create a managed code assembly by combining their own application source code with code from the .NET class libraries.

common language runtime (CLR)

The core runtime engine in Microsoft .NET Framework for executing .NET applications. The CLR loads an application using the application's assembly manifest. The CLR locates the correct version of each assembly within an application. The CLR supplies managed code with services such as cross-language integration, code access security, object lifetime management, and debugging and profiling support.

CLR—see Common Language Runtime

COM+

An extension of Microsoft Component Object Model (COM) application programming environment. COM+ is Microsoft's strategic approach for developing applications using programmatic build block.COM+ is both an object-oriented programming architecture and a set of operating services.

config files

Configuration files such as the web.config, machine.config, and the global.config let an application's assembler or deployer customize their application using declaratives specified within one or more configuration files.

csc.exe

The C# (C Sharp) compiler.

Distributed Component Object Model (DCOM)

A set of Microsoft concepts and programmatic interfaces that facilitate communication between a client program object and a server program object. The server program object is distributed on a network-connected computer.

DCOM—see Distributed Component Object Model

GAC—see Global Assembly Cache

garbage collection

A .NET feature provided by the CLR. The CLR monitors the execution of the translated managed code and periodically releases memory when the application no longer has any valid reference to that memory. The process of garbage collection absolves the application developer from having to keep track of memory usage and explicitly request unused memory be returned to the system.

Global Assembly Cache (GAC)

A .NET Framework facility that lets assemblies be stored globally in a cache. The GAC lets .NET applications access assemblies without having an explicit reference in their own assemblies.

IIS—see Internet Information Server

IL—see Intermediate Language

instance

A term used to describe a particular instantiation of a programmatic abstraction of a template, such as a class, object, or a computer process.

Intermediate Language (IL)

The language in which assemblies are written. It is a set of instructions that represent the code of an application. It is considered to be "intermediate" because it is not code that is native for a particular processor. The intermediate code must be translated into a processor's native code. When the code that describes a method is required to run, it is compiled into native code with the Just-in-Time (JIT) compiler.

Internet Information Server (IIS)

A Microsoft product that provides a group of Internet servers including a Web or HTTP server, and an FTP server.

JIT—see Just-In-Time Compilation

Just-In-Time Compilation

A term that refers to the CLR's ability to defer the translation of an application's MSIL until the code is actually needed during execution.

managed code

A .NET term that refers to MSIL code that is loaded, translated, verified and executed by .NET's CLR. Managed code must supply the information necessary for the CLR to provide services such as memory management, cross-language integration, code access security, and automatic lifetime control of objects. All MSIL code executes as managed code.

Microsoft Intermediate Language (MSIL)

A CPU-independent instruction set that is generated by the .NET Framework compiler. Before MSIL can be executed it must be converted to native, CPU-specific code by the CLR.

module

A single file that contains executable content. An assembly can encapsulate one or more modules; a module does not stand-alone without an assembly referring to it. A class exists in the .NET base class library that encapsulates most of the features of a module. This class is called Module. The Module class exists in the System namespace.

MSIL—see Microsoft Intermediate Language

object

In object-oriented programming (OOP), objects are programmatic units that contain code and optionally data.

resgen.exe

A .NET utility that converts files from one resource format to another.

serialization

In .NET, serialization refers to the process of encoding an object, and the objects that are reachable from that object, into a byte stream. The byte stream represents an instantiated object and contains sufficient information to create a reconstruction of the object. The byte stream is suitable for transmitting over a network.

type

Within each computer language, a type is a definition from which values can be instantiated. The .NET's Common Type System (CTS) eliminates the need for each language to implement data types in its own unique and incompatible way. CTS understands two fundamental different types; Value types and Reference types. Types can have static, instance and virtual methods, and static and instance fields, events and properties.

type loader

The part of the .NET CLR that is responsible for loading the implementation of a class into memory, checking it for consistency, and preparing the class for execution.

unmanaged code

Code that is native to the processor and is executed directly by the operating system. Unmanaged code provides its own memory management, type checking, and security support. Interoperability features of the CLR let applications mix managed and unmanaged code. It is important to note that unmanaged code executes outside the control of the CLR and therefore an application is vulnerable to failures occurring in unmanaged code.

Windows Server 2008

A Microsoft multipurpose operating system capable of handling a wide range of server roles. Some server roles include; File and Print server, Web server, Application server, Mail server, Terminal server, Remote Access server, Virtual Private Network (VPN) server, Directory Service server, Domain Name System (DNS) server, Dynamic Host Configuration Protocol (DHCP) server, Streaming Media server, and Windows Internet Name Service (WINS) server.

xcopy deployment

The deployment of .NET assemblies can be accomplished by doing nothing more than xcopy'ing the application directories to a different path destination.

.NET Framework

A set of integrated components, provided as class libraries in a single, logically organized class hierarchy. The .NET Framework provides .NET application developers with the software they need to build code that executes under the control of NET's CLR. The .NET Framework provides an enormous collection of system support functions including technologies for Web services, and Web applications (ASP.NET), data access (ADO.NET), smart client applications (Windows Forms), file system and network access, XML functionality and many others.

Web Services Terminology

SOAP—Simple Object Access Protocol

A *Simple Object Access Protocol* is a lightweight protocol for exchange of information in a decentralized, distributed environment.

WSDL—Web Services Description Language

A *Web Services Description Language* is an XML application for describing Web services. It is designed to separate the descriptions of the abstract functions offered by a service, and the concrete details of a service, such as how and where that functionality is offered.

Web Service

A *Web Service* is a software system designed to support interoperable machine-to-machine interaction over a network.

TCP/IP Terminology

address

A unique, digital identity assigned to each computer connected to a network.

DNS—see Domain Name Service

domain name

A Domain is part of a naming hierarchy in which the individual host names belong. A host's domain name is fully qualified when each portion of the naming hierarchy is specified to form its name. Each name within the hierarchy is separated by a dot.

Domain Name Service (DNS)

An Internet service that provides a translation of a host's Internet name to its Internet address.

host

A term used to refer to a computer connected to an IP Network. Each host is uniquely identified by its IP Network address. Additionally, a host can be identified by it host name.

host name

A label or mnemonic used to identify a computer in a more easily recognized form.

Internet

A large, worldwide "network of networks." The networks are interconnected, all sharing a common addressing scheme.

Internet Protocol (IP)

A connectionless protocol that routes data through a network or interconnected networks. IP acts as an intermediary between the higher protocol layers and the physical network.

IPv4 – Internet Protocol version 4

IPv4 address

A "dotted-decimal notation" networking address scheme used within an IP network. Each host connected to an IPv4 Network must have its own unique IP Address. The IPv4 Address is broken into parts. One part identifies the network (or subnet) address of which the particular host belongs. Another part of the IP Address identifies the specific host within that network.

IPv6 – Internet Protocol version 6

IPv6 address

IPv6 is a 128-bit addressing scheme, as opposed to the 32-bit addressing scheme of IPv4, supporting a much higher number of addresses. IPv6 is backward compatible and is designed to fix the shortcomings of IPv4, such as data security and maximum number of user addresses. It also features other improvements over IPv4, such as support for multicast and anycast addressing.

IP address resolution

The process of mapping a computer's network interface IP address with its hardware, or MAC, address.

MAC—see Media Access Control

Media Access Control (MAC)

A unique address burned into a read-only memory (ROM) by the manufacture of a network interface card.

socket

An endpoint of that provides the means by which an application connects to an IP network.

stream

A series of bytes that is transmitted over a connection-based transport protocol such as TCP.

TCP—see Transmission Control Protocol

TCP/IP—see Transmission Control Protocol/Internet Protocol

Transmission Control Protocol

A data-transparent connection oriented transport protocol providing communications between two connected sockets. The TCP protocol defines the process by which data is transmitted in streams across the network. TCP provides for reliable data transfers by handling the acknowledgement of data received, retransmission of data if not properly acknowledged, preserving the sequence of data, and flow control.

Transmission Control Protocol/Internet Protocol (TCP/IP)

The two primary protocols within the Internet Suite of protocols. TCP/IP provided network connection between any computer systems, across varying network media.

Windows Sockets (WinSock)

An open interface or API for network programming used within a Windows environment.

Windows Socket API (WSA)

WSA provides a Windows application standard access to network services and underlying protocol stack.

WinSock—see Windows Sockets

WSA—see Windows Socket API

SNA Terminology

APPC—see Advanced Program-to-Program Communications

Advanced Program-to-Program Communications (APPC)
A set of application programming verbs that is used by an application program to communicate with another program. APPC provides a programmatic interface to SNA's LU 6.2 protocol.

APPL Statement
A VTAM application definition.

APPN—see Advanced Peer-to-Peer Networking
An extension to SNA that features greater distributed network control, dynamic definition of network resources, and automated resource registration and directory lookup.

BIND
An SNA command that results in to Logical Units (LUs) going into session. The BIND establishes the characteristics of the LU-to-LU session.

Common Program Interface—Communications (CPI/C)
IBM's Systems Applications Architecture (SAA) describes a set of software interfaces, conventions, and protocols that provide a framework for building distributed processing applications. CPI/C is part of SAA, and provides calls for transaction programs to participate as a one of the programs in an LU 6.2 program-to-program application. The verbs provided as part of CPI/C provide a common interface to the underlying APPC implementation.

Communications Server
An IBM product that provides SNA networking components used to connect PC-base Local-Area Network workstations to an SNA Network.

conversation
A term used to describe the logical connection between a pair of transaction programs. An established conversation has exclusive use of the session the conversation has been allocated to use. Depending on the type of LU used to support an LU 6.2 conversation, the LU can support the execution of conversation in a serialized fashion or concurrently. It is possible, when using an Independent LU, for multiple conversations between different pairs of transaction programs to be active concurrently, with each conversation being allocated to a distinct session.

CPI/C—see Common Program Interface—Communications

dependent LU

An Logical Unit that can only support a single session at a time. Therefore, a dependent LU can support being used by a single conversation.

FMH-5

An LU 6.2 Function Management Attach Header that describes the request for an LU 6.2 conversation to be established between two application programs.

Host Integration Server (HIS)

A Microsoft product that provides a collection of products that integrate Web or workstation-based applications to applications spread across a network. HIS offers an SNA networking component which connects PC-based Local-Area Networks with SNA networks.

Logical Unit (LU)

A type of SNA NAU that serves as the port through which an end-user of an SNA network communicates. The communications path through an SNA network is represented by an LU-to-LU session, with each user of the network represented by its associated LU. An LU connects a program to the underlying SNA network.

LU—see Logical Unit

LU 6.2

A type of SNA LU that defines the protocol for the message and control data that is exchanged between an APPC application program and it partner program.The LU 6.2 protocol also describes the interaction between the APPC program and the control programs of the SNA network.

ILU—see Independent Logical Unit

Independent Logical Unit (ILU)

A specific type of Logical Unit that is capable of supporting multiple sessions concurrently with other LU's defined in the SNA network. The ability of an ILU to manage parallel sessions lets multiple conversations be allocated and processing concurrently, each conversation allocated to its own session.

NAU—see Network Addressable Unit

Network Addressable Unit (NAU)

A resource managed by the products that implement an SNA network. An NAU is a uniquely identified by it network name and is assigned a unique network address.

Network Node (NN)

An APPN implementation of an SNA Physical Unit.

NN—see Network Node

parallel sessions

A term used to describe the occurrence of two or more sessions between the same pair of LUs. Multiple sessions can concurrently use the same physical resources used to connect the pair of LUs.

Physical Unit (PU)

A type of SNA NAU that represents a node within an SNA network. The PU controls the physical configuration and logical resources associated with its associated SNA node.

PU—see Physical Unit

Request Unit (RU)

A formatted message that is transmitted between NAUs.

RU—see Request Unit

session

A formally bound paring between two NAUs. A bound session must exist before the two partners involved in the session can communicate. A session provides a relatively long-lived connection between two LUs. A session can be used by a succession of conversations.

SNA—see Systems Network Architecture

SNA Network

A logical set of interconnected NAUs over which explicitly formatted message traffic flows. The NAU's are connected by a path control network that is implemented over a physical network of interconnected communications gear, each implementing a portion of the SNA network. An SNA network consists of physical nodes that are interconnected by communications links.

SNA node

A grouping of hardware and software that implement a portion of an SNA network. There are different types of SNA nodes, each helping to facilitate the implementation of the network.

SSCP—see System Service Control Point

Systems Network Architecture (SNA)

A communications architecture developed by IBM that is divided into well-defined logical layers. SNA provides the set of rules that govern the format, definition, and sequence of information sent across an SNA network. SNA formally defines the functional responsibility and protocol assigned to various logical and physical components communicating within the SNA network.

