Oracle9*i*AS InterConnect Adapter for CICS

Installation and User's Guide

Release 2 (9.0.2)

February 2002

Part No. A95442-01

Oracle9*i*AS InterConnect Adapter for CICS Installation and User’s Guide, Release 2 (9.0.2)

Part No. A95442-01

Contents

Send Us Your Comments

Oracle9*i*AS InterConnect Adapter for CICS Installation and User's Guide, Release 2 (9.0.2)

Part No. A95442-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: iasdocs_us@oracle.com
- FAX: 650-506-7407 Attn: Oracle9*i* Application Server Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle9*i* Application Server Documentation
  500 Oracle Parkway, M/S 2op3
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This preface contains these topics:

- Intended Audience
- Documentation Accessibility
- Organization
- Related Documentation
- Conventions

Intended Audience

This guide is intended for those who perform the following tasks:

- install applications
- maintain applications

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This document contains:

Chapter 1, "Introduction"

This chapter describes the CICS adapter and the hardware and software requirements.

Chapter 2, "Installation and Configuration"

This chapter describes installation and configuration of the CICS adapter.

Chapter 3, "CICS and the CICS Adapter"

This chapter describes the concepts for the CICS adapter.

Chapter 4, "Systems Network Architecture Definitions"

This chapter provides system network architecture definitions for the CICS adapter.

Chapter 5, "Systems Network Architecture Concepts"

This chapter provides concepts for the system network architecture for the CICS adapter.

Chapter 6, "Message Description Language Reference"

This chapter provides a reference to the message description language.

Chapter 7, "Using the Configuration Editor"

This chapter provides information on using the Configuration Editor to configure the CICS adapter.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9iAS InterConnect User Guide* in the Oracle9*i* Application Server Documentation Library
- *Oracle9i Application Server Installation Guide*
- *Oracle9iAS InterConnect Adapter Configuration Editor User's Guide*

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/admin/account/membership.html`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/docs/index.htm`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples
- Conventions for Microsoft Windows Operating Systems

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.` Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`

|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}`
		`[COMPRESS | NOCOMPRESS]`

...	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS `*`subquery`*`;`
	■ That you can repeat a portion of the code	`SELECT `*`col1`*`, `*`col2`*`, ... , `*`coln`*` FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);`
		`acct    CONSTANT NUMBER(4) := 3;`

Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`*
		`DB_NAME = `*`database_name`*

Convention	Meaning	Example

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Conventions for Microsoft Windows Operating Systems

The following table describes conventions for Microsoft Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Oracle Database Configuration Assistant, choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32` is the same as `C:\WINNT\SYSTEM32`

Convention	Meaning	Example
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`

	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (’) do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job=’SALESMAN’ and sal<1600\"`
		`C:\>imp SYSTEM/`*`password`*` FROMUSER=scott TABLES=(emp, dept)`

`HOME_NAME`	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`

Convention	Meaning	Example
ORACLE_HOME and *ORACLE_BASE*	In releases prior to Oracle8*i* release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory that by default used one of the following names: ■ `C:\orant` for Windows NT ■ `C:\orawin95` for Windows 95 ■ `C:\orawin98` for Windows 98 This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install Oracle9*i* release 1 (9.0.1) on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora90`. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions.	Go to the *ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\admin` directory.

1 Introduction

Oracle connects to CICS through the CICS adapter. This book introduces the CICS environment and any CICS specific information. This chapter discusses the following topic:

- What is CICS?

What is CICS?

The IBM Customer Information Control System (CICS) allows data exchanges by sending and receiving buffers to and from an application using the CICS adapter. The CICS adapter uses the CPI-C LU6.2 SNA (LU6.2) protocol API and/or the ECI protocol API. The LU 6.2 protocol communicates with a CICS transaction, while the ECI protocol communicates with a CICS program. A CICS transaction contains presentation logic and business logic, while the CICS program contains only the business logic.

System Requirements and Platforms

To use CICS adapter with Oracle9*i*AS InterConnect, the following requirements must be met.

For the ECI protocol:

- IBM CICS Universal client

For using CPI-C LU6.2 SNA (LU6.2):

- IBM SNA client or the MS SNA client

The CICS adapter runs on:

- Windows NT
- Windows 2000

Definitions

The following terms are specific to the CICS adapter:

- Logical Unit (LU)
- CPI-C

Logical Unit (LU)

A logical unit represents the logical destination of a communication data flow. The formal definition of a logical unit is the means by which an end user gains entry into a network. An end user is defined as the ultimate source, or destination, of data flow in a network. SNA supports several different types of logical units. These are grouped together in numbered logical unit types, such as logical unit type 2 for 3270 display terminals, and logical unit type 4 for printers. The logical unit type for CICS-to-CICS communication is logical unit type 6.2, and is frequently referred to as advanced program-to-program communication (APPC). Each logical unit is given a unique name that identifies it in the network. There are two types of logical units 6.2 pertinent to CICS adapter:

- Dependent logical unit 6.2 can have only a single session and therefore only one conversation at a time.
- Independent logical unit 6.2 can have more one session with other logical units—any conversations can be held simultaneously between two logical units.

CPI-C

CPI (CPI Communications) provides a cross-system-consistent and easy-to-use programming interface for applications that require program-to-program communication. From an application's perspective, CPI-C provides the function necessary to enable this communication. The conversational model is implemented in two major communications protocols: Advanced Program-to-Program Communication (APPC) and Open Systems Interconnection Distributed Transaction Processing (OSI TP). The APPC protocol is also referred to as Logical Unit type 6.2 (logical unit 6.2). CPI-C provides access to both APPC and OSI-TP.

2

Installation and Configuration

This chapter describes installation and configuration of the CICS adapter. This chapter discusses the following topics:

- Installing the CICS Adapter
- CICS Adapter Configuration

Installing the CICS Adapter

This section contains these topics:

- Preinstallation Tasks
- Installation Tasks

Preinstallation Tasks

The CICS adapter must be installed in one of the following Oracle homes:

- An existing Oracle9*i* Application Server Oracle home
- An existing Oracle9*i* Application Server Infrastructure Database Oracle home
- An existing Oracle9*i*AS InterConnect Oracle home
- A new Oracle home (the installer creates this for you)

Consult the *Oracle9i Application Server Installation Guide* before proceeding with the CICS adapter installation. This guide includes information on:

- CD-ROM mounting
- Oracle Universal Installer startup
- Oracle9*i*AS InterConnect installation
- Oracle9*i*AS InterConnect software, hardware, and system requirements

Note: Oracle9*i*AS InterConnect Hub is installable through the Oracle9*i*AS InterConnect Hub installation type. You must install the Oracle9*i*AS InterConnect Hub before proceeding with the CICS adapter installation.

Installation Tasks

To install the CICS adapter:

1. Click **Next** on the Welcome page.

   The File Locations page displays.

2. Enter the following information in the Destination fields:

   - Name—The Oracle home name.

- Path—The full path to the Oracle home in which to install the CICS adapter.

Note: Do not change the path specified in the Source field. This is the location on the CD-ROM from which to install the CICS adapter.

3. Click **Next**.

   The Installation Types page displays.

4. Select Oracle9*i*AS InterConnect Adapters and click **Next**.

   The Available Product Components page displays.

5. Select Oracle9*i*AS InterConnect CICS Adapter and click **Next**.

6. If the CICS adapter is not being installed on the same computer as Oracle9*i*AS InterConnect Hub and another adapter is not installed in the current Oracle home, the Oracle9*i*AS InterConnect Hub Database screen appears. Enter the following information about the Oracle9*i*AS InterConnect Hub to use:

   - Host Name—The hostname of the computer on which Oracle9*i*AS InterConnect Hub is installed.
   - Port Number—The port number of the computer.
   - Database SID—The system identifier (SID) of the Oracle9*i*AS InterConnect Oracle9*i*AS Metadata Repository.
   - Password—The password for the Oracle9*i*AS Metadata Repository schema.

   The Oracle9*i*AS Metadata Repository stores metadata used by Oracle9*i*AS InterConnect to coordinate communication between components.

7. Click **Next**.

   The Oracle9*i*AS InterConnect CICS Adapter page displays.

8. Enter the name of the application associated with the CICS adapter. White spaces or blank spaces are not permitted. The default value is `myCICSApp`.

9. Click **Next**.

   The Oracle9*i*AS InterConnect CICS Adapter - Specify CICS client binaries location page displays.

10. Enter the location for the client binaries location.

11. Click **Next**. Complete the fields for any other components selected for installation, such as other adapters. When finished, the Summary page displays.
12. Click **Install** to install the CICS adapter and other selected components. The CICS adapter is installed in the following directory:

Platform	Directory
Windows	`%ORACLE_HOME%\oai\9.0.2\adapters\`*`Application`*
UNIX	`$ORACLE_HOME/oai/9.0.2/adapters/`*`Application`*

`Application` is the value you specified in Step 8 on page 2-3.

CICS Adapter Configuration

Table 2–2, Table 2–3, and Table 2–4 describe executable files, configuration files, and directories. These files and directories are accessible from the directory shown in Table 2–1:

Table 2–1 CICS Adapter Directory

On..	Go to...
UNIX	`$ORACLE_HOME/oai/9.0.2/adapters/`*`Application`*
Windows	`%ORACLE_HOME%\oai\9.0.2\adapters\`*`Application`*

Table 2–2 Executable Files

File	Description
`start.bat` (Windows)	Takes no parameters, starts the adapter.
`start` (UNIX)	
`stop.bat` (Windows)	Takes no parameters; stops the adapter.
`stop` (UNIX)	
`ignoreErrors.bat` (Windows)	If an argument is specified, then the given error code will be ignored. If no argument is specified, than all error codes specified in the `ErrorCodes.ini` will be ignored.
`ignoreErrors` (UNIX)	

Table 2–3 Configuration Files

File	Description
`ErrorCodes.ini` (Windows and UNIX)	Should contain one error code per line.
`adapter.ini` (Windows and UNIX)	Consists of all the initialization parameters which the adapter reads at startup. Refer to Appendix A for a typical `adapter.ini` file.