System Service Control Point (SSCP)

A type of SNA NAU that represents the component(s) within the network that provide general management of the network. The SSCP controls the bring up of the network, establishment of sessions, and network recovery.

transaction programs

A program distinguished from other programs by the characteristic that other programs invoke it by using a mechanism known as an Attach. An Attach initiates a conversation with the target transaction program. Transaction programs inter-communicate over the conversation using a set of transaction programming verbs as provided by its associated type 6.2 LU.

Virtual Telecommunications Access Method (VTAM)

An IBM MVS product that provides SSCP communications support within an SNA network.

VTAM—see Virtual Telecommunications Access Method

ECI Terminology

CICS Transaction Gateway (CTG)

An IBM product that offers a set of client and server software that let a Java application invoke a program deployed to a CICS region.

CTG—see CICS Transaction Gateway

CTG Client Daemon

See Universal Client.

DPL—see Distributed Program Link

Distributed Program Link (DPL)

A programming mechanism that lets a CICS client program call a server program running in a remote CICS region, and pass and receive data using a COMMAREA.

ECI—see External Call Interface

External Call Interface (ECI)

An API that lets a non-CICS program running on a CICS client call a CICS program located on a CICS server, and pass and retrieve data from it. The target CICS load module is invoked as a Distributed Program Link (DPL). ECI provides client applications a remote call interface to COMMAREA-based CICS applications.

Universal Client

A client component of IBM's CTG. The Universal Client is also known as the CTG Client Daemon. On Windows platforms, the Universal Client runs as a service and provides the connection for applications executing on the Windows platform to programs that execute on one or more CICS regions.

Java Terminology

application server

A vendor provided Java environment that conforms to a certain version level of the Java EE specification. An application server provides the execution and administration environments suitable for providing a wide range of application services and web services.

bean

During development of Java software, using one or more commonly available Java IDE tools, a bean is recognized as a reusable Java component that can be visually manipulated by the tools. Beans are developed using a Beans Development Kit. When beans are in use by an application they can run inside a number of application environments, know as containers, including Web browsers, word processors and other Java applications. The properties of a bean are visible to other beans. An application learns about a bean's properties dynamically and interacts with the bean accordingly. A bean includes JavaBean statements that describe properties such as user interface and events that trigger the bean to communicate with other beans in the same container or available elsewhere in the network.

bytecode

Object code that is processed by a Java Virtual Machine (JVM). A Java compiler creates a bytecode stream from Java source code. The resulting bytecode stream is designed to be platform independent and is considered to be an architecture neutral object file format. A Java bytecode stream is intended to be easy to interpret on any machine and is easily translated into native machine code. Mechanisms vary by JVM on how bytecode is converted to native machine language (interpretation, just-in-time compiling, other conversion schemes).

class

A software construct that defines the data (state) and methods (behavior) of the specific objects that are subsequently constructed from that class. A class in and of itself is not an object. A class is a description of an object. A class describes how the object looks and behaves when the object is created or instantiated from the specification declared by the class.

class loader

An object that is responsible for loading classes. Given the name of a class, a class loader should attempt to locate or generate data that constitutes a definition for the class. A typical strategy is to transform the name into a file name and then read a "class file" of that name from a file system.

deployment

The process of distributing the various parts of Java application (for example, Web pages, JSPs, EJBs, and so on.) to the appropriate server within an application server. Application deployment is a process performed by the application server.

deployment descriptor

An XML file within a WAR, EJB-JAR or EAR file that contains the description of how the contents of the parts of the associated Java application are to be deployed by the application server.

EJB—see Enterprise JavaBeans

Enterprise JavaBeans (EJB)

An architecture that is described as part of the Java EE specifications. EJB is used for the development and deployment of object-oriented, distributed, enterprise-level applications. Applications written using the EJB architecture are scalable, transactional, and secure.

garbage collection

The process of automatic detection and freeing of memory that is no longer in use. The Java runtime system performs garbage collection so that programs never explicitly have to free unreferenced objects.

Java Development Kit (JDK)

A software development environment for writing applets and applications in the Java programming language.

Java Naming and Directory Interface (JNDI)

A Java EE API that provides naming and directory functionality.

Java Runtime Environment (JRE)

A subset of the Java Development Kit (JDK) for end-users and developers who want to redistribute the runtime environment alone. The Java runtime environment consists of the Java virtual machine, the Java core classes, and supporting files.

Java Server Page (JSP)

A server-side scripting language that allows combining HTML text with Java source code in the same document.

Java Virtual Machine (JVM)

A software "execution engine" that safely and compatibly executes the byte codes in Java class files on a microprocessor (whether in a computer or in another electronic device).

Java Platform, Enterprise Edition (Java EE)

A software specification for developing enterprise and distributed applications from JavaSoft (Sun Microsystems). Java EE describes a set of technologies described in the following list:

- JavaServer Pages (JSP)
- Servlets
- Enterprise JavaBeans (EJB) JDBC
- Java Naming and Directory Interface (JNDI)
- Support for XML
- Java Messaging
- Java Transaction Support
- JavaMail
- Java support for CORBA

Java Platform, Standard Edition (Java SE)

The Software Development Kit (SDK) is development environment for building applications, applets, and components using the Java programming language.

javac.exe

javac.exe is the Java compiler.

JDK—see Java Development Kit

JNDI—see Java Naming and Directory Interface

JRE—see Java Runtime Environment

JSDK—see Java Servlet Development Kit

JSP—see Java Server Page

Java Servlet Development Kit (JSDK)

A suite of software for easing the development and testing of java servlets, the javax.servlet package sources, and API documentation.

JVM—see Java Virtual Machine

Java EE—see Java Platform Enterprise Edition

Java SE—see Java Platform Standard Edition

Remote Method Invocation (RMI)

A Java programming construct that enables programmers to create distributed Java-to-Java applications, in which a remote Java object can be invoked from other Java virtual machines, possibly on different hosts. A Java technology-based program can make a call on a remote object after it obtains a reference to the remote object, either by looking up the remote object in the bootstrap naming service provided by RMI or by receiving the reference as an argument or a return value. A client can call a remote object in a server, and that server can also be a client of other remote objects. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism.

RMI—see Remote Method Invocation

serialization

A process that extends the core Java Input/Output classes with I/O support for objects. Serializing an object involves the encoding the object into a stream of bytes. Serialization also supports the reconstructing of an object from the serialized byte stream. Serialization is used for lightweight persistence and for communication using sockets or Remote Method Invocation (RMI). The default encoding of objects protects private and transient data, and supports the evolution of the classes. A class can implement its own external encoding and is then responsible for the external format. Serialization includes an API that lets the serialized data of an object be specified independently of the fields of the class and lets those serialized data fields be written to and read from the stream using the existing protocol to ensure compatibility with the default writing and reading mechanisms.

servlet

A Java class that is loaded dynamically to extend the functionality of a web server. Servlets are invoked from client applications executing in a web browser. Servlets execute as part of the invoked server application.

servlet engine

Software that extends the support provided by a browser. A servlet engine provides the environment that supports the execution of servlets.

WAR—see Web Archive

Web Archive (WAR)

A specialized JAR file that contains the portion of the application that will be deployed to a Web Server. The WAR includes a deployment descriptor, plus all the files to be placed onto the Web Server.

COM Terminology

apartment managed processing

Execution management processing that is part of the COM threading architecture. Apartments are used to allow access to individual instances of COM objects. An individual instance of a COM object lives in exactly one apartment. Apartments can be either single-threaded or multi-threaded. Calls to methods to a COM object within a single-threaded apartment are synchronized by the apartment's thread by making use of the Windows message queue. In contrast, a multi-threaded apartment can have one or more threads servicing calls to the methods of the associated COM object instance. The object itself must synchronize calls to COM object methods in a multi-threaded apartment.

COM—see Component Object Model

Component Object Model (COM)

A Microsoft object-oriented programming model that defines how objects interact within a single application or between applications. COM's architecture lets applications and systems be constructed using components supplied by different software vendors.

COM+

Microsoft's next step in the Component Object Model evolution. COM+ is a service oriented and handles many of the resource management tasks previously performed by application code. Applications that use COM+ objects automatically become more scalable by such facilities of thread pooling, object pooling, and just-in-time object activation. COM+ also provides distributed transaction support letting transactions span multiple databases and applications spread across a network.

DCOM—see Distributed COM

Distributed COM (DCOM)

Extends support of Microsoft's COM implementation to support communication between objects deployed to different computers connected to a network.

GUID—see Globally Unique Identifier

Globally Unique Identifier (GUID)

A 16-byte code used to uniquely identify interfaces to objects spread across all computers and networks.

IIS—see Internet Information Server

Internet Information Server (IIS)

An integrated Windows service offered Microsoft for hosting Internet or Intranet applications on various Windows platforms.

in-process

An application isolation level that lets a web application be loaded and run from within application server process space within IIS. The Web applications that execute in process are not isolated from one another. It is possible for a misbehaved Web application executing in process to effect other applications configured to run in process.

registration

The process of adding data to the Windows registry.

registry

A central, hierarchical database maintained in Windows used to store configuration information for the system or users of the system. The registry contains information that is constantly accessed during operation. The information includes such items as user profiles, application profiles and hardware properties.

out-of-process

An application isolation level within IIS that causes that associated application to run within its own execution environment. The process associated with the Web base request executes as part of a separate dllhost.exe process. The Web processing designated to execute as out-of-process is isolated from all other Web applications.

XML Terminology

attribute

A property of an element.

element

A tagged field within an XML document.

Extensible Markup Language (XML)

A tag oriented, self-defining, markup language that defines the set of rules for describing the content and context of data within a document. Unlike other markup languages, XML is easily extended letting the markup language itself be user defined. XML describes a document's structure. It does not describe the rules of how the document should be formatted, presented or rendered. XML is a meta-markup language used for describing other markup languages.

Extensible Style Language (XSL)

A transformation language and a formatting language. The two languages operate independently of each other. The latter is used to describe how an XML document should be rendered when presented to a reader. Generally, a style sheet uses the XSL transformation language to transform an XML document into a new XML document that uses XSL formatting vocabulary which in turn is used to present the transformed document.

Extensible Style Language Transformations (XSLT)

The transformation part of XML's XSL support. Transformation language provides elements that define the rules for how one XML document is transformed into another XML document.

namespace

A hierarchical naming technique used within XML to identify the domain a given element name belongs. Using namespaces let XML documents make use of like named elements that are defined within different XML schemas. Namespaces help distinguish which element is being used within an XML document.

parser

A software process that is able to interpret the markup of an XML document.

schema

An externalized version of the rules that govern the formatting of an XML Document.

World Wide Web Consortium (W3C)

An Internet forum primarily dealing with information and technologies associated with the World Wide Web. The W3C communicates a collective understanding of items related to the past and future development of the World Wide Web.

W3C—see World Wide Web Consortium

XML—see Extensible Markup Language

XSLT—see Extensible Stylesheet Language Transformations

HP NonStop Terminology

Event Management System (EMS)

A subsystem within the NonStop operating system to generate, manage, and monitor events and the log files that these events produce. User-written applications interface with system log files using EMS.

Guardian

An environment that supports interactive or programmatic use with the NonStop operating system.

Guardian Application Execution Facility (GAEF)

An application execution environment built in conjunction with HP NonStop to make use of NonStop products such as Guardian, Pathway, TMF, and NonStop SQL/MP. Use of GAEF enables CA Gen generated applications run efficiently on a NonStop host.

Intelligent Device Support (IDS)

Intelligent Device Support (IDS) is the ability of an application to affect the operation of an application and to obtain data from an intelligent device through a user-written program providing device control. Application programmers can provide set of requester programs that establish an interface between the existing GAEF components and the intelligent devices.

Transaction Management Facility (TMF)

The Transaction Monitoring Facility manages the integrity and consistency of transactions including data recovery on behalf of applications. Normally these transactions involve access to a database.

Pathway

A group of tools (also called Transaction Server/MP, or TS/MP) that provides a transaction processing environment for terminal-based applications. Pathway tools enable development, installation, and management of online transaction processing applications. Pathway/TS is a transaction processing environment for transaction-based applications, using the run-time environment of TS/MP and TM/MP software.

PATHMON

The monitor process of a Pathway system.

Remote Server Call (RSC/MP)

NonStop RSC/MP software lets workstations invoke HP NonStop Transaction Services/MP (formerly TMF) software residing on NonStop servers. RSC/MP improves the performance of applications and maintains the ability to handle high transaction volumes. The HP NonStop platform uses RSC over Piccolo over TCP/IP as its network transport.

Requester

A host-based client in the NonStop Requester/Server model, which is NonStop terminology for the Client/Server model. Requesters are part of Pathway-based applications.

Server

A host-based server in the NonStop Requester/Server model, which is NonStop terminology for the Client/Server model. Servers are part of Pathway-based applications.

Setup Tool

A Pathway–based application that lets users install and deploy CA Gen-generated applications on a NonStop host.

TACL

The command interpreter provided with the NonStop operating system.

Terminal Control Process (TCP)

A terminal control process (TCP) program supplied by HP NonStop that monitors the terminal activity generated by traditional requesters and interprets and executes the screen programs. TCP also controls terminal I/O devices and the server process communication for the transaction processing applications.

Transaction Delivery Process (TDP)

A NonStop Remote Server Call (RSC/MP) process that, in a distributed processing environment, manages transactions between clients and servers.

Appendix B: Two-Phase Commit Processing

Certain server execution environments that are supported by CA Gen let multiple servers participate in a distributed transaction. A CA Gen application that is implemented as a distributed transaction extends the scope of a single unit-of-work to include more than one server process. The activity that is associated with coordinating the completion of processes that are involved in the distributed transaction is a process that is known as *two-phase commit*. A CA Gen DPS uses the two-phase commit mechanism of the execution environment.

The use of a two-phase commit process enforces the integrity of the distributed transaction and its data sources. The two-phase commit process obtains a consensus from all participants to determine the outcome of all work that is associated with the transaction.

This appendix describes the components of the execution environment that are involved in providing the two-phase commit processing. This appendix also briefly covers the actions that take place to complete a two-phase commit.

Two-Phase Commit

For more information about the formal two-phase commit description, see the *X/Open Distributed Transaction Processing: The XA Specification* in the following URL: http://www.opengroup.org/publications/catalog/c193.htm.

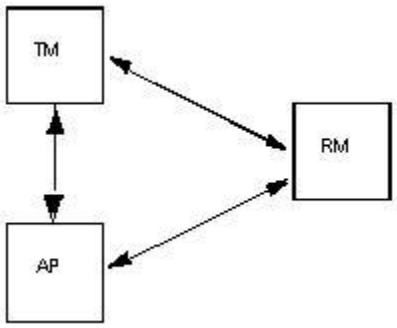
X/Open Distributed Transaction Processing

To understand how a two-phase commit works within a CA Gen Distributed Processing (DP) application, you can find it useful to understand how the X/Open definitions relate to CA Gen processing.

The X/Open Distributed Transaction Processing (DTP) model defines the following separate entities that participate in a distributed application:

- Application Program (AP)
- Transaction Manager (TM)
- Resource Manager (RM)

The following illustration shows the relationships among these three entities:



The Transaction Manager or TM is responsible for managing transactions on behalf of the application. The AP is the CA Gen server application code that executes under the control of a Transaction Manager. The CA Gen server application issues transactional statements to the Resource Manager or RM. The Resource Manager for CA Gen is the Database Management Software (DBMS) that manages access to the data as follows:

- The interface between the AP and RM is generally SQL. No specification is given for this interface. Servers open a connection to the RM through the XA API xa_open(). This establishes a connection to the RM for global transactions. At server shutdown, the XA API xa_close() closes the connection. These XA APIs are managed through the vendor software hosting the CA Gen server applications. For example, Tuxedo provides the tpopen() and tpclose() API call for use by the generated Tuxedo server.
- The execution environment hosting the CA Gen server applications manages the interface between the TM and RM. There are many procedural interfaces that are defined for XA-compliant RMs (for example, xa_open(), xa_start(), xa_end(), xa_prepare(), xa_commit(), and xa_recover()). It is through these interfaces that the TM communicates with the RM, on behalf of the AP.
- The interface between the AP and TM is mainly through transactional APIs provided by the server execution environment. For example, Tuxedo provides the following APIs; tpbegin(), tpcommit() and tpabort().

Two-Phase Commit Process

The Transaction Manager that is chosen to coordinate the commit of the distributed transaction is the one hosting the server application that begins the global transaction. As the server-to-server flows transmit between the CA Gen servers, a distributed transaction participant tree is built of all servers that are involved in the transaction. The server application that begins a global transaction is the server that commits or aborts the transaction.

Phase 1

Prepare—The first phase of a Two-Phase Commit involves the following actions:

1. The coordinating TM sends *prepare to commit* message to every process involved in the transaction.
2. A process returns a *prepared* message when it has determined validity of the work to be performed and can accomplish that work.
3. It is guaranteeing that it can commit.
4. It cannot decide to abort after preparing.
5. If a process cannot prepare, it must abort.
6. The process waits for permission to commit.
7. If an abort message is received at any time, the process must abort.

Phase 2

Commit—The second phase of a Two-Phase Commit involves the following actions:

1. The coordinating TM keeps track of *prepared* responses.
2. If any process aborts, the coordinator sends an *abort* message to all processes, which must then abort.
3. If all processes return the *prepared* response, the coordinator sends a *commit* message to all participants.
   - The processes receiving the commit message can now implement their part of the transaction.
   - The coordinator waits for all participants to acknowledge the commit.
4. When all participants have acknowledged their commits, the coordinating TM marks the transaction complete.

Appendix C: User Exits

There are various users exits invoked when processing a cooperative flow between a Distributed Processing Client (DPC) and a Distributed Processing Server (DPS).

CA Gen offers various choices of how to implement the parts of a Distributed Processing application, and therefore offers many user exits. An application developer is concerned with the set of exits that are invoked from the execution environments in which their applications components are deployed.

This appendix describes user exits that are invoked during the processing of a cooperative flow. It does not attempt to guide the user exits that are invoked from processing that is not involved in the processing of a cooperative flow.

The following table shows user exits falling into the following categories:

- User exits invoked by DPC (client processing)
- User exits invoked by communications utilities and middleware processing
- User exits invoked by DPS (server processing)

<table>
<tr><th colspan="6">Distributed Processing Applications</th></tr>
<tr><th colspan="2">Client Execution Environments</th><th colspan="4">Server Execution Environments</th></tr>
<tr><th colspan="2">DPC application</th><th colspan="2">Communications and Middleware</th><th colspan="2">DPS application</th></tr>
<tr><td>Generated Clients
Proxy Clients</td><td>Client Runtime</td><td>Utilities and Transport Protocols</td><td>TP Monitors</td><td>Server Runtime</td><td>Generated Servers</td></tr>
<tr><td>EABs</td><td>User Exits</td><td>User Exits</td><td>User Exits</td><td>User Exits</td><td>EABs</td></tr>
</table>

This section contains the following topics:

Client Side Exits (see page 132)
Communication Utilities and Middleware Exits (see page 134)
Server Side Exits (see page 143)

Client Side Exits

C/C++—GUI Runtime

The following table gives a brief description of each of the C/C++ - GUI Runtime exits.

Client Side: Language: C/C++ Function Area: GUI Runtime		
User Exit Name	**Source Code**	**Description**
WRSECTOKEN	wrexitn.c	Client Side Security exit
WRSECENCRYPT	wrexitn.c	Cooperative flow encryption exit (CFB only)
WRSECDECRYPT	wrexitn.c	Cooperative flow decryption exit (CFB only)
WRSRVRERROR	wrexitn.c	Synchronous Cooperative flow server error exit
WRASYNCSRVRERROR	wrexitn.c	Asynchronous Cooperative flow server error exit

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

C/C++—C and COM Proxy Runtime

The following table gives a brief description of the exits under C/C++ and COM Proxy Runtime.

Client Side: Language: C/C++ Function Area: C and COM Proxy Runtimes		
User Exit Name	**Source Code**	**Description**
WRSECTOKEN	proxyxit.c	Client Side Security exit
WRSECENCRYPT	proxyxit.c	Cooperative flow encryption exit (CFB only)
WRSECDECRYPT	proxyxit.c	Cooperative flow decryption exit (CFB only)

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

Java—Web Generation Client and Java Proxy Runtime

The source code for each of the Java user exits can be found in a file of the same name as the exit with a .java suffix.

Example:

The exit CFBDynamicMessageSecurityExit is contained within a file named CFBDynamicMessageSecurityExit.java.

The following table gives a brief description of the exits under Java – Internet Client/Java Proxy Client Runtime.

User Exit Name	Description
CFBDynamicMessageEncodingExit	The Java CFB encoding user exit
CFBDynamicMessageSecurityExit	The Java CFB client security user exit
CFBDynamicMessageEncryptionExit	The Java CFB client encryption user exit
CFBDynamicMessageDecryptionExit	The Java CFB client decryption user exit
SrvrErrorExit	The Web Generation Client to Server Flow Error exit. This exit is not available to Java proxy clients.

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

C#—ASP.NET Client and .NET Proxy Runtime

The source code for each of the C# user exits can be found in a file of the same name as the exit with a .cs suffix.

Example:

The exit CFBDynamicMessageSecurityExit is contained within a file named CFBDynamicMessageSecurityExit.cs.

The following table gives a brief description of the exits under C# - Client/C# Proxy Client Runtime.

User Exit Name	Description
CFBDynamicMessageSecurityExit	The C# CFB Client Security user exit
CFBDynamicMessageEncryptionExit	The C# CFB Client Encryption user exit

User Exit Name	Description
CFBDynamicMessageDecryptionExit	The C# CFB Client Decryption user exit
SrvrErrorExit	The C# ASP.NET Client to Server Flow Error exit

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

Communication Utilities and Middleware Exits

Communications Bridge Exits

All supplied Communications Bridge user exits are written using the C programming language. The following table briefly describes each of the exits:

Comm. Bridge: Language: C		
User Exit Name	**Source Code**	**Description**
eci_client_exit	ioeciclx.c	This exit is called to permit overriding the name of the target CICS System (as it is known to the CICS Universal Client), the specified ECI timeout value, and the specified CICS Mirror Transaction associated with the request.
DECRYPT	decrexit.c	This exit will decrypt the CFB from a client if the data in the CFB's Enhanced Security offset area is to be used and the target server environment has a derived Security_Level of Remote and if the CFB has been encrypted.
CIDE_INIT	cidexit.c	Conversation Instance Data exit - Initialize This exit disables or enables subsequent CIDE_PROC calls.
CIDE_PROC	cidexit.c	Conversation Instance Data exit—Process This exit lets modification of certain fields of the Conversation Instance data (for example UserId and Password), prior to the conversation supporting a cooperative flow being created.
GetTCPHostName	inetipux.c	Allows disabling the host name lookup that maps a connected client's IP address to a host name string.

Comm. Bridge: Language: C		
User Exit Name	**Source Code**	**Description**
RSCUserEntry	iorscclx.cxx	This exit supports multiple APIs that allow inspection and modification of user data and application data before it is sent to a target server on HP NonStop. User and application data received from the target server can also be inspected or modified before forwarding to the client application.

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

Client Manager Exits

All supplied Client Manager user exits are written using the C programming language. The following table briefly describes the Client Manager Exits:

Client Manager: Language: C		
User Exit Name	**Source Code**	**Description**
ci_cm_id	cicmclx.c	Client Manager unique ID (supports the use of Multi-Instance Client Manager). Used by the Client Manager application.
CIDE_INIT	cidexit.c	Conversation Instance Data—Initialize. Used to disable or enable subsequent CIDE_PROC calls.
CIDE_PROC	cidexit.c	Conversation Instance Data—Process. Used to modify certain fields of the Conversation Instance data (for example UserId and Password), prior to the conversation supporting a cooperative flow being created.
DECRYPT	decrexit.c	Decrypt CFB from GUI client if the data in Enhanced Security is to be used and the target server environment has a derived Security_Level of Remote and the CFB encryption flag indicates the CFB has been encrypted.
IEFDP_InitDir	iefdir.c	Client Manager Directory Services—Initialize (disable or enable subsequent Directory services calls)

Client Manager: Language: C		
User Exit Name	**Source Code**	**Description**
IEFDP_SearchDir	iefdir.c	Client Manager Directory Services—Search Implementation of the transaction server search algorithm.
IEFDP_CleanUpDir	iefdir.c	Client Manager Directory Services—Cleanup Allows for deallocation of resources, which can have been allocated to support directory services.
RSCUserEntry	iorscclx.cxx	This exit supports multiple APIs that allow inspection and modification of user data and application data before it is sent to a target server on HP NonStop. User and application data received from the target server can also be inspected or modified before forwarding to the client application.

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

C/C++—Middleware Communications Runtime, Client Side

The following table gives a brief description of the exits under C/C++ - Middleware Communications Runtime.