Table 2–4 Directories

File	Description
persistence	The messages are persisted in this directory. This directory or its contents should not be edited.
logs	The logging of adapter activity is done in subdirectories of the log directory. Each new run of the adapter creates a new subdirectory in which logging is done in an `oailog.txt` file.

Using the Application Parameter

Adapters do not have integration logic. The CICS adapter has a generic transformation engine that processes metadata from the repository as runtime instructions to do transformations. The application defines for an adapter what its capabilities are. For example, it can define what messages it can publish, what messages it can subscribe to, and what are the transformations to perform. The application parameter allows the adapter to become smart in the context of the application to which it is connected. It allows the adapter to retrieve from the repository only that metadata that is relevant to the application. The application parameter must match the corresponding application that will be defined in *i*Studio under the Applications folder.

If you are using pre-packaged metadata, after importing the pre-packaged metadata into the repository, start up *i*Studio to find the corresponding application (under the Applications folder in *i*Studio) to use as the application for the adapter you are installing (unless the package you are using provides directions for what the application should be).

adapter.ini Initialization Parameter File

This section contains these topics:

- Hub.ini
- Agent Connection Parameters
- CICS Adapter Parameters

Hub.ini

The CICS adapter connects to the hub database using parameters from the `hub.ini` file located in the hub directory. The following table lists the parameter name, a description for each parameter, the possible and default values, and an example.

Parameter	Description	Example
`hub_username`	The name of the hub database schema (or username). Possible values are valid hub database username. There is no default value.	`hub_username=myhub`
`hub_password`	The password for the hub database user. Possible values are the valid password for the hub database user. There is no default value.	`hub_password=manager`
`hub_host`	The name of the machine hosting the hub database. Possible values are the valid machine name. There is no default value.	`hub_host=mpjoshipc`
`hub_instance`	The valid SID of the hub database. There is no default value.	`hub_instance=orcl`
`hub_port`	The TNS listener port number for the HUB database instance. There is no default value.	`hub_port=1521`
`repository_name`	The valid name of the repository this adapter talks to. There is no default value.	`repository_name=myrepo`

Agent Connection Parameters

The CICS adapter connects to the spoke application using parameters from the `adapter.ini` file. The following table lists the parameter name, a description for each parameter, the possible and default values and an example.

Parameter	Description	Example
`application`	The name of the application this adapter connects to. This must match with the name specified in iStudio during creating of metadata. Any alphanumeric string can be used. There is no default value.	`application=aqapp`
`partition`	The partition this adapter handles as specified in iStudio. Any alphanumeric string is a possible value. There is no default value.	`partition=germany`
`instance_number`	To have multiple adapter instances for the given application with the given partition, each adapter should have a unique instance number. Possible values are any integer greater than 1. There is no default value.	`instance_number=1`
`agent_log_level`	Specifies the amount of logging necessary. Possible values are: 0=errors only 1=status and errors 2=trace, status, and errors The default value is 1.	`agent_log_level=2`
`agent_subscriber_name`	The subscriber name used when this adapter registers its subscription. The possible value is a valid Oracle Advanced Queuing subscriber name and there is no default value.	`agent_subscriber_name=aqapp`
`agent_message_selector`	Specifies conditions for message selection when registering its subscription with the hub. The possible value is a valid Oracle Advanced Queuing message selector string. There is no default value.	`agent_message_selector=recipient_list like '%aqapp,%'`
`agent_reply_subscriber_name`	The subscriber name used when multiple adapter instances for the given application with the given partition are used. Optional if there is only one instance running. The possible value is application name (`parameter: application`) concatenated with instance number (`parameter: instance_number`). There is no default value.	If `application=aqapp`, `instance_number=2`, then, `agent_reply_subscriber_name=aqapp2`

Parameter	Description	Example
`agent_reply_message_selector`	Used only if multiple adapter instances for the given application with the given partition. The possible value is a string built using concatenating application name (`parameter:application`) with instance number (`parameter:instance_number`). There is no default value.	If `application=aqapp`, `instance_number=2`, then `agent_reply_message_selector=receipient_list like '%,aqapp2,%'`
`agent_tracking_enabled`	Specifies if message tracking is enabled. Set to false to turn off all tracking of messages. Set to true to track messages with tracking fields set in iStudio. Possible values are `true` or `false`. The default value is `true`.	`agent_tracking_enabled=true`
`agent_throughput_measurement_enabled`	Specifies if throughput measurement is enabled. Set to true to turn on all throughput measurements. Possible values are `true` or `false`. The default value is `true`.	`agent_throughput_measurement_enabled=true`
`agent_use_custom_hub_dtd`	Specifies if a custom DTD should be used for the common view message when handing it to the hub. By default adapters use an Oracle9*i*AS InterConnect-specific DTD for all messages sent to the hub as other Oracle9*i*AS InterConnect adapters will be retrieving the messages from the hub and know how to interpret them. Set to true if for every message, the DTD imported for the message of the common view is to be used instead of the Oracle9*i*AS InterConnect DTD. Only set to true if a Oracle9*i*AS InterConnect adapter is not receiving the messages from the hub. Possible values are `true` or `false`. There is no default value.	`agent_use_custom_hub_dtd=false`
`agent_metadata_caching`	Specifies the metadata caching algorithm. Possible values are: ▪ `startup`—Cache everything at startup. This may take a while if there are a lot of tables in the repository. ▪ `demand`—Cache metadata as it is used. ▪ `none`—No caching. This slows down performance. The default value is `demand`.	`agent_metadata_caching=demand`

Parameter	Description	Example
`agent_dvm_table_caching`	Specifies the DVM caching algorithm. Possible values are: ■ `startup`—Cache all DVM tables at startup. This may take a while if there are a lot of tables in the repository. ■ `demand`—Cache tables as they are used. ■ `none`—No caching. This slows down performance. The default value is `demand`.	`agent_dvm_table_caching=demand`
`agent_lookup_table_caching`	Specifies the lookup table caching algorithm. Possible values are: ■ `startup`—Cache all lookup tables at startup. This may take a while if there are a lot of tables in the repository. ■ `demand`—Cache tables as they are used. ■ `none`—No caching. This slows down performance. The default value is `demand`.	`agent_lookup_table_caching=demand`
`agent_delete_file_cache_at_startup`	With any of the agent caching methods enabled, metadata from the repository is cached locally on the file system. Set this parameter to `true` to delete all cached metadata on startup. Note: After changing metadata or DVM tables for this adapter in iStudio, you must delete the cache to guarantee access to the new metadata or table information. Possible values are `true` or `false`. The default value is `false`.	`agent_delete_file_cache_at_startup=false`
`agent_max_ao_cache_size`	Specifies the maximum number of application objects' metadata to cache. Possible values are any integer greater than 1. The default value is `200`.	`agent_max_ao_cache_size=200`
`agent_max_co_cache_size`	Specifies the maximum number of common objects' metadata to cache. Possible values are any integer greater than 1. The default value is `100`.	`agent_max_co_cache_size=100`
`agent_max_message_metadata_cache_size`	Specifies the maximum number of messages' metadata to cache (publish/subscribe and invoke/implement). Possible values are any integer greater than 1. The default value is `200`.	`agent_max_message_metadata_cache_size=200`

Parameter	Description	Example
agent_max_dvm_ table_cache_size	Specifies the maximum number of DVM tables to cache. Possible values are any integer greater than 1. The default value is 200.	agent_max_dvm_table_ cache_size=200
agent_max_ lookup_table_ cache_size	Specifies the maximum number of lookup tables to cache. Possible values are any integer greater than 1. The default value is 200.	agent_max_lookup_ table_cache_size=200
agent_max_queue_ size	Specifies the maximum size that internal Oracle9*i*AS InterConnect message queues can grow. Possible values are any integer greater than 1. The default value is 1000.	agent_max_queue_ size=1000
agent_ persistence_ queue_size	Specifies the maximum size that internal Oracle9*i*AS InterConnect persistence queues can grow. Possible values are any integer greater than 1. The default value is 1000.	agent_persistence_ queue_size=1000
agent_ persistence_ cleanup_interval	Specifies how often the persistence cleaner thread should run. Possible values are any integer greater than 30000. The default value is 60000.	agent_persistence_ cleanup_interval=60000
agent_ persistence_ retry_interval	Specifies how often the persistence thread should retry when it fails to push a Oracle9*i*AS InterConnect message. Possible values are any integer greater than 5000. The default value is 60000.	agent_persistence_ retry_interval=60000
service_path	Windows only. The value that the environment variable PATH should be set to. path is set to the specified value before forking the Java VM. Typically, all directories containing all necessary DLLs should be listed here. Possible values are the valid path environment variable setting. There is no default value.	service_ path=%JREHOME%\bin;D:\ oracle\ora902\bin
service_ classpath	The classpath used by the adapter Java VM. If a custom adapter is developed and as a result, the adapter is to be used to pick up any additional jars, add the jars to the existing set of jars being picked up. Possible values are the valid classpath. There is no default value.	service_ classpath=D:\oracle\ ora902\oai\902\lib\ oai.jar;%JREHOME%\lib\ i18n.jar;D:\oracle\ora 902\jdbc\classes12.zip
service_class	The entry class for the Windows NT service. The possible value is oracle/oai/agent/service/AgentService. There is no default value.	service_ class=oracle/oai/agent /service/AgentService
service_max_ java_stack_size	Windows only. The maximum size to which the Java VM's stack can grow. Possible values are the valid Java VM maximum native stack size. The default value is the default for the Java VM.	service_max_java_ stack_size=409600