Client Side Middleware: Language: C/C++ Function Area: Communications			
Comm. Type	**User Exit Name**	**Source Code**	**Description**
CSU	CSUGetLibraryVersionName	csuglvn.cxx	Provides a mapping of specified library/dll name to version specific name if one exists, otherwise, the specified name value is returned.
Client Manager	ci_cm_id	cicmclx.c	Client Manager unique ID This ext supports the use of Multi-Instance Client Manager. Used by the Client Manager application.

Client Side Middleware: Language: C/C++

Function Area: Communications

Comm. Type	User Exit Name	Source Code	Description
Client Manager	CI_CM_DPC_Flow_ Complete_Comm_Error	cicmclx.c	Direct the client side runtime to retry a synchronous cooperative flow that fails with a specified communication error. Communication errors seen most often by GUI applications include 609, 619, and 629 failures.
TCP/IP	CI_TCP_DPC_DirServ_Exit	citcpclx.c	Programmatic runtime override of TCP/IP destination (host, port)
TCP/IP	CI_TCP_DPC_setupComm_ Complete	citcpclx.c	Directs the runtime to retry a cooperative flow request that failed during setup.
TCP/IP	CI_TCP_DPC_handleComm_Com plete	citcpclx.c	Directs the runtime to retry a cooperative flow request that failed during the non-setup related processing of a cooperative flow.
MQSeries	CI_MQS_DPC_Exit	cimqclex.c	Programmatic runtime override of various pieces of information used to interface with MQSeries.
MQSeries	CI_MQS_DynamicQName_ Exit	cimqclex.c	Override the default structure of Dynamic Reply to Queues.
MQSeries	CI_MQS_DPC_setupComm_Com plete	cimqclex.c	Direct the runtime to retry a cooperative flow request that failed during setup.
MQSeries	CI_MQS_DPC_handleComm_Co mplete	cimqclex.c	Direct the runtime to retry a cooperative flow request that failed during the non-setup related processing of a cooperative flow.
MQSeries	CI_MQS_MQShutdownTest	cimqclex.c	Override the default behavior of keeping the connection to the Queue Manager following the completion of a cooperative flow.
MQSeries	CI_MQS_DPC_ setReportOptions	cimqclex.c	Override the default report options set in the MQ Put Message Descriptor.

Client Side Middleware: Language: C/C++

Function Area: Communications

Comm. Type	User Exit Name	Source Code	Description
ECI	ci_eci_get_system_name	cieciclx.c	Provide the CICS system name, if one has not previously been provided by data obtained from the commcfg.ini file.
ECI	ci_eci_get_tpn	cieciclx.c	Provides override of the CICS Mirror Transaction used for processing the current request. The default Mirror Transaction is CPMI.
Tuxedo	ci_c_sec_set	cictuxwsx.c	Set user supplied security data into the security data fields located in Tuxedo's TPINIT structure.
Tuxedo	ci_c_user_data_out	cictuxwsx.c	Give the user the opportunity to inspect or modify the cooperative flow request buffer prior to Tuxedo sending the request to the target Tuxedo service. Invoked prior to the TPCALL for those clients connecting to Tuxedo servers residing on a separate host.
Tuxedo	ci_c_user_data_in	cictuxwsx.c	Give the user the opportunity to inspect or modify the cooperative flow request buffer on return from the target Tuxedo service. Invoked on return from the TPCALL for those clients connecting to Tuxedo servers residing on a separate host. Additionally, this exit allows the client to disconnect from the server following each flow.
Tuxedo	ci_c_user_data_out	cictuxx.c	Give the user the opportunity to inspect or modify the cooperative flow request buffer prior to Tuxedo sending the request to the target Tuxedo service. Invoked prior to the TPCALL for those clients connecting to Tuxedo servers residing on the same host.

Client Side Middleware: Language: C/C++ Function Area: Communications			
Comm. Type	**User Exit Name**	**Source Code**	**Description**
Tuxedo	ci_c_user_data_in	cictuxx.c	Give the user the opportunity to inspect or modify the cooperative flow request buffer on return from the target Tuxedo service. Invoked on return from the TPCALL for those clients connecting to Tuxedo servers residing on the same host. Additionally, this exit allows the client to disconnect from the server following each flow.
Web Services	CI_WS_DPC_Exit	ciwsclx.c	Programmatic runtime override of parameters baseURL and contextType for Web Service destination.
Web Services	CI_WS_DPC_URL_Exit	ciwsclx.c	Programmatic runtime override of URL for Web Service destination.

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

Java—Middleware Communications Runtime, Client Side

The source code for each of the Java user exits can be found in a file of the same name as the exit with a .java suffix. For example, the exit ECIDynamicCoopFlowExit is contained within a file named ECIDynamicCoopFlowExit.java.

The source code files for these exits can be found within the appropriate subdirectories under the CA Gen installation area %GENxx%Gen\classes\com\ca\genxx\exits.

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

The following table gives a brief description of the exits under Java – Middleware Communications Runtime:

Client Side Middleware: Language: Java **Function Area: Communications**		
Comm. Type	**User Exit Name**	**Description**
ECI	ECIDynamicCoopFlowExit	The Java ECI Cooperative flow user exit
EJBRMI	EJBRMISecurityExit	The Java EJBRMI client security user exit
EJBRMI	EJBRMIDynamicCoopFlowExit	The Java EJBRMI Cooperative flow user exit
MQSeries	MQSDynamicCoopFlowExit	The Java MQSeries Cooperative flow user exit
TCP/IP	TCPIPDynamicCoopFlowExit	The Java TCP/IP Cooperative flow user exit
Tuxedo	TUXDynamicSecurityExit	The Java Tuxedo client security user exit
Tuxedo	TUXDynamicCoopFlowExit	The Java Tuxedo Cooperative flow user exit
Web Services	WSDynamicCoopFlowExit	The Java Web Services Cooperative flow user exit

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

C#—Middleware Communications Runtime, Client Side

The source code for each of the C# user exits can be found in a file of the name of the exit with a .cs suffix.

Example:

The exit TCPIPDynamicCoopFlowExit is contained within a file named TCPIPDynamicCoopFlowExit.cs.

The source code files for these exits can be found within the appropriate subdirectories under the CA Gen installation area, %GENxx%Gen\.net\exits\src. The following table gives a brief description of the exits under C# - Middleware Communications Runtime.

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

Client Side Middleware: Language: C# Function Area: Communications		
Comm. Type	**User Exit Name**	**Description**
.NET	NETDynamicCoopFlowSecurityExit	.Net cooperative flow security exit
.NET	NETDynamicCoopFlowExit	.Net cooperative flow exit
TCP/IP	TCPIPDynamicCoopFlowExit	TCP/IP C# Cooperative flow exit
Web Services	WSDynamicCoopFlowExit	Web Services C# Cooperative flow exit

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

CICS TCP/IP Direct Connect Exits

The CA Gen CICS TCP/IP Direct Connect product consisting of the TILSTNR and TICONMGR has been stabilized and only the CICS Sockets Server Listener, TISRVLIS, implementation is provided.

The following table describes the Assembler exits invoked by the Sockets Server Listener program:

CICS TCP/IP Direct Connect (TISRVLIS): Language: Assembler		
User Exit Name	**Source Code**	**Description**
TIRSLEXT	SAMPLIB/TIRSLEXT	CICS Sockets Server Listener exit
TIRSLTMX	SAMPLIB/TIRSLTMX	CICS Sockets Server Listener Timeout exit

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

WebSphere MQ Transaction Dispatcher for CICS (TDC) Exit

The CA Gen WebSphere MQ Transaction Dispatcher product provides the following user exit. This exit is implemented in COBOL. The following table briefly describes this exit:

WebSphere MQ TDC: Language: COBOL		
User Exit Name	**Source Code**	**Description**
TIRMQTDX	SAMPLIB/TIRMQTDX	TDC Parameter exit

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

IMS TCP/IP Direct Connect Exits

The CA Gen IMS TCP/IP Direct Connect product provides the following set of user exits. These exits are implemented in Assembler. The following table briefly describes each of the exits:

IMS TCP/IP Direct Connect (CAGRITCP): Language: Assembler		
User Exit Name	**Source Code**	**Description**
CAGRITD	SAMPLIB/CAGRITD	IMS TCP/IP Destination ID exit
CAGRITDC	SAMPLIB/CAGRITDC	IMS TCP/IP Decryption exit
CAGRITSC	SAMPLIB/CAGRITSC	IMS TCP/IP Security exit

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

Server Side Exits

C—Transaction Enabler, Server Manager Runtime Exits

The following table gives a brief description of the C exits. These user exits are invoked from the Transaction Enabler components and Distributed Processing generated for C.

Server Side: Language: C	**Function Area: Server Manager/Server Runtime**	
User Exit Name	**Source Code**	**Description**
SRVRERROR	tirserrx.c	Server to server error exit that is invoked when an error condition occurs during server-to-server flow.
TIRDCRYP	tirdcryp.c	Server side cooperative flow decryption exit (CFB only)
TIRELOG	tirelog.c	Server side Error logging and Error token creation exit.
TIRMTQB	tirmtqb.c	Runtime Message Table Exit number to error text translation exit. The passed in error number is used to obtain a standardized error message string defined within the exit.
TIRNCRYP	tirncryp.c	Server side cooperative flow encryption exit (CFB only)
TIRSECR	tirsecr.c	Security check interface. This exit provides an opportunity for a server to validate the user ID as returned from TIRUSRID.
TIRSECV	tirsecv.c	Validates security given Client Userid, Client Password, Client Security Token retrieved from the security offset of an enhanced CFB. This exit will not be called if the CFB is not an enhanced security CFB.
TIRSYSID	tirsysid.c	Allows modification of the system ID. This user exit returns a default system ID, the value of which depends on the platform on which the server is executing.
TIRUSRID	tirusrid.c	Allows modification of the user ID. This routine returns a default user ID, the value of which depends on the platform on which the server is executing. The returned ID is passed to TIRSECR for validation.
TIRXINFO	tirxlat.c	This user exit provides information about the local environments os, codepage and default padding character. The os and codepage are passed to the TIRXLAT exit.

Server Side: Language: C Function Area: Server Manager/Server Runtime		
User Exit Name	**Source Code**	**Description**
TIRXLAT	tirxlat.c	This national language translation exit allows conversion of textual data based on from/to codepage and operating system information.
USEREXIT	dpsuetdm	This user exit supports multiple APIs for access to user data and transaction data when the target server resides on an HP NonStop host. USEREXIT is called twice–just after the request data is received from the Distributed Processing Client (DPC), and again just before response data is returned to the DPC.

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

COBOL—CICS and IMS Server Manager Runtime Exits

The following table describes the COBOL exits:

Server Side: Language: COBOL Function Area: Server Manager/Server Runtime		
User Exit Name	**Source Code**	**Description**
TIRDCRYP	SAMPLIB/TIRDCRYP	Server side cooperative flow decryption exit (CFB only)
TIRNCRYP	SAMPLIB/TIRNCRYP	Server side cooperative flow encryption exit (CFB only)
TIRSECR	SAMPLIB/TIRSECR	Security interface exit. This exit provides an opportunity for an application to validate the user ID as returned from TIRUSRID.
TIRSECV	SAMPLIB/TIRSECV	Validates security given Client Userid, Client Password, Client Security Token retrieved from the security offset of an enhanced CFB. This exit is called for every cooperative flow.
TIRSYSID (CICS)	SAMPLIB/TIRCSYS	Allows modification of the system ID. This user exit returns a default system ID, the value of which depends on the platform on which the server is executing.

Server Side: Language: COBOL Function Area: Server Manager/Server Runtime		
User Exit Name	**Source Code**	**Description**
TIRSYSID (IMS)	SAMPLIB/TIRISYS	Allows modification of the system ID. This user exit returns a default system ID, the value of which depends on the platform on which the server is executing.
TIRUSRID (CICS)	SAMPLIB/TIRCUSR	Server User ID
TIRUSRID (IMS)	SAMPLIB/TIRIUSR	Server User ID
TIRALLOC	SAMPLIB/TIRALLOC	Server-to-Server Allocate Conversation
TIRPROUT (CICS)	SAMPLIB/TIRCROUT	Server-to-Server Trancode Routine
TIRPROUT (IMS)	SAMPLIB/TIRIROUT	Server-to-Server Trancode Routine
TIRPTOKN	SAMPLIB/TIRPTOKN	Server-to-Server Security Token CA Generation
TIRCSGN	SAMPLIB/TIRCSGN	Direct Connect for CICS Server Signon
TIRELOG	SAMPLIB/TIRELOG	Server Error Logging/Error Token Creation
TIRSIPEX	SAMPLIB/TIRSIPEX	CICS Sockets Server exit

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

Assembler—CICS and IMS Server Manager Exit

The following table describes these exits.

Server Side: Language: COBOL Function Area: Server Manager /Server Runtime		
User Exit Name	**Source Code**	**Description**
TIRXINFO	SAMPLIB/TIRXINFO	National Language Translation

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

C#—.NET Server Manager Runtime Exits

The source code for each of the C# user exits can be found in a file of the same name as the exit with a .cs suffix. For example, the exit SrvrErrorExit is contained within a file named SrvrErrorExit.cs.

These user exits are located within appropriate subdirectories within the CA Gen installation area. Typically %GENxx%Gen\.net\exits\src. C# exits are supported on Windows platforms only.

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

The following table gives a brief description of the exits present in this exit:

User Exit Name	Description
SrvrErrorExit	C# Server to Server Error exit
SecurityValidationExit	C# Server Security Validation

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

C/C++—Middleware Exits Supporting Server-to-Server Flows

The following table briefly describes this exit.

Server Side Middleware: Language: C/C++ Function Area: Communications			
Comm. Type	**User Exit Name**	**Source Code**	**Description**
MQSeries	CI_MQS_DPC_Exit	cimqsvex.c	Programmatic runtime override of various pieces of information used to interface with MQSeries.
MQSeries	CI_MQS_DynamicQName_Exit	cimqsvex.c	Override the default structure of Dynamic Reply to Queues.
MQSeries	CI_MQS_DPC_setReportOptions	cimqsvex.c	Override the default report options set in the MQPut message descriptor.
Tuxedo	ci_s_user_data_out	cistuxx.c	Give the user the opportunity to inspect or modify the cooperative flow request buffer prior to it being passed to the server manager.

Server Side Middleware: Language: C/C++ Function Area: Communications			
Comm. Type	**User Exit Name**	**Source Code**	**Description**
Tuxedo	ci_s_user_data_in	cistuxx.c	Give the user the opportunity to inspect or modify the cooperative flow request buffer on return from the server manager.
Tuxedo	ci_s_post_begin	cistuxx.c	This user exit is called after Tuxedo's tpbegin() is invoked.
Tuxedo	ci_s_pre_end	cistuxx.c	This user exit is called just prior to a transaction being committed using Tuxedo's tpcommit().
Tuxedo	ci_s_post_end	cistuxx.c	This user exit is called just after a transaction has been committed using Tuxedo's tpcommit().
Tuxedo	ci_s_post_svrinit	cistuxx.c	This user exit is called during server initialization by Tuxedo's tmboot processing.
Tuxedo	ci_s_post_svrdone	cistuxx.c	This user exit is called during server shutdown by Tuxedo's tmshutdown processing.

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

Java—Middleware Exits Supporting Server-to-Server Flows

The source code for each of the Java user exits can be found in a file of the same name as the exit with a .java suffix. For example, the exit CFBDynamicMessageEncodingExit is contained within a file named C CFBDynamicMessageEncodingExit.java. The following table gives a brief description of the exits under Java – EJB Middleware Sever-to-Server Runtime.

Server Side Middleware: Language: Java Function Area: Communications		
Comm. Type	**User Exit Name**	**Description**
	CFBDynamicMessageEncodingExit	Java CFB Encoding exit

Server Side Middleware: Language: Java	Function Area: Communications	
Comm. Type	**User Exit Name**	**Description**
EJBRMI	EJBRMISecurityExit	Java EJBRMI Security Exit
EJBRMI	EJBRMIContextExit	Java EJBRMI Context exit
	SrvrErrorExit	Java Server to Server Error exit

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

C#—Middleware Exits Supporting Server-to-Server Flows

The source code for each of the C# user exits can be found in a file of the same name as the exit with a .cs suffix. For example, the exit COMPLUSDynamicCoopFlowExit is contained within a file named C COMPLUSDynamicCoopFlowExit.cs. The following table briefly describes the exits present under this exit.

Comm. Type	User Exit Name	Description
COMPLUS	COMPLUSDynamicCoopFlowExit	COMPLUS cooperative flow exit
COMPLUS	COMPLUSDynamicCoopFlowSecurityExit	COMPLUS cooperative flow security exit

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

Customizing and Installing z/OS User Exits

The easiest way to customize a user exit is to copy and modify the code that is supplied by CA. Always read the source code of the exit and the additional information included in comment lines. These lines are placed where you would need to enter code to customize the exit for a specific need or function.

The steps that are involved in customizing any user exit are as follows:

- Copy the default exit from the CA Gen SAMPLIB library to one of your libraries.
- Add the desired new code to the copied exit. Do not change the 'Program ID' (entry point) or argument list that is passed to the exit.

Most user exits are called with RUNTIME-PARM1 and RUNTIME-PARM2 as the first two parameters. Under IMS, the RUNTIME-PARM parameters are mapped to the IO-PCB and ALT-IOPCB, respectively. If an exit is modified to use the IO-PCB, the ALT-IOPCB, or both, remember to remove the corresponding RUNTIME-PARM from the LINKAGE SECTION and the PROCEDURE DIVISION USING statement.

Under CICS, the RUNTIME-PARM parameters are mapped to the CICS EXEC Interface Block (EIB) and the COMMAREA, respectively. If an exit is modified to use DFHEIBLK, FHCOMMAREA, or both remember to remove the corresponding RUNTIME-PARM from the LINKAGE SECTION and the PROCEDURE DIVISION USING statement. In addition, remove these parameters if the modified exit is processed by the CICS translator as the CICS translator automatically includes a reference to DFHEIBLK and DFHCOMMAREA.

Information about the presence and use of RUNTIME-PARM parameters is included in the SAMPLIB member for each exit and is covered in this chapter if relevant.

The remaining parameters are unique to each exit and is not modified.

Ensure that the new runtime modules are placed in the proper data sets as follows:

- CICS – in the CICS DFHRPL concatenation and is New Copied into CICS.
- IMS – in the steplib concatenation.

As delivered, the Exits contain no SQL and no DBRMs. If a User Exit calls another routine that does database access, add that routine's DBRM to the installation control file so the routine can be found. The best way to accomplish this is to modify the Installation CLIST, TICINSTX, so that it adds the DBRM to the DBRM list.

The following table identifies the z/OS JCL procedures that are used to compile and bind each of the z/OS Server Manager User Exits into their respective DLLs. These JCL procedures are located in the CA Gen SAMPLIB library.

Each DLL contain other user exits that are not directly in support of the processing of a z/OS Distributed Processing Server (DPS). Additionally, the list of z/OS user exits also be used to support z/OS block mode and batch operation.

JCL Procedure	DPS User Exit Name	Description
MKORUN	TIRDCRYP, TIRNCRYP, TIRSECR, TIRSECV, TIRSYSID, TIRUSRID, TIRALLOC, TIRPROUT, TIRPTOKN, TIRCSGN, TIRELOG, TIRSIPEX	Incorporates customized user exits into the following z/OS DLLs: TIRORUNC TIRODCMC TIRORUNI TIRORUNT TIRBRUNB TIRBRUNT
MKORUNX		Incorporates customized user exits into the following z/OS DLLs: TIRTERAZ TIRMTQBZ TIRPRFQZ
MKCRUN	TIRXINFO	Incorporates customized user exits into the following z/OS DLLs: TIRCRUNC TIRCRUNI

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

Customizing and Installing NonStop User Exits

User exits are distributed as part of the NonStop IT and are delivered as a source archive named UEXITSC. The source archive contains more user exits than will be documented within this appendix. This appendix covers only those exits called during the processing of a cooperative flow. Non cooperative flow user exits are documented in the *User Exit Reference Guide*.

NonStop Runtime User Exits

All runtime user exits are provided in both source and object format (in the form of libraries called UEXITCO and DPSUECO) and are loaded onto the target system when the IT is installed. You can use these user exits without changing them because the object format runtime user exits are compiled and are ready to be linked into applications.

Modifying User Exits

User exit source code is provided in the C language so that you can modify them to meet custom site requirements. To assist with the lifecycle of user exits, the TACL macro MKEXITS is provided. This macro can be found in the subvolume where the IT has been installed. You can use MKEXITS to perform the following tasks:

- Extract all user exit source files from the UEXITSC source archive
- Edit a specific user exit source file
- Compile a specific user exit source file
- Rebuild the user exit library UEXITCO and DPSUECO

Note: When working with user exits, link all user exits for a particular library (the set of runtime user exits in UEXITCO or the set of distributed processing server user exits in DPSUECO) regardless of the number of changes as the NLD linker program does not let you individually link one exit at a time. Therefore, track source code changes across all the user exits to prevent unexpected functionality to be included in the user exit libraries.

Important! Failure to link all user exits result in application failures due to unresolved externals at execution time.

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

The following server-side user exits are documented in the *User Exit Reference Guide*:

Name	Description
TIRDCRYP	Server Decryption Exit
TIRELOG	Server Error Logging and Error Token Creation Exit
TIRMTQB	Runtime Message Table Exit
TIRNCRYP	Server Encryption Exit
TIRSECR	Security Check Interface Exit
TIRSECV	Server Client Security Validation Exit
TIRSYSID	Server System ID Exit
TIRUSRID	Server User ID Exit
TIRXINFO	Server System Code Page and Operating System Id
TIRXLAT	National Language Translation User Exit
USEREXIT	NonStop RSC/MP Distributed Processing Flow Data Access Exit

Note: Not all of the user exits in the UEXITSC archive are documented in this appendix. For this user exits not directly involved in processing a cooperative flow, see the *User Exit Reference Guide*.

Appendix D: Comm Config Files

For more information about the use of the three comm config files, see the "Overriding Communications Support at Execution Time (see page 57) chapter.

commcfg.ini

The commcfg.ini file provides the following users the ability to override the generated communication type and communications parameters at the time a given flow is executed.