Parameter	Description	Example
service_max_ native_stack_ size	Windows only. The maximum size to which the Java VM's native stack can grow. Possible values are the valid Java VM maximum native stack size. The default value is the default for the Java VM.	service_max_native_ size=131072
service_min_ heap_size	Windows only. Specifies the minimum heap size for the adapter Java VM. Possible values are the valid Java VM heap sizes. The default value is the default Java VM heap size.	service_min_heap_ size=536870912
service_max_ heap_size	Windows only. Specifies the maximum heap size for the adapter Java VM. Possible values are any valid Java VM heap sizes. The default value is 536870912.	service_max_heap_ size=536870912
service_num_vm_ args	Windows only. The number of service_vm_arg<number> parameters specified. Possible values are the number of service_vm_arg<number> parameters. There is no default value.	service_num_vm_args=1
service_vm_ arg<number>	Windows only. Specifies any additional arguments to the Java VM. For example, to get line numbers in any of the stack traces, set service_vm_arg1=java.compiler=NONE. If there is a list of arguments to specify, use multiple parameters as shown in the example by incrementing the last digit starting with 1. Be sure to set the service_ num_vm_args correctly. Possible values are any valid Java VM arguments. There is no default value.	service_vm_ arg1=java.compiler= NONE service_vm_ arg2=oai.adapter=.aq
service_jdk_ version	Windows only. The JDK version the adapter Java VM should use. The default value is 1.3.1.	service_jdk_ version=1.3.1
service_jdk_dll	Windows only. The dll the adapter Java VM should use. The default value is jvm.dll.	service_jdk_ dll=jvm.dll

CICS Adapter Parameters

The following table lists the parameters specific to the CICS adapter.

Parameter	Description	Example
bridge_class	This indicates the entry class for the CICS adapter. Do not modify this value. A possible value is com.actional.oai.Agent. There is no default value.	bridge_ class=com.actional.oai. Agent

3 CICS and the CICS Adapter

This chapter discusses the following:

The CICS Adapter

In CICS, both partners (CICS and the CICS adapter) must define the content of the CICS buffer. When this is done, it is possible to exchange data.

The CICS adapter provides the Oracle9*i*AS InterConnect system with the ability to interact with CICS. To make the CICS buffer description visible to Oracle9*i*AS InterConnect, the adapter specifies how to format data (representing a call) on a communication line by describing the format of the data being passed. The format is described in an Message Description Language (MDL) file. It abstracts a method call as input and output messages. A normal method call has arguments passed from the Agent (caller) to CICS (callee) represented as synchronous or asynchronous messages going up and back from arbitrary services.

Individual message reply and request pairs (each request message can have a reply message) describe interactions between clients and servers.

The CICS adapter represents a method call as a pair of messages:

- A request message containing all input arguments.
- A reply message containing all output arguments.

Message Description Language (MDL)

Message-oriented technology does not have any type description which object-technologies require. A language specification, the Message Description Language, describes the internal data format of each message buffer.

The CICS adapter uses the Message Description Language to describe the CICS buffer.

Message Description Language elements are message buffers, sent or received by the CICS adapter, mapped as Message Description Language method arguments. The mapping allows object-oriented technologies to have a familiar view of the message buffers; with each message treated as a single argument or separated into multiple arguments. The CICS adapter automatically concatenates the arguments at run-time. The request and reply messages are grouped as a single method with input and output arguments. One Message Description Language interface groups Message Description Language methods (performing similar tasks) for a specific message queue.

Classes

To make CICS servers visible as components to Oracle9*i*AS InterConnect applications, you first describe a set of methods using adapter’s message formats. A method call translates into a request message and a reply message. The request message contains all the input arguments and the reply message contains all the output arguments.

The CICS adapter uses Message Description Language `*.cls` files as the representation of component interfaces with methods having elements as arguments. For example, the message definition:

```
method GetBalance
    in BankName bank
    in CustName customer
    out Balance balance
    out CustStatus status
end method
```

defines a method containing four arguments with the type defined using Message Description Language fixed length string types:

```
typedef string(54,' ',tail) BankName
typedef string(30,' ',tail) CustName
typedef string(20,' ',head) Balance
typedef string(20,' ',tail) CustStatus
```

The following is an example of a Message Description Language file for CPI-C LU6.2 SNA protocol. This file must be named `cics62b.cls` (this is the class name and the cls extension):

```
#cics62b.CLS
# Class for CICS A62B transaction invoking ACTB62P1 Program
#   Note the ACTB62P1 program is invoking (CICS LINK) ACTBNKP1 Program

class cics62b (lu62cpic://CICSVIET)

      struct dfhcommarea
          string(8,' ',tail) transCode
          number(5,0,none) in string(5,'0',head) acctNumber
          string(20,' ',tail)clientName
          number(6,2,none) in string(8,'0',head)Amount
          number(8,2,none) in string(10,'0',head)Balance
          string(3,' ',tail) ReturnCode
          string(80,' ',tail) Information
      end struct
```

```
    method AB62
         return void
         inout dfhcommarea programdata
    end method

end class

        #           01  IOAREA.
        #           05 TRANSACTION-CODE     PIC X(8).
        #           05 ACCOUNT-NUMBER       PIC 9(5).
        #           05 CLIENT-NAME          PIC X(20).
        #           05 TRANSACTION-AMOUNT   PIC 9(6)V99.
        #           05 ACCOUNT-BALANCE      PIC 9(8)V99.
        #           05 APPL-RETURN-CODE     PIC X(3).
        #           05 APPL-ERROR-MESSAGE   PIC X(80).
```

LU6.2 CPI-C Protocol Stack, ECI Protocol Stack, and URLs

At run-time, the CICS adapter interacts with the communication framework to pass along data over the adapter CPI-C protocol stack or the ECI protocol API**.** You can use either of these protocol stacks with the CICS adapter.

Both the ECI protocol API and the CPI-C protocol stack (CPI-C LU6.2 SNA protocol API) defines the transport layer used to communicate data between CICS applications and adapter. Just as the metadata describes how to format documents, the protocol stack defines the shipping mechanism.

Message Description Language class files must specify a URL in the class definition. URLs specify routing information when describing destinations within the CICS world. The URL contains CPI-C transport, ECI transport, and protocol under the form of a protocol stack identifier, as well as transport specific server identification. The following is an example using the CPI-C LU6.2 SNA protocol API:

```
lu62cpic://luname/tpname
```

where:

- lu62cpic—Specifies the IBM snalu62 protocol stack, using the CPI-C API.
- Luname—Alias of the remote location unit name of the destination.
- tpname—CICS transaction name.

The following is an example using the ECI protocol API:

```
eci://cicsservername/cicsprogramname
```

where:

- eci—Specifies the ECI protocol API.
- cicsservername—Specifies the CICS server name.
- cicsprogramname—Specifies the CICS program name.

How the CICS Adapter Communicates With CICS

The CICS adapter uses either a SNA LU62 CPI-C interface or ECI protocol application to communicate with CICS.

Using SNA LU 6.2 CPI-C Protocol API

To achieve the communication, install the CICS adapter on a machine that has the IBM eNetwork communication server or the IBM eNetwork communication client, or on a machine having a Microsoft SNA server or Microsoft SNA client.

Install all the required definitions in the following locations:

- Local SNA Server
- Remote SNA Server (Virtual Telecommunications Access Method / Network Program Control)
- CICS tables

To communicate with CICS, the CICS adapter needs two pieces of information:

1. Remote LU alias—The name defined in the local SNA server. Usually it is the same name as the remote LU name (this name is often also the name of the CICS in VTAM (VTAM ACB)).
2. transaction program Name—The name of the CICS transaction. Its length is usually four characters.

The URL specified in the Message Description Language class definition provides this information.

Using the ECI Protocol API

To achieve the communication, install the CICS adapter on a machine that has the IBM CICS Universal Client. This software may require one of the following pieces of software to communicate with the CICS server:

- TCP/PI—This protocol can be used with non-mainframe CICS server.
- TCP62—This protocol can be used with all CICS servers. It requires IBM Personal Communication software installed on the computer running ACB with the CICS adapter using the ECI protocol.
- SNA LU6.2—This protocol can also be used with all CICS servers. It requires IBM eNetwork Communication Server client or server software installed on the computer running ACB with the CICS adapter using the ECI protocol.

To communicate with CICS, the CICS adapter needs two pieces of information:

1. CICS server name—The name defined in the CICS Universal Client configuration. It is the server name.
2. CICS program name—The name of the CICS program. Its length can be up to eight characters. This is the name of the program as defined in the CICS region

The URL specified in the Message Description Language class definition provides this information.

CICS Adapter Security

Security is provided by the CICS adapter and by the different software needed by the protocols.

Using the LU 6.2 CPI-C Protocol

There are different levels of security when using the CICS adapter with the SNA LU 6.2 CPI-C protocol. Security may be optional, but it is almost always used in mainframe applications. You may also have security between the SNA Client and the SNA server (if you are using a SNA client) and security between SNA servers (the mainframe SNA server and the SNA server used by the CICS adapter to communicate. For more details refer to your system administrator or to the SNA (Microsoft or IBM) books for the CICS LU6.2 CPI-C protocol and to the IBM SNA and CICS mainframe books.