- MFC GUI clients
- COM Proxy clients
- C Proxy clients

```
#
# CA Gen
# Copyright (c) 2013 CA. All rights reserved.
#
#   commcfg.ini - Communications configuration file for cooperative clients.
#
# This file should be modified to associate TRANCODEs with the locations of
# the target cooperative servers. It can also be used to control some runtime
# behaviors of cooperative client runtimes.
#
# Refer to the FILE LOCATION and MULTIPLE OVERRIDE SUPPORT section below for
# possible locations and overrides of this file
#
##############  TRANCODES #################
#
# To associate TRANCODEs with hosts, an entry must exist in this file to point
# to the associated host. The format of a line within this file is dependant
# on what communication type will be used.  <...> is required, {...} is optional.
# For TCP/IP the format is:
#           <TRANCODE> TCP <host> <service/port> {connection_persistence}
#              connection_persistence:  optional, controlled by client runtime
#                           Not Specified  -  for persistent connection
#                                      'Y' -  for persistent connections
#                                      'N' -  for non persistent connections
# For IMS TCP/IP Direct Connect the format is:
#           <TRANCODE> ITP <host> <service/port> {connection_persistence}
#              connection_persistence:  optional, controlled by client runtime
#                           Not Specified  -  for persistent connection
#                                      'Y' -  for persistent connections
#                                      'N' -  for non persistent connections
# For MQSeries (MQS) the format is:
#           <TRANCODE> MQS <Queue Name> <Queue Manager> {Reply Queue Name}
# For MQSeries (MQI-client) the format is:
#           <TRANCODE> MQSC <Queue Name> <Queue Manager> {Reply Queue Name}
# For ECI the format is: (Use the ECI exit to control the system, hostName, portNumber)
#           <TRANCODE> ECI <ECI System Name> {hostName} {portNumber}
# For Client Manager the format is:
#           <TRANCODE> CM
# For EJBRMI, the format is:
#           <TRANCODE> EJBRMI {Initial Factory Class} {Name Manager URL}
# For NET the format is:
#           <TRANCODE> NET <hostName> <portNumber> <protocolCode>
#                     protocolCode can be : 'S' for Http/Soap
#                                           'B' for Http/Binary
# For Web Services the format is:
#           <TRANCODE> WS <baseURL> <contextType>
#                   baseURL: Scheme, Domain and Port of a Web Service end point URL
#                           ex: http://<hostname>:CA Portal
```

```
#                    contextType: part of the path of a Gen Web Service end point
URL
#                         'P' to use ProcedureStep Name (with WebLogic)
#                         'L' to use LoadModule Name (everywhere else)
#
# It is possible to declare "wild-card" style TRANCODEs.  This is
# accomplished by ending the TRANCODE with a '*'.  An example use is:
#
#     ABCD TCP myhost1 2008 N
#     ABC* TCP myhost2 2008 Y
#     A*   TCP myhost3 2008
#     *    TCP myhost4 2008
#
# This example would:
#   - Route an exact match of ABCD to myhost1 using non persistent socket
#     connections
#   - Route all TRANCODEs starting with ABC (but not ABCD) to myhost2 using
#     persistent socket connections
#   - Route all TRANCODEs starting with A (but not ABC) to myhost3 using
#     persistent socket connections
#   - Route all other TRANCODEs to myhost4 using persistent socket connections
#
############## CACHING ###################
#
# The commcfg.ini file may also be used to control how the client runtimes
# cache this ini file.  By default caching is not performed and the file is
# reread and reparsed for each flow out of the running process.  This allows
# file changes to be reflected immediately.
#
# That default behavior may not be ideal in all environments.  Therefore the
# special token 'CACHETIMEOUT' can be set to control the number of seconds to
# wait between rereading of the file.  All flows that occur between the read
# and the timeout use the cached version of the file in memory.  The first flow
# after the timeout has expired will force the file to be reread and the timeout
# to restart.  An example is:
#
#    CACHETIMEOUT   0     (no timeout, default)
#    CACHETIMEOUT   180   (3 minute timeout)
#    CACHETIMEOUT   NEVER (cache will never be reread)
#
# The timeout is only useful within one process.  The NEVER setting will require
# the process to be stopped and the client runtime DLL to be unloaded before
# rereading the file.
#
############## TRACING ##################
#
# The commcfg.ini file can be used to enable and disable tracing within the
# client runtimes.  Use the special token 'CMIDEBUG' to enable and disable the
# tracing.  Examples of the use of this token are:
```

```
#
#    CMIDEBUG ON  (turn tracing on)
#    CMIDEBUG OFF (turn tracing off)
#
# The tokens can appear multiple times within the file, but the last one
# encountered will be the setting that is in effect.  The trace will be
# written to a file called trace-<procname>-<procid>.out (where <procname>
# is the application name and <procid> is the application's process id)
# within the %USERPROFILE%\AppData\Local\CA\Gen 8.5\logs\client directory.
#
############## FILE LOCATION ###############
########## MULTIPLE OVERRIDE SUPPORT #######
#
#
# By default there exists only one copy of this file. The installed location
# is the CA Gen installation directory. As there is only one file all client
# applications must use this same file.
# Alternatively, a Client Application will search for this file in the following
# order:
#
# 1) %COMMCFG_HOME% - The directory value contained within this environment
#                     variable
# 2) %USERPROFILE%\AppData\Local\CA\Gen 8.5\cfg\client
# 3) %ALLUSERSPROFILE%\CA\Gen 8.5\cfg\client
# 4) %PATH% - In each of the path components within this environment variable
#
# Unix/Linux  only
# 1) $COMMCFG_HOME - The directory value contained within this environment
#                    variable
# 2) $PATH - In each of the path components within this environment variable
#
#
# If the file is not found in one of these places, it is assumed to not exist.
# In case there is more than one copy of the file, the order of precedence
# is as described above. The use of this search order is defined below:
#
# %COMMCFG_HOME%, $COMMCFG_HOME:
#    The environment variable COMMCFG_HOME can be set to point to a
#    directory containing the commcfg.ini file to be used by the invoking
#    application. This will allow customization on an application by
#    application basis.
#
# %USERPROFILE%\AppData\Local\CA\Gen 8.5\cfg\client (Windows only):
#    If the commcfg.ini file is stored in this location, it is accessable by
#    all applications executed by a particular user on the system.
#
# %ALLUSERSPROFILE%\CA\Gen 8.5\cfg\client (Windows only):
#    If the commcfg.ini file is stored in this location, it is accessable by
#    all applications executed by all users on the system.
```

```
#
# %PATH%, $PATH:
#    If the commcfg.ini file has not been found in the other 3 locations above,
#    all paths listed in the PATH environment variable will be searched. By
#    default, CA Gen is part of the PATH environment variable.
#
##########################################
#
```

commcfg.properties

The commcfg.properties file provides the following users the ability to override the generated communication type and communications parameters at the time a given flow is executed.

- Java Web Generation clients
- Java Web View clients
- Java Proxy clients
- Java EJB servers flowing to other CA Gen servers

```
#
# CA Gen
# Copyright (c) 2013 CA. All rights reserved.
#
#   commcfg.properties - Communications configuration file for Gen.
#
# This file should be modified to associate TRANCODEs with the locations of
# Gen servers.  It can also be used to control some runtime behaviors of
# a Gen application/proxy. This file must reside in a location where the
# JVM can load it at application execution.
#
##############  TRANCODES #################
#
# To associate TRANCODEs with hosts, an entry must exist in this file to point
# to the associated host. The format of a line within this file is dependant
# on what communication type will be used.  <...> is required, {...} is
# optional.
# For TCP/IP the format is:
#           <TRANCODE>=TCP <host> <service/port> {connection_persistence}
#              connection_persistence:  optional, controlled by client runtime
#                           Not Specified  -  for persistent connection
#                                      'Y' -  for persistent connections
#                                      'N' -  for non persistent connections
# For IMS TCP/IP Direct connect the format is:
#           <TRANCODE>=ITP <host> <service/port> {connection_persistence}
#              connection_persistence:  optional, controlled by client runtime
#                           Not Specified  -  for persistent connection
#                                      'Y' -  for persistent connections
#                                      'N' -  for non persistent connections
# For MQSeries (MQS) the format is:
#           <TRANCODE>=MQS {Queue Manager} {Queue Name} {Reply Queue Name}
#
#     NOTE: Unlike the C++ implementation of commcfg.ini, the Java implementation
#           of commcfg.properties does not differentiate between MQSeries local
#           verses remote queue manager within this property. Thus the property
#           values of MQS and MQSC are functionally identical. The local verses
#           remote queue manager distinction is handled via the MQChannel and
#           MQHostname properties located in the MQSeries section of this file.
#           Please note the positional ordering of Queue Manager and Queue Name
#           must be maintained.
#
# For ECI the format is:
#           <TRANCODE>=ECI <protocol> <address> CA Portal <system> \
#                                {Client Security Class} {Server Security Class}
#           (For ECI, <protocol> may be local, tcp, auto, http, https, or ssl.)
#
#     NOTE: Unlike the C++ implementation of commcfg.ini, the Java implementation
#          of commcfg.properties allows the CICS transaction id to be set to a value
#           other than CPMI. More information about this can be found in the ECI
```

```
#           section of this file.
#
# For EJBRMI the format is:
#           <TRANCODE>=EJBRMI  {Initial Factory Class} {Name Manager URL}
#
# For Web Services the format is:
#           <TRANCODE>=WS <baseURL> <contextType>
#                   baseURL: Scheme, Domain and Port of a Web Service end point URL
#                          ex: http://<hostname>:CA Portal
#                     contextType: part of the path of a Gen Web Service end point
URL
#                          'P' to use ProcedureStep Name (with WebLogic)
#                          'L' to use LoadModule Name (everywhere else)
#
# It is possible to declare "wild-card" style TRANCODEs.  This is
# accomplished by ending the TRANCODE with a '*'.  An example use is:
#
#     ABCD=TCP myhost1 2008 N
#     ABC*=TCP myhost2 2008 Y
#     A*=TCP myhost3 2008
#     *=TCP myhost4 2008
#
# This example would:
#   - Route an exact match of ABCD to myhost1 using non persistent socket
#     connections
#   - Route all TRANCODEs starting with ABC (but not ABCD) to myhost2 using
#     persistent socket connections
#   - Route all TRANCODEs starting with A (but not ABC) to myhost3 using
#     persistent socket connections
#   - Route all other TRANCODEs to myhost4. using persistent socket
#     connections
#
# An MQSeries example
#     ABCD=MQS myQmgr myLocalQ
#     ABC*=MQS myQmgr myLocalQ
#     *=MQS
#
# This example would:
#   - Route an exact match of ABCD to queue manager myQmgr, local queue myLocalQ
#   - Route all TRANCODEs starting with ABC (but not ABCD) to myQmgr/myLocalQ
#   - Route all other TRANCODEs to myQmgr
#
##########################################

############## CACHING ###################
#
# The commcfg.properties file may also be used to control how the Gen runtimes
# cache this file.  By default caching is not performed and the file is
# reread and reparsed for each flow out of the running application.  This allows
```

```
# file changes to be reflected immediately.
#
# That default behavior may not be ideal in all environments.  Therefore the
# special token 'CACHETIMEOUT' can be set to control the number of seconds to
# wait between rereading of the file.  All flows that occur between the read
# and the timeout use the cached version of the file in memory.  The first flow
# after the timeout has expired will force the file to be reread and the timeout
# to restart.  An example is:
#
#    CACHETIMEOUT=0     #(no timeout, default)
#    CACHETIMEOUT=180   #(3 minute timeout)
#    CACHETIMEOUT=NEVER #(cache will never be reread)
#
# The timeout is only useful within one process.  The NEVER setting will require
# the process to be stopped and the runtime to be unloaded before
# rereading the file.
#
# NOTE: Depending on how the commcfg.properties file is deployed in the
# CLASSPATH of the JVM and what the behaviors of the JVM class loader are, the
# file may never be reloaded regardless of the CACHETIMEOUT setting.  Basically, the
# JVM may make its own local cache of the file after loading it once and would
# then simply never reread it no matter how frequently the application code requested
# it.
#
########################################
#CACHETIMEOUT=0

############## TRACING #################
#
# The commcfg.properties file can be used to enable and disable tracing within
# the proxy runtimes.  Use the special token 'CMIDEBUG' to enable and disable
# the tracing.  Examples of the use of this token are:
#
# To turn tracing off. Default.
#        CMIDEBUG=OFF
# To turn tracing on and write to the default file: trace.out)
#        CMIDEBUG=ON
# To turn tracing on, and write to system: standard output, which
# might be the console, a command shell, or a service logfile depending
# on how the program was started.
#        CMIDEBUG=ON SYSTEM
# To turn tracing on and write to user specifiled file. Please make sure
# that the you have read/write permission for the same.
#        CMIDEBUG=ON FILENAME
#
# The traces written to a file will be written to a file called trace.out.
# The traces written to the system will be written to standard output, which
# might be the Java Console, a command shell, or a service logfile depending
# on how the program consuming the proxy was started.
```

```
#
########################################
#CMIDEBUG=ON

############## MQSeries #################
#
# The commcfg.properties file can be used to modify some of the environmental
# settings used by MQSeries.  In particular, these set any of the
# MQEnvironment settings.  Optionally these settings may be accomplished
# programmatically via the MQSDynamicCoopFlowExit class.
#
# See the MQSeries documentation for the proper values to be used for the
# following settings.
#
#   Note:    The MQChannel and MQHostname properties are required if the target
#            Queue manager is located on a remote system. Failure to specify
#            these properties will result in a connection attempt to a locally
#           defined Queue manager. This local connection may fail if the MQSeries
#          product is not properly configured with a local Queue manager defined.
#            One symptom of this failure is MQSeries runtime dll load failures
#            if the MQSeries server software is not installed. If all Queue
#            managers are to located on remote machines, only MQSeries Client
#            software is required.
#            MQPort needs to be set to the TCP port number as defined within the
#            target queue manager configuration. A default value of 1414 will be
#            used if not overridden here.
#
########################################
#MQChannel=myChannel
#MQHostname=localhost
#MQPort=1414
#MQUserID=myUser
#MQPassword=myPassword

############## ECI #################
#
# Gen applications use the ECI_call_type of ECI_SYNC for the ECIRequestType of
# the ECI cooperative flow. An ECI call results in a CICS Distributed Program Link
# (DPL) call using the CICS provided Mirror Program (DFHMIRS) to invoke the requested
# Gen server. Gen ECI cooperative flows pre-set the transaction id that CICS uses to
# execute the Mirror Program to CPMI.
# In some cases it may be desirable to use a transaction id other then CPMI to refer
# to refer to the CICS Mirror Program (DFHMIRS). Setting ECIUseSyncTpn=Yes will
# result in the trancode associated with the target server application to be used as
# the transid in place of CPMI.
#
# NOTE:
# If ECIUseSyncTPN flag is used the relevant trancode must be defined in CICS to point
# to the DFHMIRS program and use profile DFHCICSA.
```

```
# This trancode cannot be shared by Gen applications using ECI and non-ECI cooperative
# flows.
#
# See the ECI documentation for the uses of the various request types.
#
########################################
#ECIUseSyncTpn=YES


############ TUXEDO / JOLT ##############
#
# The commcfg.properties file can be used to set the Jolt System Network Address
# (JSL_NETWORK_ADDRESS) to point to the Remote System & Port Number hosting the JSL.
#
# This value in turn can be overridden by modifying the Security Exit that is
# invoked prior to the flow.
#
########################################
#JSL_NETWORK_ADDRESS=//localhost:portNumber
```

commcfg.txt

The commcfg.txt file provides the following users the ability to override the generated communication type, and communications parameters at the time a given flow is executed.

- ASP.NET Web Generation clients
- .NET Proxy clients
- .NET Servers flowing to other CA Gen servers

```
# CA Gen
# Copyright (c) 2013 CA. All rights reserved.
#
#   commcfg.txt - Communications configuration file for CA Gen .NET support.
#
# This file should be modified to associate TRANCODEs with the locations of
# CA Gen servers.  It can also be used to control some runtime
# behaviors of an CA Gen application/proxy.
#
# The file must be placed in the application private path, the
# application base directory, a subdirectory under the application base
# directory.
#
##############  TRANCODES #################
#
# To associate TRANCODEs with hosts, an entry must exist in this file to point
# to the associated host. The format of a line within this file is dependant
# on what communication type will be used.  <...> is required, {...} is
# optional.
# For TCP/IP the format is:
#           <TRANCODE>=TCP <host> <service/port> {connection_persistence}
#              connection_persistence:  optional, controlled by client runtime
#                           Not Specified  -  for persistent connection
#                                      'Y' -  for persistent connections
#                                      'N' -  for non persistent connections
#
# For IMS TCP/IP Direct connect the format is:
#           <TRANCODE>=ITP <host> <service/port> {connection_persistence}
#              connection_persistence:  optional, controlled by client runtime
#                           Not Specified  -  for persistent connection
#                                      'Y' -  for persistent connections
#                                      'N' -  for non persistent connections
#
# For MQSeries (MQS) the format is:
#           <TRANCODE>=MQS {Queue Manager} {Queue Name} {Reply Queue Name}
# For MQSeries (MQI-client) the format is:
#           <TRANCODE>=MQSC {Queue Manager} {Queue Name} {Reply Queue Name}
#
# For NET the format is
#           <TRANCODE>=NET <hostName> <portNumber> <protocolCode>
#                      protocolCode can be : 'S' for Http/Soap
#                                            'B' for Http/Binary
# For Web Services the format is:
#           <TRANCODE>=WS <baseURL> <contextType>
#                  baseURL: Scheme, Domain and Port of a Web Service end point URL
#                         ex: http://<hostname>:CA Portal
#                    contextType: part of the path of a Gen Web Service end point
URL
#                          'P' to use ProcedureStep Name (with WebLogic)
```

```
#                          'L' to use LoadModule Name (everywhere else)
#
#### Examples for TCP/IP
# It is possible to declare "wild-card" style TRANCODEs.  This is
# accomplished by ending the TRANCODE with a '*'.  An example use is:
#
#     ABCD=TCP myhost1 2008 N
#     ABC*=TCP myhost2 2008 Y
#     A*=TCP myhost3 2008
#     *=TCP myhost4 2008
#
# This example would:
#   - Route an exact match of ABCD to myhost1 using non persistent socket
#     connections
#   - Route all TRANCODEs starting with ABC (but not ABCD) to myhost2 using
#     persistent socket connections
#   - Route all TRANCODEs starting with A (but not ABC) to myhost3 using
#     persistent socket connections
#   - Route all other TRANCODEs to myhost4 using persistent socket connections
#
##########################################
#*=TCP localhost 2008


#### Examples for MQSeries
# It is possible to declare "wild-card" style TRANCODEs.  This is
# accomplished by ending the TRANCODE with a '*'.  An example use is:
#
#     ABCD=MQS QManager QName ReplyQName
#     ABC*=MQS QManager QName
#     A*=MQS QManager
#     *=MQS
#
# If any items are missing (QManager, Qname, ReplyQName) the default
# values defined within the model at generation time will be used.
#
############### MQSeries MQEnvironment #################
#
# The commcfg.txt file can be used to modify the MQSeries MQEnvironment
# class properties.  Optionally these settings may be accomplished
# programmatically via the MQSDynamicCoopFlowExit class.
#
# See the MQSeries documentation for the proper values to be used for the
# following settings.
#
#   Note:    The MQChannel and MQHostname properties are required if the target
#            Queue manager is located on a remote system. Failure to specify
#            these properties will result in a connection attempt to a locally
#            defined Queue manager. This local connection may fail if the
```

```
#            MQSeries product is not properly configured with a local Queue
#            manager defined.
#            One symptom of this failure is MQSeries runtime dll load failures
#            if the MQSeries server software is not installed. If all Queue
#            managers are to located on remote machines, only MQSeries Client
#            software is required.
#            MQPort needs to be set to the TCP port number as defined within the
#            target queue manager configuration. A default value of 1414 will be
#            used if not overridden here.
#
########################################
#MQChannel=DEFAULT.MQI.CHANNEL
#MQHostname=hostname
#MQPort=1414
#
# Secure Socket Layer values for MQSeries client connections
# SSLKeyRepository value should not contain a .sto suffix.
# the value "c:\mq\sslstore" implies "c:\mq\sslstore.sto"
#
#MQSSLKeyRepository=c:\mqm\sslstore
#MQSSLCipherSpec=NULL_MD5
# PeerName is an SSL Distinguished Name pattern
#MQSSLPeerName = "CN=QMGR*, OU=Company, OU=Product";
#
########################################

############# CACHING ##################
#
# The commcfg.txt file may also be used to control how the Gen runtimes
# cache this file.  By default caching is not performed and the file is
# reread and reparsed for each flow out of the running application.  This allows
# file changes to be reflected immediately.
#
# That default behavior may not be ideal in all environments.  Therefore the
# special token 'CACHETIMEOUT' can be set to control the number of milliseconds
# to wait between rereading of the file.  All flows that occur between the read
# and the timeout use the cached version of the file in memory.  The first flow
# after the timeout has expired will force the file to be reread and the timeout
# to restart.  An example is:
#
#    CACHETIMEOUT=0      #(always timeout)
#    CACHETIMEOUT=180000 #(3 minute timeout)
#    CACHETIMEOUT=NEVER  #(cache will never be reread, default)
#
# The timeout is only useful within one process.  The NEVER setting will require
# the process to be stopped and the runtime to be unloaded before
# rereading the file.
#
########################################
```

```
#CACHETIMEOUT=NEVER

############### TRACING ##################
#
# The commcfg.txt file can be used to enable and disable tracing within
# the runtimes.  Use the special token 'CMIDEBUG' to enable and disable
# the tracing.  Examples of the use of this token are:
#
# To turn tracing off. Default.
#        CMIDEBUG=OFF
# To turn tracing on and write to the default file: trace.{pid}.out)
#        CMIDEBUG=ON
# To turn tracing on, and write to system: standard output, which
# might be the console, a command shell, or a service logfile depending
# on how the program was started.
#        CMIDEBUG=ON SYSTEM
# To turn tracing on and write to user specified file. Please make sure
# that the you have read/write permission for the same.
#        CMIDEBUG=ON FILENAME
#
##########################################
#CMIDEBUG=ON
```

Appendix E: z/OS Security

CICS regions and IMS systems use an external security manager such as Resource Access Control Facility (RACF) to provide the necessary security functions in those systems.

Different security features are used depending on application type and middleware.This appendix covers only very basic security information about LU6.2 and TCP/IP security as it applies to CA Gen DPS applications; for more comprehensive coverage of z/OS security as it applies to CICS or IMS See the appropriate IBM documentation.

LU6.2 Security

CICS and IMS handle the LU6.2 security differently but both TP Monitors use an external security manager (ESM) to implement security.

CICS

To provide the necessary LU6.2 security for CICS regions, CICS uses the z/OS System Authorization Facility (SAF) to route authorization requests to an external security manager (ESM) for validation of security data.

Transaction Security

CICS transaction security ensures that CICS calls an external security manager each time a transaction is entered at a terminal to verify that the terminal user is authorized to run the transaction.

APPC Session Security

Bind-time or session security can prevent an unauthorized connection between CICS and a remote system. If there is a possibility that a transmission line is switched or severed, bind-time security can prevent unauthorized session binds. A successful bind request requires that both the binding entities hold the same bind password (or SNA session key). Your SNA server for Win32 platforms provides Bind-time security.

APPC Link Security

Link security further restricts the resources that a user can access depending on the remote system from which they are accessed. The practical effect of link security is to prevent a remote user from attaching a transaction or accessing a resource for which the link user ID has no authority. Link security works by signing on each end of a session to RACF when the session is bound. Each half-session then has the access requirements of the link user ID, whose user profile is applied when a transaction is attached and whenever that transaction accesses a protected resource. Link security can be associated with a connection or a session, according to whether you want to control the link security for each group of sessions separately.

To define link security for a connection as a whole, specify the Securityname parameter in the connection definition. To define link security for individual groups or sessions within a connection, specify the user ID in the session definition as a user identifier.

The connection or session definition must be associated with a RACF user profile. This user profile must have access to any protected resources to which the inbound transaction needs access.

CICS log in the user ID specified in a similar way to that used for a CICS terminal user. If the logon fails, the logon failure message is sent to the master terminal destination, and the link is given the security of the default user in the receiving system; that is, the receiving system is able to access only those resources to which the default user has access.

If both user ID and security name are specified, user ID is the effective RACF user identifier for the set of sessions that are defined by that particular session definition. For other session definitions for the same connection that do not specify user ID, security name is used.

If there is no user ID for link security (neither user ID nor security name was specified), the user profile for the group or session defaults to the user ID defined for that of the default user. Changing the link authority requires the release and re-acquisition of the link.

APPC User Security

User security checks that a user is authorized to attach a given transaction. It causes a second check to be made against a user signed on to a terminal in addition to link security. Levels of user security are set using the ATtachsec parameter of the CICS connection definition.

The ATtachsec operand specifies the logon requirements for incoming transaction attach requests. When an APPC session is bound, each side tells the other the level of attach security user verification that is performed on its incoming requests. There is no negotiation on this.

ATtachsec = Local specifies that a user identifier or password is not required. The link security is considered sufficient for that system, and the transaction sender (client) need not provide further security information. Any user ID and password information, if it is received, is ignored. CICS makes the user security profile equivalent to the link security profile.

A non-local ATtachsec parameter requires a user identifier, or a user identifier and a password. The following are some non-local values:

- ATtachsec = Identify requires a user identifier, but no password.
- ATtachsec = Verify requires a user identifier and password.

IMS

To provide the necessary LU6.2 security for IMS systems, IMS uses the services of APPC z/OS to interface to an external security manager such as Resource Access Control Facility (RACF) or other equivalent external security manager (ESM). In general, the security precautions that are needed are as follows:

- Limiting the client LUs from which a conversation request can enter the system
- Ensuring that the client conversation request contains security information, user ID and password
- Limiting by user ID, those users who can request a particular transaction in the system

Client LU Security

User access to APPC z/OS LUs (application systems) can be controlled by limiting the client LUs that user requests can come from. The RACF profiles in the APPCPORT class control which user IDs may access a given system.

Example:

- RDEFINE APPCPORT IMSLU VACC(NONE)
- PERMIT IMSLU CLASS(APPCPORT) ID(IMSUSER) ACCESS(READ)

These let user ID (IMSUSER) to access system (IMSLU).

Ensuring Conversation Security

VTAM and RACF both allow definition of an APPC z/OS LU to specify that security information is required to be passed from the initiating client. The VTAM APPL statement has a SECACPT keyword of CONV that requires user ID and password from the client on conversation requests.

Example:

```
APPL ACBNAME = IMSLU . . . SECACPT = CONV
```

The CONVSEC field in the session segment of the RACF APPCLU profile can also be set to CONV that performs the same check as the VTAM SECACPT parameter.

Limiting Transaction Access to User IDs

The RACF APPL class controls which user IDs allocate conversations to a particular system.

Example:

- RDEFINE APPL IMSLU VACC(NONE)
- PERMIT IMSLU CLASS(APPL) ID(IMSUSER) ACCESS(READ) allows user ID (IMSUSER) to access system (IMSLU). IMSUSER must also have a RACF profile that limits which transactions this user can access.

LU6.2 Security Features Used by CA Gen

CA Gen's LU6.2 implementation supports Transaction Attach Security for both CICS and IMS by including a client-supplied user ID and password in the SNA FMH-5, the Attach Format Management Header for LU6.2.

Depending on the values set in the client's security exit and, if used, the Client Manager configuration CA Gen Enhanced or Standard security data is used to populate the FMH-5 user ID and password fields. As long as a security option other than NONE is chosen a user ID and password is sent with every message.

TCP/IP Security

TCP/IP security is handled differently by CICS and IMS.

CICS

CA Gen TCP/IP servers execute as CICS non-terminal transactions so they are subject to two types of security checks: Surrogate user checking and Transaction-attach security. The user ID and password values that are passed to CICS must be checked using an external security manager. Both Surrogate and Transaction-attach security use only the user ID for security checking.