Security can be specified in the user profile. If you are using security, you must provide a user identification and a password that your mainframe application accepts.

See Also: "Using the Configuration Editor" on page 7-1

Using the ECI Protocol

There are different levels of security when using the CICS adapter with the ECI protocol. Security may be optional, but it is almost used in mainframe applications. The security used in the CICS Universal agent depends on which communication protocol is used. For more details, refer to your system administrator or to the CICS Universal Agent documentation.

Security can be specified in the user profile. If you are using security, you must provide a user identification and a password that your mainframe application accepts.

See Also: "Using the Configuration Editor" on page 7-1

Implementing the CICS Adapter

There are two parts in the implementation:

- LU 6.2 protocol or the ECI protocol API
- CICS adapter

SNA LU 6.2 CPI-C Protocol API

LU 6.2 provides the services required to establish a conversation with the Remote partner of the CICS adapter, the Mainframe CICS region. The services related to a transaction program are:

- Start transaction program (identify to LU 6.2 a transaction program that can issue ALLOCATE or MC_ALLOCATE).
- Stop the transaction program.
- Data related to a transaction program—transaction program identification and transaction program behavior (dead or alive).
- Services related to a Conversation:
  - Receive_Allocate (start an invoked transaction program)

  - Send Data
  - Receive data
  - All the other APPC conversation verbs
- Data related to a conversation—Conversation Identification, conversation behavior (dead or alive), and conversation states (reset, receive, or send state).

ECI Protocol API

ECI provides the service requires to establish a Distributed Program Link (DPL) call to a CICS program running in a CICS region, through the Commarea. The services related to a Distributed Program Link call are:

- Provides security information if required.
- Establish contact with the CICS Universal Agent.
- Do the call to a remote CICS program, by passing to the CICS Universal agent:
  - Name of the CICS server.
  - Name of the CICS program.
  - Commarea containing the application data.
  - User Id and password, if required.

CICS Adapter

The CICS adapter in inbound mode does the following:

- Uses services provided by the SNA LU 6.2 CPI-C protocol or by the ECI protocol.
- Provides UserID and Password for security (at data communication message level).
- Builds from the metadata and the request received messages sent to the CICS region.

If using services provided by the SNA LU 6.2 CPI-C protocol, the following is achieved:

- Allocates a conversation with the remote Transaction Program (CICS region).
- Sends the required data and security.

- Receives reply (replies) from the CICS program.
- Converts the data to the original format.

If using services provide by the ECI protocol:

- Connect with the CICS Universal Agent (using the CICS server Name and security).
- Sends the required data using to the CICS program.
- Receives reply (replies) from the CICS region.
- Converts the data to the original format.

CICS Adapter Information Flow

The following is the CICS adapter information flow:

- Receives a request from the Oracle9*i*AS InterConnect application.
- Transforms the component (in a CICS transaction and in procedures to obtain what it is required).
- For SNA LU 6.2 CPI-C:
  - Transaction definition
  - Transaction code and input data (format) (fields position + type: binary, ASCII, EBCDIC)
  - Transaction output data format
  - Transaction destination and protocols
  - LU name (Transaction destination)
  - Protocol used (LU 6.2 CPI-C)
  - Security (application level—logical to be mapped in configuration data (logonids or others)
- For ECI:
  - Commarea definition
  - Program name and input data (format (fields position & type: binary, ASCII, EBCDIC)
  - Message Destination (CICS Server name)

  - Security (application level—logical to be mapped in configuration data logonIds or others)
- Sends the result of the transformation to the Communication layer.
- Receives transaction responses from the Communication layer.
- Transforms the transaction response.
- Sends the response to the Oracle9*i*AS InterConnect application.
- Uses functions library for sending / receiving data (CPI-C and / or APPC or ECI Protocol API).

The Communication Layer manages physical communication with the mainframe (physical links, PUs and logical units activation / deactivation, and link security). It sends and receives data received from the CICS adapter.

Multi-Threading

The following are multi-threaded for SNA LU62 CPI-C:

- CICS
- SNA servers (VTAM and Windows NT)

The SNA API DLLs support multiple calls from a program using APPC or CPI-C SNA APIs.

More than one instance of the CICS adapter are possible:

- one to one
- one to two or plus
- two or plus to one
- two or plus to two or plus

It is possible for one instance of the CICS adapter to have more than one conversation with multiple remote transaction programs.

Note: Implementation of security and / or implementation of multi-threading is specific to the SNA server and to the functions provided by the API. For example, Microsoft provides a Windows standard APPC where they allow asynchronous APPC calls on Windows 3.1.

The following are multi-threaded for ECI:

- CICS
- The CICS Universal Agent
- Windows NT

The CICS Universal Agent DLLs support multiple calls from a program using ECI API.

More than one instance of the CICS adapter is possible. The possibilities are:

- one to one
- one to two or plus
- two or plus to one
- two or plus to two or plus

It is possible for one instance of the CICS adapter to have more than one program call to multiple CICS regions.

Implementation of security and/or implementation of multi-threading is specific to the CICS Universal Agent and to the functions provided by the API.

Using the CICS Adapter Inbound

Sending messages inbound means that the CICS adapter is the client and CICS is the server. To send messages to CICS using the CPI-C LU 6.2 SNA protocol, ensure that the SNA client and the adapter configuration settings are setup properly. To use an ECI protocol, ensure that the IBM Universal client and adapter settings are set up properly.

SNA LU 6.2 CPI-C Protocol

Before using the CICS adapter, you must prepare your environment. For example:

- SNA controllers are up and running.
- CICS at the mainframe is up and running.
- Session(s) between LUs are either active or inactive. If sessions are active, the Bind security was done at session activation.

Application Start-up

Launch the CICS adapter. In its initialization process, the CICS adapter sends a `TP_START APPC` verb. The SNA returns a `TP_id`. All conversations and commands sent to the SNA controller use this `TP_id`. It is valid until the adapter issues a `TP_END` later in the allocation of a conversation.

Receiving a CICS Adapter Request from Oracle9*i*AS InterConnect

When the CICS adapter receives a request, complete the following:

1. Extract the data required to build the CICS transaction.
2. Get and set all related settings for the conversation:
   - Security information—`UserID` and `Password`.
   - ModeName—Characteristics of the session between the 2 LUs.
   - Synchronization level of the conversation—`NONE` or `CONFIRM`.
   - Remote LU name—The SNA name of the CICS region.
   - Remote transaction program name—For CICS, he CICS transaction name.
   - Start (allocate) the conversation—The protocol obtains from SNA a conversion identifier. This identifier is used on each subsequent call to the SNA LU62 CPI-C API for this conversation.
   - Data is sent to the CICS region through SNA.
3. Issue a `confirmation` command if `CONFIRM` is set as synchronization level.
4. Issue a `receive` command to receive the reply from the CICS transaction.

   When all the data is received, the session will be de-allocated and the CICS adapter receives notification. With the data, the CICS adapter builds back a reply to the requestor.

ECI Protocol

Before using the CICS adapter, you must prepare your environment. For example:

- CICS Universal agent is up and running.
- Communications software used by CICS agent is up and running.
- CICS at the mainframe is up and running.

Application Start-up

To start the application, launch the CICS adapter.

Receiving a CICS Adapter Request from Oracle9*i*AS InterConnect

When the CICS adapter receives a request, complete the following:

1. Extract the data required to build the CICS program Commarea.
2. Initialize and set all the control information, such as:
   - CICS Server Name—Name of the CICS server, as known by the CICS Universal Agent.
   - CICS Program Name—Name of the CICS program, as known by the CICS region.
   - If required, enter the User Id and Password.
3. Send the data to the CICS region.
4. The CICS program in the CICS Region receive data in a memory buffer called Commarea. It processes the data and puts back output data in the same buffer. When finished, control returns to the CICS Server.
5. The CICS server sends back the Commarea to the CICS Universal Agent.
6. The CICS Universal Agent passes the buffer back to the CICS adapter.

Design Time

Create an Message Description Language `*.cls` file describing the messages buffer format to send and receive as Message Description Language method argument parameters.

See Also:

- "Message Description Language Reference" on page 6-1
- "Classes" on page 3-3 for an example of an Message Description Language class file

Create a new sub-directory under config/CICS and copy the Message Description Language class file into the new directory. After the Message Description Language files have been copied into the directory, the interfaces are visible to Oracle9*i*AS InterConnect. Now iStudio can be used in the normal manner to create definitions, procedures, and events.

You use the CICS adapter to:

1. Expose the Message Description Language interface in iStudio.
2. Define application views in iStudio.

Runtime

Performing a call requires a bidirectional exchange of information with the CICS servers. On performing a call, the CICS adapter extracts all input information from the passed in arguments; it uses information within the Message Description Language to format these arguments into an input message. The CICS adapter uses the URL provided in the Message Description Language file to connect to the service.

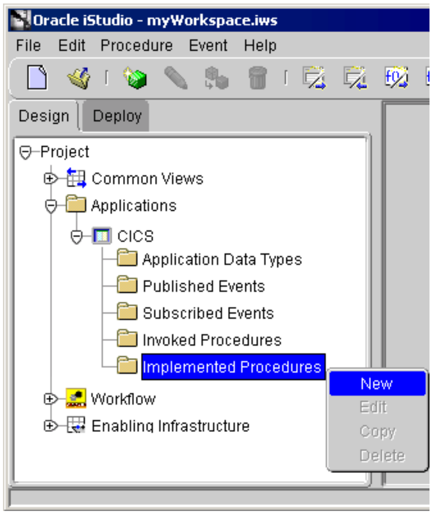
Creating an Implemented Procedure

To create an implemented procedure using iStudio:

1. Start iStudio.
2. Open your project.
3. Expand the Applications folder.

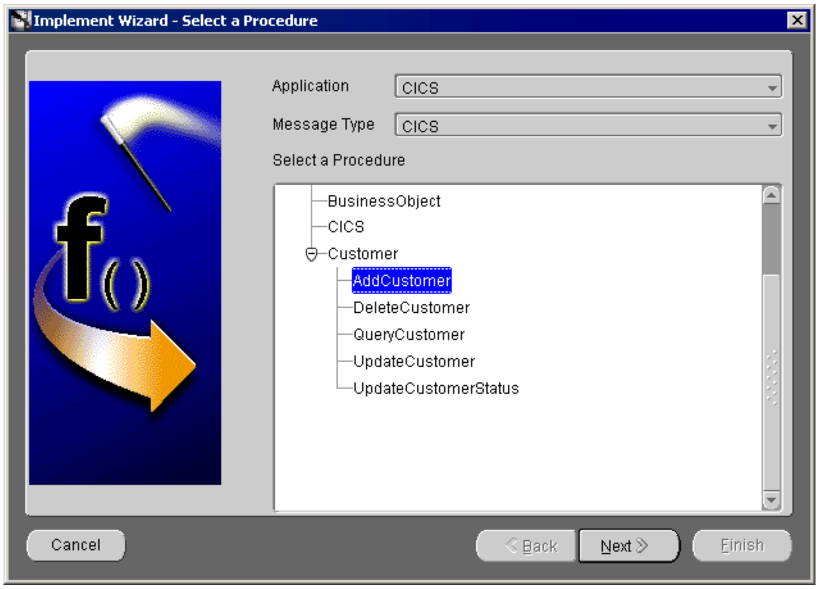
4. Right-click **Implemented Procedures** and select **New**.

Figure 3–1 iStudio - New Implemented Procedure



The Implement Wizard—Select a Procedure dialog displays.

Figure 3–2 Selecting a Procedure


Implement Wizard - Select a Procedure

Application: CICS

Message Type: CICS

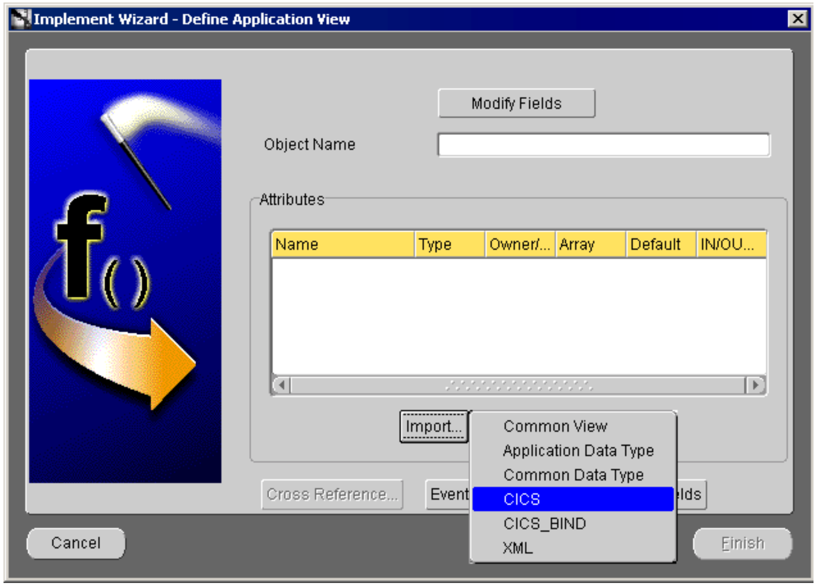
Select a Procedure

```
—BusinessObject
—CICS
—Customer
   —AddCustomer
   —DeleteCustomer
   —QueryCustomer
   —UpdateCustomer
   —UpdateCustomerStatus
```

5. Select the **Application** and **Message Type** from the dropdown lists.

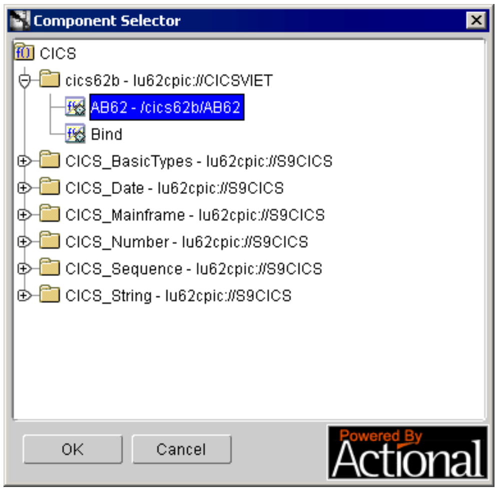
6. Select a procedure and click **Next**. The Implement Wizard—Define Application View dialog displays.

Figure 3–3 Implement Wizard - Define Application View - Importing CICS



7. Click **Import** and select **CICS** from the dropdown list. The Component Selector dialog displays.

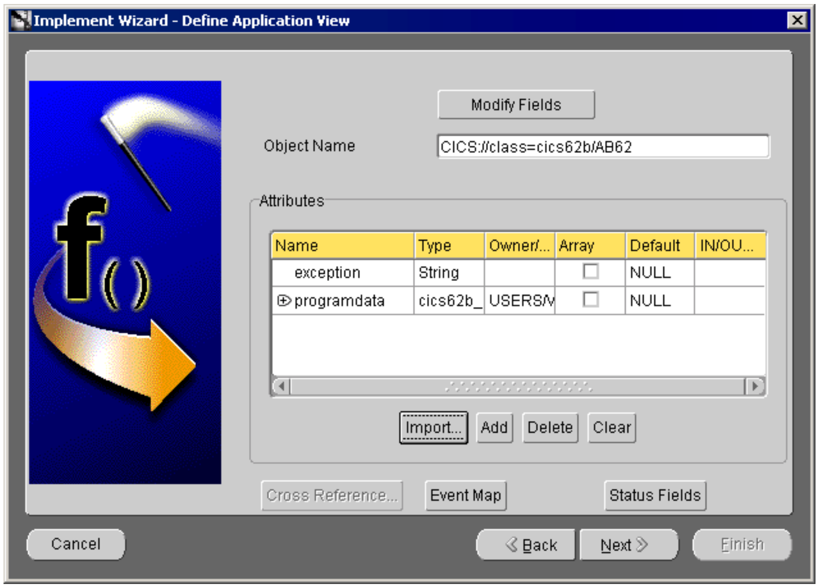
Figure 3–4 Component Selector


Component Selector
CICS
cics62b - lu62cpic://CICSVIET
AB62 - /cics62b/AB62
Bind
CICS_BasicTypes - lu62cpic://S9CICS
CICS_Date - lu62cpic://S9CICS
CICS_Mainframe - lu62cpic://S9CICS
CICS_Number - lu62cpic://S9CICS
CICS_Sequence - lu62cpic://S9CICS
CICS_String - lu62cpic://S9CICS
OK
Cancel
Powered By Actional

8. Expand the **CICS** tree to display the component for selection.

9. Select a component and click **OK**. The populated Define Applications View dialog displays.

Figure 3–5 Implement Wizard - Define Application View Dialog



Object Name: CICS://class=cics62b/AB62

Attributes

Name	Type	Owner/...	Array	Default	IN/OU...
exception	String		☐	NULL	
⊕ programdata	cics62b_	USERS/V	☐	NULL	

10. Click **Next** to define the mappings.

    The Define Mappings dialog displays.

11. Click **New** to define mappings and click **Finish**.

    The new populated event displays in the right panel of iStudio.

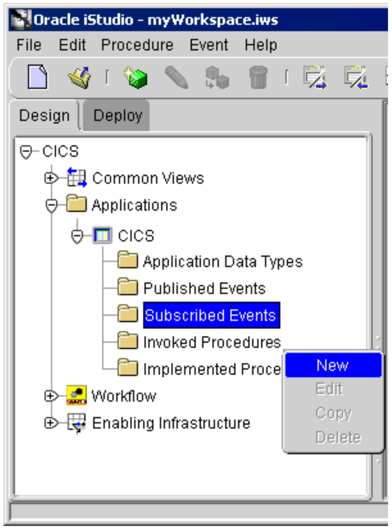
Creating a Subscribed Event

To create a subscribed event in iStudio:

1. Start iStudio.
2. Open your project.
3. Expand the **Applications** folder.

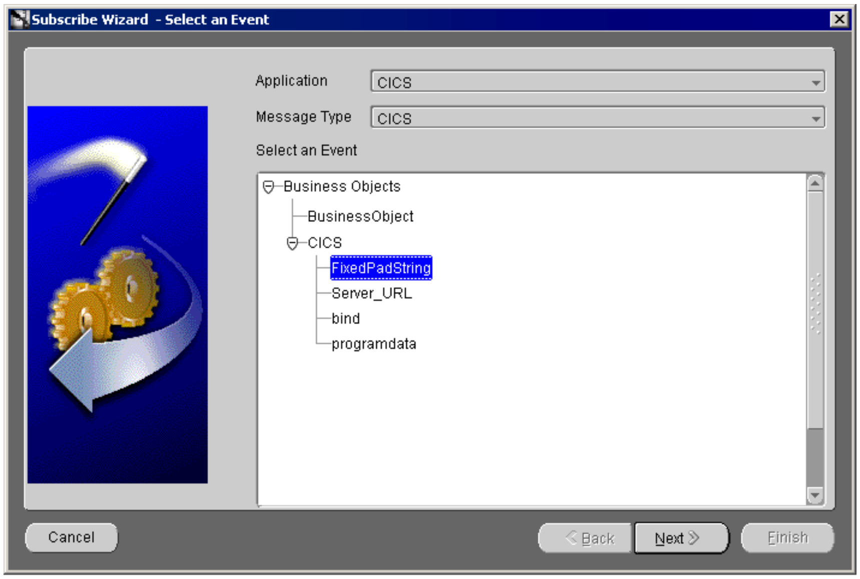
4. Right-click **Subscribed Events** and select **New**.