User ID and password values are entered in uppercase if RACF is the ESM used because RACF only recognizes uppercase characters. Other external security managers also require uppercase, or they are case-sensitive.

Surrogate User Checking

CICS performs surrogate user security checking using the surrogate user facility of an external security manager (ESM) such as RACF. A surrogate user is one who has the authority to start work on behalf of another user and is authorized to act for that user without knowing that other user's password. Surrogate user checking of the following CICS resources is important for TCP/IP CA Gen servers:

- The CICS default user
- PLT post-initialization processing
- Started the transactions

CICS Default User

CICS performs a surrogate user security check against its own user ID (the CICS region user ID) to ensure that it is properly authorized as a surrogate of the default user ID specified on the DFLTUSER system initialization parameter.

PLT Post-Initialization Processing

When a program list table is specified on a PLTPI system initialization parameter, CICS checks that the region user ID is authorized as a surrogate user of the user ID specified in the PLTPIUSR system initialization parameter. The PLTPIUSR parameter specifies the user ID that CICS is to use for PLT programs that run during the CICS initialization. All PLT programs that are run under the authority of the specified user ID, which must be authorized to all the resources referenced by the programs.

The PLTPISEC parameter defines the scope of PLT security checking. This specifies whether command security checks and resource security checks are to apply to PLTPI programs.

If you do not specify the PLTPIUSR parameter, CICS runs PLTPI programs under the authority of the CICS region user ID, the DFLTUSER, in which case CICS does not perform a surrogate user check. However, the CICS region user ID must then be authorized to all the resources referenced by the PLT programs. Furthermore, the CICS region user ID is associated with any transactions started by PLT programs, and therefore must be authorized to run such transactions.

Started Transactions

CICS performs surrogate user checks when the EXEC CICS START command is used to start a transaction that is not associated with a terminal. The user ID under which the transaction issuing the START command runs is named the starting-user ID, and the user ID under which the started transaction runs is named the started-user ID.

Notes:

- If the USERID option is specified on the START command, the started-user ID is set to that specified user ID.
- If the TERMID option is specified on the START command, surrogate user checking does not apply. The started-user ID is inherited from the terminal at which the transaction runs.
- If neither TERMID nor USERID is specified on the START command, the started-user ID is set to be the same as the starting-user ID.

CICS requires that all user IDs associated with the transaction issuing the START are surrogates of the started-user ID. CICS also assumes that any user ID is always a surrogate of itself. Therefore, user IDs that are the same as started-user ID are regarded as surrogates already, and the external security manager is not named for them.

The special conditions apply when CICS intercommunication is used in that the transaction can be associated with user IDs that are different from the starting-user ID. If an EXEC CICS START command (without TERMID) is function that is shipped or is executed from a transaction-routed transaction, the command can be subject to link security (link security is discussed under LU6.2 CICS Security because the links are mostly implemented as LU6.2 resources). If link security is in effect, CICS also performs a surrogate user check to verify that the user ID for link security is authorized as a surrogate user to the user ID for the started transaction. The surrogate check is done at this stage even if the USERID is omitted (if the started-user ID is different from the link user ID).

Transaction Attach Security

There are no CICS parameters that let you control transaction-attach security at the individual transaction level, because CICS Transaction Server for z/OS Release 3 transaction-attach security for non-terminal transactions is always done if transaction security is active (SEC=YES and XTRAN is not NO). In this case CICS always checks the authority of any user ID to attach a transaction.

When the transactions are started by the EXEC CICS START TRANSID command with no TERMID but with a USERID operand, they are started using the specified user ID that must be authorized to attach the started transaction. However, when transactions are started by the EXEC CICS START TRANSID command with neither a TERMID nor a USERID operand they inherit the user ID associated with the starting transaction, and the inherited user ID must be authorized to attach the started transaction. Users who were able to run such transactions successfully in prior releases of CICS now authorize to attach these transactions. An option is to customize RACF (exit ICHRFX01) so that the transaction-attach check is always successful for non-terminal transactions. See the CICS RACF security guide for more information about this option.

IMS

CA Gen's TCP/IP implementation uses the Open Transaction Manager Access (OTMA) feature of the IMS Transaction Manager (IMS TM) and IMS Connect.

OTMA

The OTMA feature is installed with IMS TM as a part of the product, so it does not require any specific action to get installed. Also, any IMS system generation specifications for the OTMA implementation are not necessary. OTMA is a transaction-based connectionless client-server protocol that provides an access path and an interface specification for sending and receiving transactions and data from IMS.

OTMA is implemented for IMS in a sysplex-capable environment and as such it is restricted to the domain of MVS Cross System Coupling Facility (XCF). OTMA provides the facility for IMS to communicate efficiently with MVS applications other than VTAM. These MVS applications are named OTMA clients. These OTMA clients can be an IBM written application, independent software vendor's application, or user-written code. IMS Connect is a commonly used the OTMA client.

The OTMA client handles all device dependencies for a particular network protocol, which can be TCP/IP or System Network Architecture (SNA) and IMS TM can operate without needing any device-characteristic information. The OTMA client communicates with IMS by sending and receiving messages. It adds the OTMA message prefix to the input messages and it uses the message prefix to route the output data to the originating device or program. The application data section follows the message prefix, which can be the data sent to the IMS application, or the response sent to the client.

Messages that are destined for IMS/OTMA fall into the following categories:

- client-bid messages—sent by the OTMA clients at connect time
- end-user messages—sent by client applications

When security checking is turned on in IMS/OTMA all messages received using OTMA clients must include security-related information in the OTMA prefix.

OTMA Security Levels

There are four OTMA security levels: NONE, CHECK, FULL, and PROFILE. The OTMA security level for an IMS system may be established by an IMS startup parameter specification, OTMASE=, in the DFSPBxxx member of the IMS procedure library or by the /SECURE OTMA command. By default, the OTMA security level in IMS is set to FULL (or F). The meanings of the parameters are as follows:

- CHECK—Causes existing RACF calls to be made. The IMS commands are checked using the RACF resource class of CIMS. The IMS transactions are checked using TIMS.
- FULL—Causes the same processing as the CHECK parameter, but uses more RACF calls to create the security environment for dependent regions.
- NONE—Does not call RACF within IMS for security verification.
- PROFILE—Causes the values in the security data section of the OTMA message prefix for each transaction to be used.

If RACF is used for security, the XCF group name and the member name (IMSXCF.group.member) are defined in the FACILITY class. If the IMSXCF.group.member is not defined in the FACILITY class, a client bid is not allowed.

The /SECURE OTMA PROFILE command can be used to enable security validation at individual transactions level.

IMS Connect

IMS Connect is a TCP/IP server that enables TCP/IP clients to exchange messages with IMS OTMA. IMS Connect provides communication linkages between TCP/IP clients and IMS (datastores). It supports multiple TCP/IP clients accessing multiple datastore resources. The IMS Connect runs on an MVS or z/OS platform. IMS Connect performs router functions between TCP/IP clients and datastores. Input (request) messages that are received from TCP/IP clients, using TCP/IP connections, are passed to an IMS datastore through XCF sessions. IMS Connect receives response messages from the datastore and then passes them back to the originating TCP/IP clients.

IMS Connect supports TCP/IP clients communicating with socket calls, but it can support any TCP/IP client that communicates with a different input data stream format. Supplied or User-written message exits can execute in the IMS Connect address space to convert customer message format to OTMA message format before IMS Connect sends the message to IMS. The user-written message exits also convert the OTMA message format to customer message format before sending a message back to IMS Connect. IMS Connect then sends output to the client.

In addition to TCP/IP client communications, IMS Connect also supports local communications through the "Local" option.

IMS Connect User Message Exit Routine

Each message that is received by IMS Connect is processed by a user message exit routine to translate/format the message so that it is in a format acceptable to IMS/OTMA. The user message exit that is invoked to process a message depends on the source/origination of the message.

IMS Connect supports the capability to invoke a security exit routine from the message exit routines that are used to translate messages. This security exit routine can be a user-provided security exit or the IMSLSECX sample security exit that is provided by TCP/IP is customized to meet installation security requirements.

Client-bid Security

IMS Connect must be assigned a RACF user ID and connected to a RACF group. The user ID and group for IMS Connect is supplied in the client-bid message when IMS Connect attempts to connect to IMS/OTMA. If IMS/OTMA security is turned on, the IMS Connect user ID (and optionally, the group name) is required for a successful client-bid.

If the datastore (which is IMS) is RACF protected, you can start IMS Connect as a job with the JOB card specifying a valid USERID to make the connection from IMS Connect to IMS. Alternatively, you can use the RACF STARTED class or the Started Procedure Table to associate the IMS Connect user ID to the started procedure used to initialize IMS Connect.

End-User Validation

IMS Connect allows security for individual client messages by checking a RACF flag. If you turn the RACF flag ON, IMS Connect uses the System Authorization Facility (SAF) interface to invoke RACF by issuing a RACROUTE REQUEST=VERIFY call to verify a user ID and password. This is a global option in that IMS Connect either verifies for all users or does not verify for any users. There are two ways to activate the RACF flag, as follows:

- Set the RACF flag in the configuration file, HWSCFG00, by setting the flag to Y (RACF=Y).
- Issue the HWS command SETRACF to set the RACF flag.

The security information pertaining to end users must be provided for IMS Connect to invoke RACF to verify the user IDs associated with end users' messages. Security information is passed from TCP/IP clients in the IMS Request Message (IRM) or can be provided by the user message exit portion of the IRM that has the security-related information.

IMS System

From an IMS perspective after a message has been received by OTMA, the security checking is the same as for any other request. When OTMA security is activated, upon receipt of a message IMS invokes RACF to perform one of the following actions:

- If the incoming message contains a valid UTOKEN, build an ACEE from the information in the UTOKEN and return the ACEE to IMS.
- If the incoming message does not contain a UTOKEN, validate that the user ID in the incoming message has been defined to RACF and build (and return to IMS) an ACEE for valid user IDs.

IMS/OTMA determines which, if any, security facilities is invoked for messages received using OTMA based on the OTMA security level and on whether security exits have been included in the system.

TCP/IP Security Features Used by CA Gen

CA Gen TCP/IP implementation supports Transaction-attach security for both CICS and IMS by including a client-supplied user ID and password in CA Gen Common Format Buffer. All requests that target CICS, IMS, or both are always in a CFB format.

Depending on the options that are selected in the client's security exits and, if used, the Client Manager configuration CA Gen Enhanced or Standard security data is used to populate the CFB user ID and password fields. As long as a security option other than NONE is chosen a user ID and password is sent with every message.

CICS

CA Gen offers only one TCP/IP Direct Connect for CICS product, the CICS Sockets Server implementation. The Direct Connect for the CICS implementation that is offered in earlier releases, TILSTNR and TICONMGR, has been stabilized and is not included with CA Gen.

In the CICS Sockets Servers implementation, the CA Gen Listener program TISRVLIS calls user exit TIRSLEXT. TIRSLEXT receives the ASCII and ECBDIC versions of trancode and code page, ASCII User ID and one of the following:

- The EBCDIC password when standard security is used
- A flag indicating that enhanced security is used (the password is only available in the server TIRSECV exit)

In TIRSLEXT the ASCII user Id is translated to EBCDIC and can be modified, including blanked out, if necessary. If the ESM requires the security data to be in uppercase the conversion of the security data must be done in this exit.

The TIRSLEXT exit can be modified to invoke the ESM so that the security data can be verified and the request that is accepted or rejected. When accepted the server is started using an EXEC CICS START TRANSID with the USERID operand, with the exception that if the user Id is blanked out the server is started without specifying the USERID operand.

If the START is successful but the Attach fails, or the server is unable to execute, the TAKESOCKET Socket API fails and closes the server socket connection.

IMS

For IMS the CA Gen IMS Connect message user exit CAGRITCP calls user exit CAGRITSC to obtain the SECOTMA flag that match the IMS/OTMA OTMASE parameter. This SECOTMA flag controls how the OTMA security prefix is populated as follows:

- If the OTMA security flag is N, we do not populate any of the fields in the OTMA security header prefix.
- If the OTMA Security Flag is F or C, we populate the user ID part of the OTMA security header prefix with the CFB security data from sent in the client request.

- The password is placed in the OTMA User Data Header.
- If a Group ID or a Token is supplied (using the CAGRITSC exit), we populate the relevant fields in the OTMA Security Header.

Since the CAGRITSC exit also gets visibility of the user ID and password that is sent by the client, a call to the ESM can be done in it and other security checks bypassed. Also, this exit is an ideal place to do lowercase to uppercase conversion of the security data that is required.

Index

E

F

G

I

J

M

P

R

S

T

U

W

X