Figure 3–6 iStudio—Creating a Subscribed Event



The Subscribe Wizard—Select an Event dialog displays.

Figure 3–7 Select an Event



Subscribe Wizard - Select an Event

Application: CICS

Message Type: CICS

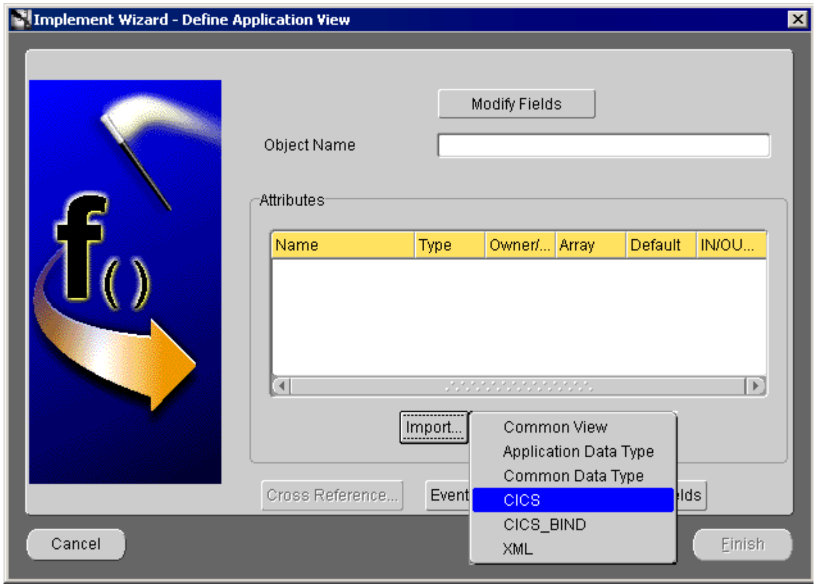
Select an Event

```
Business Objects
    BusinessObject
    CICS
        FixedPadString
        Server_URL
        bind
        programdata
```

5. Select the **Application** and **Message Type** from the dropdown lists.
6. Select an event and click **Next**.

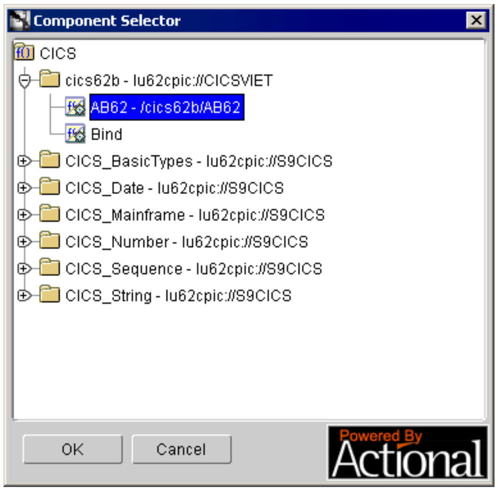
The Define Application View dialog displays.

Figure 3–8 Subscribe Wizard - Define Application View - Importing CICS



7. Click **Import** and select **CICS**. The Component Selector dialog displays.

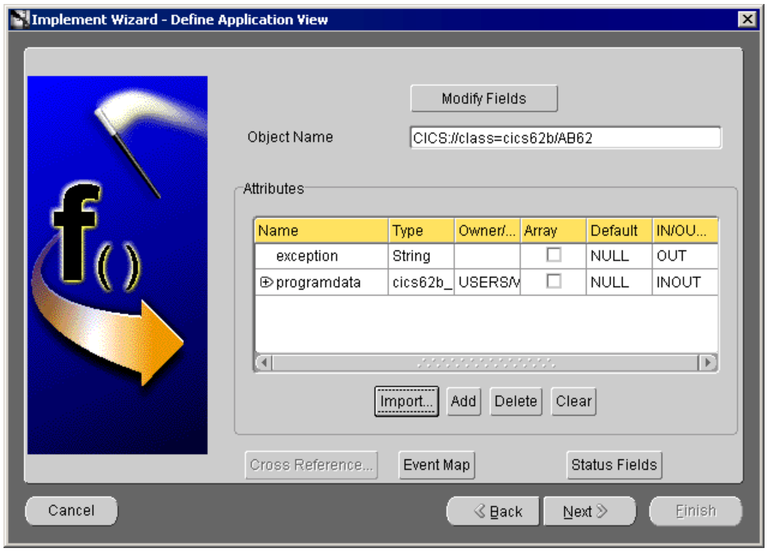
Figure 3–9 Component Selector


Component Selector
CICS
cics62b - lu62cpic://CICSVIET
AB62 - /cics62b/AB62
Bind
CICS_BasicTypes - lu62cpic://S9CICS
CICS_Date - lu62cpic://S9CICS
CICS_Mainframe - lu62cpic://S9CICS
CICS_Number - lu62cpic://S9CICS
CICS_Sequence - lu62cpic://S9CICS
CICS_String - lu62cpic://S9CICS
OK
Cancel
Powered By Actional

8. Expand the **CICS** tree to display the correct component for selection.

9. Select a component and click **OK**. The populated Define Applications View dialog displays.

Figure 3–10 Subscribe Wizard - Define Application View



Object Name: CICS://class=cics62b/AB62

Name	Type	Owner/...	Array	Default	IN/OU...
exception	String		☐	NULL	OUT
⊕ programdata	cics62b_	USERS/V	☐	NULL	INOUT

10. Click **Next** to define the mappings.

    The Define Mappings dialog displays

11. Click **New** to define mappings and click **Finish**.

    The new populated event displays in the right panel of iStudio.

4

Systems Network Architecture Definitions

This chapter describes the terms commonly used when referring to systems network architecture. This chapter discusses the following topics:

- LU 6.2 CPI-C Protocol Stack and URLs
- Concepts and Terms

LU 6.2 CPI-C Protocol Stack and URLs

At run-time, the CICS adapter interacts with the communication framework to pass along data over the CPI-C protocol stack.

The CPI-C protocol stack (CPI-C LU 6.2 SNA protocol API) defines the transport layer used to communicate data between CICS applications and the CICS adapter. Just as the meta-data describes how to format documents, the protocol stack defines the shipping mechanism.

URLs specify routing information when describing destinations within CICS. The URL contains CPI-C transport and protocol under the form of a protocol stack identifier, as well as transport specific server identification. The following is an example using the CPI-C LU 6.2 SNA protocol API:

```
lu62cpic://luname/tpname
```

where:

- `lu62cpic`—Specifies the IBM snalu62 protocol stack, using the CPI-C API.
- `Luname`—Alias of the remote logical unit name of the destination.
- `tpname`—CICS transaction name.

A default URL must be specified on the class definition. It must contain the protocol name (logical unit name) for the CPI-C LU 6.2 SNA protocol. If the `tpname` (lu62cpic) is not specified, then the method name is used.

For example:

```
class CICS_BasicTypes(lu62cpic://S9CICS) ascii littleendian
  Method testsigned8
   return void
   in signed8 inArg
   out signed8 outArg
  end method
```

Concepts and Terms

This section describes the following concepts:

- ABEND
- Advanced Program-to-Program Communication (APPC)
- CICS
- CICS Region
- CICS Transaction
- Conversation
- CPI Communications (CPI-C)
- Logical Unit
- Mode Name
- Node
- Node Type
- Physical Unit (PU)
- Session
- System Management Facility (SMF)
- Systems Network Architecture (SNA)
- Systems Network Architecture (SNA) Controllers
- Synchronization Levels
- Synchronization Services
- Transaction Program (TP)
- Verb Control Block (VCB)
- Verbs
- CICS as a Transaction Program

ABEND

In OS/390, it is an abnormal termination of a program task (thread) or an address space (process). There are two types of abends: system and user. A system abend is done by the system because a system request (this is a supervisor call, usually done by a SVC) cannot be completed; therefore, the program that issued it cannot continue to work. Examples are unconditional requests for memory or problem accessing files. A user abend is an abend generated by the application program. This occurs when issuing a SVC number 13 instruction in the executable code. A system abend is prefixed by the letter S and followed by three hexadecimal digits, for example, `S80A` (missing memory). A `User` abend is prefixed by the letter U followed by four decimal digit, for example, abend `U1001`.

Note: These abends can be trapped (caught) by user written programs.

In CICS, it is an abnormal termination of a transaction. When CICS has an OS/390 abend, the entire CICS region is not available. When a CICS transaction abends, only that transaction is terminated. A CICS abend code is usually made up of 4 characters, for example, abend ASRA. This abend is usually a program exception (for example, divides by zero, invalid addressing, non decimal data in a packed decimal field, and so forth). These types of errors are “trapped” by CICS and converted in a CICS abend ASRA. If, for example, the packed decimal error is not trapped, the whole CICS region abends with a OS/390 system abend code S0C7.

Advanced Program-to-Program Communication (APPC)

Advanced Program-to-Program Communication is the general facility characterizing the LU 6.2 architecture and its various implementations in products.

APPC is sometimes used to refer to the LU 6.2 architecture and its product implementations as a whole, or to a LU 6.2 product feature in particular, such as an APPC application program interface. In this document, APPC is referred to as the API, which allows a program to communicate with another program via a LU 6.2. This API is implemented as APPC verbs. Transaction programs can directly use these verbs to communicate with the LU 6.2 or they can use another layer of API, such as CICS. CPI-C is an example of a higher layer-programming interface.

CICS

CICS is a transaction-oriented system. Basically, a transaction is entered via a terminal or programmatically. The data entered contains a transaction ID, which enables CICS to recognize the program to be executed. CICS provides this data as input to the called program. Processing includes calls to databases, to other programs, or even to other systems. Next, a reply is built and the data is sent back via CICS. The receiving program reads the data and proceeds with it. If it is a terminal, control characters may have been embedded with the data to display the data correctly.

To communicate with other terminals, programs, or both, CICS can use a large number of protocols, including TCP/IP and systems network architecture protocols. In this case, LU 6.2 is a peer-to-peer protocol used to transmit messages between programs.

CICS Region

The CICS region refers to CICS and ESA only. In MVS (or OS/390), a variable-size subdivision of virtual storage that is allocated to a job step or system task. CICS/ESA runs in an MVS/ESA region, usually referred to as the CICS region.

A named collection of resources controlled by CICS as a unit. The collection includes programs, BMS map sets, transactions, terminals, files, transient data queues, temporary storage queues, journals, products, and users. One installation of CICS can run a number of regions on the one processor. Regions are likely to be application-specific, but one clear distinction is between a production region and a test region.

CICS Transaction

A CICS transaction is a unit of application data processing, consisting of one or more application programs, initiated by a single request, often from a terminal.

Conversation

In systems network architecture, conversation describes the communication between two transaction programs. That is, when two APPC transaction programs are in communication, they are said to be holding a conversation. Conversations flow on LU-LU sessions. Each conversation is allocated a session for its own private use. When the conversation ends, the session is free to be used by another conversation. There can only be one conversation between any two transaction programs, but one transaction program could have multiple conversations with different transaction programs. LU 6.2 transaction programs may select either two-way alternate (half-duplex) or two-way simultaneous (full-duplex) conversations, if both the local and remote logical units support full-duplex conversations. In a full-duplex conversation, each transaction programs can send data simultaneously. In half-duplex, the transaction program doing the allocation is in `send` state at the beginning and the other transaction program is in a receive state. There are 2 types of conversation:

- APPC mapped conversation—The systems provide and interpret protocol headers, and the application programs deal only with user data.
- APPC basic conversation—The sending application must prefix the data with the header required by the communications protocol. The receiving application must interpret this header.

In CICS, the communication commands you code in your application depend on whether you intend to use basic or mapped conversations. CICS-to-CICS applications need only use mapped conversations. Basic conversations (also referred to as unmapped) are useful only when communicating with systems that do not support mapped conversations. These include some APPC devices.

The two conversation types are similar. The main difference is in the way user data is formatted for transmission:

- In mapped conversations, the application sends the data to the partner.
- In basic conversations, the application has to add a few control bytes to convert the data into an systems network architecture-defined format called a generalized data stream (GDS).

CPI Communications (CPI-C)

CPI Communications provides a cross-system-consistent and easy-to-use programming interface for applications that require program-to-program communication. From an application's perspective, CPI-C provides the function necessary to enable this communication. The conversational model is implemented in two major communications protocols: Advanced Program-to-Program Communication (APPC) and Open Systems Interconnection Distributed Transaction Processing (OSI TP). The APPC protocol is also referred to as logical unit type 6.2 (LU 6.2). CPI-C provides access to both APPC and OSI-TP.

Logical Unit

A logical unit represents the logical destination of a communication data flow. The formal definition of an logical unit is that it is the means by which an end user gains entry into a network, and an end user is defined as the ultimate source, or destination, of data flow in a network. Systems network architecture supports several different types of logical units. These are grouped together in numbered logical unit types, such as logical unit type 2 for 3270 display terminals, and logical unit type 4 for printers. The logical unit type for CICS-to-CICS communication is logical unit type 6.2, and is frequently referred to as advanced program-to-program communication (APPC). Each logical unit is given a unique name that identifies it in the network, and this is referred to as the logical unit name. There are two types of LU 6.2 pertinent to CICS adapter:

- Dependent LU 6.2—Can have only a single session and, therefore, only one conversation at a time.
- Independent LU 6.2—Can have more than one session with other logical units. Therefore, many conversations can be held simultaneously between 2 logical units.

Mode Name

A mode name is the name used by the initiator of a session to designate the characteristics desired for the session, such as traffic pacing values, message-length limits, synchronization point and cryptography options, and the class of service within the transport network.

Node

A node is any device attached to a network that transmits and receives data.

An endpoint of a link or a junction common to two or more links in a network. Nodes can be processors, communication controllers, cluster controllers, or terminals, and can vary in routing and other functional capabilities.

Node Type

A designation of a node according to the protocols it supports or the role it plays in a network. Node type was originally denoted numerically (as 1, 2.0, 2.1, 4, and 5) but is now characterized more specifically by protocol type (APPN network node, LEN node, subarea node, and interchange node, for example) because type 2.1 nodes and type 5 nodes support multiple protocol types and roles.

Physical Unit (PU)

A Physical Unit (PU) is the hardware and software components in a device that manages its network resources. Logical units reside within a physical unit, and one physical unit may hold many logical units. There are several different types of physical units: Virtual Telecommunications Access Method (VTAM) running in a mainframe host is a physical unit type 5, and Network Program Control (NCP) running in a 37x5 network controller (physical unit type 4). When workstations connect together in a peer-to-peer manner they act as physical unit type 2.1. When a workstation connects to a mainframe host in a hierarchical manner, it acts as a physical unit type 2.0. The physical unit type 2.1 is described as an independent node (because it is independent of a mainframe host), and the physical unit type 2.0 is a dependent node.

Session

Systems network architecture uses the term session to refer to various types of data flow in a network. To avoid ambiguity, it should always be qualified by a description of the type of data flow, for example CP-CP session. However, when used by CICS for APPC, it can be assumed to refer to data flow between logical units, and therefore is a LU-LU session. There are usually several sessions between any two (independent) logical units, and these are known as parallel sessions. CICS uses the term connection to refer to a group of sessions that connect two CICS systems (or a CICS with the CICS adapter logical unit).

System Management Facility (SMF)

A System Management Facility (SMF) is a standard feature of OS/390 that collects and records a variety of system and job-related information.

Systems Network Architecture (SNA)

Systems Network Architecture (SNA), in the mainframe work, are commonly used to:

- Enable the reliable transfer of data between end users.
- Provide protocols for controlling the resources of any specific network configuration.

Systems Network Architecture (SNA) Controllers

In this document, the systems network architecture controller represents the type 2.1 node (or physical unit) when on the CICS adapter side as well as the type 5 node, when at the mainframe site. Examples of systems network architecture controller include:

- Type 2.1—Microsoft SNA Server. The LU 6.2 is also part of Microsoft SNA Server.
- Type 5—VTAM. This is the software running at the mainframe. For some telecommunications, VTAM requires a Type 4 physical unit. This is hardware equipment (the IBM 37x5 families).

Synchronization Levels

In synchronization levels, CICS defines three levels of synchronization for conversation using the APPC protocol:

- Level 0—None. There is no CICS support for synchronization of remote resources on connected systems. However, it is still possible, under the control of the application to achieve some degree of synchronization by interchanging data, using the `SEND` and `RECEIVE` commands.
- Level 1—Confirm. Special commands for communication between the two conversation partners can be used. One transaction can confirm the continued presence and readiness of the other. Both transactions are responsible for preserving the data integrity of recoverable resources by issuing synchronization point requests a the appropriate times.
- Level 2—Sync point. (Sync level 2 is not supported on single-session connections). All synchronization point requests are automatically propagated across multiple systems. CICS implies a synchronization point when it starts a transaction; that is, it initiates logging of changes to recoverable resources, but no control flows take place. CICS takes a synchronization point when one of the transactions terminates normally.

  One abending transaction causes all to rollback. The transactions themselves can initiate synchronization point or rollback requests. However, a synchronization point or rollback request is propagated to another transaction only when the originating transaction is in conversation with the other transaction, and synchronization level 2 has been selected.

Sync point and rollback are not limited to any one conversation within a transaction. They are propagated on every conversation currently active at synchronization level 2.

APPC provides support for the three levels of synchronization by providing synchronization verbs and resynchronization services. Synchronization level 2 services is an option for many systems network architecture communication servers.

Synchronization Services

When a failure occurs, an application transaction program may be accessing multiple resources that may be local or remote, which causes synchronization services to happen. Local resources reside on the same node as the application transaction program. Remote resources may or may not be on the same node as the application transaction program. The function of logical unit synchronization point services is to ensure that selected local and distributed resources are in consistent states at defined synchronization points even if failures occur. Resources within such a set have consistent states if all the actions affecting them since the last synchronization point persist or if none persist. If all persist, the changes are said to be committed at all resources. If none persist, the changes are said to be backed out, for example, all the resources are returned to their states at the last synchronization point. Resources that are kept consistent by using the synchronization point protocols are called protected resources. Following a transaction program, session, logical unit, or other protected resource failures, that occur during synchronization point protocols, protected resources are returned to consistent states by the LU 6.2 partners using resynchronization (resynchronization) protocols.

Full support of synchronization point services in actual implementations includes provisions for synchronizing local resources as well as distributed resources accessed through conversations. An application transaction program may use synchronization point services when it is not using protected conversations. For completeness, this section describes general synchronization point services. Details of synchronization point services, including resynchronization services, for resources other than LU 6.2 conversations are not defined in this document.

A transaction program selects the synchronization point service for a conversation by specifying the `SYNCPT` value of the `SYNC_LEVEL` parameter on the `ALLOCATE` verb. With other values of the `SYNC_LEVEL` parameter (`NONE` and `CONFIRM`), maintaining resource consistency is up to the application transaction program.

If a transaction program has conversations using a synchronization level of `SYNCPT`, it may use the `SYNCPT` and `BACKOUT` verbs to establish synchronization points. The

BACKOUT verb undoes all changes made to protected resources since the last synchronization point. The SYNCPT verb invokes two-phase commit protocols to commit changes to local and distributed resources. Two outcomes to the SYNCPT verb are possible:

- The changes may all be committed, establishing a new synchronization point.
- The changes may all be backed out, restoring the old synchronization point.

Application transaction programs execute a sequence of logical units of work (LUWs), with each unit of work consisting of some changes to the resources under the control of the transaction programs. If a synchronization level other than SYNCPT is used, a transaction consists of one logical units of work. In this case, recovery from a failure can be done by undoing the work accomplished up to the point of the failure and running the transaction again from the beginning. By using synchronization point services, a transaction can consist of multiple logical units of work that are delimited by the start-up of a transaction program and by the execution of each SYNCPT or BACKOUT verb. At the beginning of each logical units of work, all resources are in consistent states. As a result, the amount of work required to recover from a failure can be limited using synchronization point services.

The following failures are addressed by synchronization point services:

- Transaction program failures happen when transaction programs end abnormally. LU 6.2 synchronization point services return protected resources to consistent states following a transaction program failure.
- Conversation failures happen when conversations fail as a result of failure of the underlying sessions caused by the failures of physical components over which the sessions are carried. If protected resources are used by the transaction program, the transaction program can issue (and sometimes must issue) the BACKOUT verb to put resources into consistent states following a conversation failure. If a synchronization point operation was in progress when the conversation failed, resynchronization returns protected resources to consistent states.
- Logical unit failures happen sometimes by themselves or as a result of the failure of underlying hardware or software. The logical unit failure appears to another logical unit as failures of all sessions connecting the two logical units. After the logical unit recovers and sessions are established, resynchronization may be needed to return protected resources to consistent states.

- Local resource failures (files). Some implementations may reduce the frequency of these failures by having dual-copy file support. If the local resource is protected by the synchronization point service, recovery is managed by synchronization point services cooperating with the local resource manager.

Transaction Program (TP)

Transaction program (TP), in systems network architecture, the transaction program refers to the application program in an APPC environment. The transaction program uses the LU 6.2 (APPC) to gain access to the network.

CICS provides a choice of two application programming interfaces (APIs) for coding your DTP conversations on APPC sessions:

- CICS API, the programming interface of the CICS implementation of the APPC architecture. It consists of EXEC CICS commands. These CICS commands are converted in APPC verbs (as defined below).
- Common Programming Interface Communications (CPI Communications) is the communications interface defined by the Systems Application Architecture (SAA). It consists of a set of defined functions in the form of program calls, that are adapted for the language being used.

Verb Control Block (VCB)

Verb Control Block (VCB) is a structure passed to the APPC function. All the calls done to APPC require only one parameter: a pointer to a verb control block. The verb control block is different depending on the type of the call, but the first parameter is the operation code, telling APPC which APPC verb the CICS adapter wants to perform, and, at the same time, the format of this verb control block. There are two types of APPC verbs:

- Blocking Verb—Does not return before the completion.
- Nonblocking Verb Support—Enables the transaction program to issue a conversation verb and return control prior to the completion of the verb. The verb whose execution is left incomplete becomes an outstanding verb. A conversation can have more than one verb outstanding at a time. The completion of the verb can be checked later with a `WAIT_FOR_COMPLETE` verb. `WAIT_FOR_COMPLETION` waits for posting to occur on one or more nonblocking operations represented in the specified list of wait objects. Posting of a nonblocking operation occurs when the logic unit has completed the associated nonblocking verb and filled all the return values.

Verbs

APPC Verb is the mechanism by which a program accesses APPC. Each verb supplies parameters to APPC. There are three types of APPC verbs:

- Management Verbs—Provide the following management functions:
    - ACTIVATE_SESSION
    - CNOS (Change Number of Sessions)
    - DEACTIVATE_SESSION
    - DISPLAY
- Transaction program (TP)—Transaction program verbs start and end transaction programs and get and set transaction program properties. The following are transaction program verbs:
    - GET_TP_PROPERTIES
    - SET_TP_PROPERTIES
    - TP_ENDED
    - TP_STARTED
- Conversation Verb—Enable transaction programs to allocate and deallocate conversations, send and receive data, and change conversation states. The conversation verbs are listed in the following table.

    There are two groups of conversation verbs:

    - Mapped conversation verbs—Intended for programs that use the conversation directly.
    - Basic conversation verbs—Intended for more complex programs that provide services to other users.

    In typical situations, end-user transaction programs use mapped conversations and service transaction programs use basic conversations. Mapped conversation verbs can only be issued by a transaction program in mapped conversations, while basic conversation verbs are reserved for basic conversations. There is one exception to this rule: `ALLOCATE` can be used to start either a basic or a mapped conversation.

Table 4–1 Conversion Table

Mapped conversation verbs	Basic conversation verbs
MC_ALLOCATE	ALLOCATE
MC_CONFIRM	CONFIRM
MC_CONFIRMED	CONFIRMED
MC_DEALLOCATE	DEALLOCATE
MC_FLUSH	FLUSH
MC_GET_ATTRIBUTES	GET_ATTRIBUTES
MC_POST_ON_RECEIPT	POST_ON_RECEIPT
MC_PREPARE_TO_RECEIVE	PREPARE_TO_RECEIVE
RECEIVE_ALLOCATE	RECEIVE_ALLOCATE
MC_RECEIVE_AND_POST	RECEIVE_AND_POST
MC_RECEIVE_AND_WAIT	RECEIVE_AND_WAIT
MC_RECEIVE_IMMEDIATE	RECEIVE_IMMEDIATE
MC_RECEIVE_LOG_DATA	RECEIVE_LOG_DATA
MC_REQUEST_TO_SEND	REQUEST_TO_SEND
MC_SEND_CONVERSATION	SEND_CONVERSATION
MC_SEND_DATA	SEND_DATA
MC_SEND_ERROR	SEND_ERROR
MC_TEST_RTS	TEST_RTS

Other conversation verbs (mapped or basic) include:

- GET_LU_STATUS
- GET_STATE
- GET_TYPE

Mapped and basic verbs have the same function in their respective types of conversation. For example, `MC_CONFIRM` performs the same function in a mapped conversation that `CONFIRM` performs in a basic conversation.

CICS as a Transaction Program

CICS can utilize the following APIs to issue APPC verbs:

- EXEC CICS Command Interface—CICS is “mapping” the CICS command to an APPC verb.
- CPI-C interface.

When CICS is using its CICS command interface, the CICS adapter should use the APPC verb to communicate with the CICS transaction. It is not mandatory, the idea is to try to use interface (or API) that supports all the functions the other transaction program support.

5
Systems Network Architecture Concepts

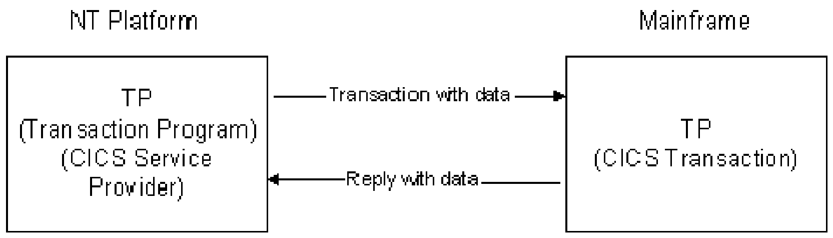
This chapter describes typical data flows between two systems network architecture transaction programs. The following topics are discussed:

Data Flow

Figure 5–1 displays a simple exchange of data between the following two transaction programs:

- The CICS adapter
- A CICS transaction on the mainframe

Figure 5–1 Windows NT Platform to Mainframe Flow



Logical Units and Parallel Sessions

LU 6.2 can provide more than one session (used by the conversation). These sessions are:

- Long-lived—Activated on demand by a command or automatically when a there is request to start a conversation. They can also be called when an `Allocate` command is received.
- The maximum number of sessions is a parameter—The maximum session is 32767 (for VTAM) for a LU 6.2.

There is a limitation of 999 sessions for the mainframe CICS LU 6.2. CICS can receive up to 999 conversations concurrently from one to `n` other LU 6.2 connection. The session can be reused by conversations.

CICS Adapter Conversations

Since it is possible that a request from an adapter may require more than one transaction, it is possible to reuse the same conversation (by not de-allocating the session). By definition, sessions are long-lived connections between the two logical units and a conversation should be allocated and deallocated as soon as possible, to allow other conversations to start. For example a terminal operator who forgot to close a session may hold a conversation open.

It is possible for the CICS adapter to use the same conversation (without de-allocating it and reallocating it again). Therefore, similar credentials from different source use the same one. Modification must be made in the CICS application (application written using CICS as APPC API). If the application in CICS issues a `EXEC CICS SEND LAST` command, it tells CICS that it will deallocate the session (by an `EXEC CICS FREE`) later.

If the application instead loops backs to an EXEC CICS RECEIVE and waits for new data, then the conversation could be reused if the program supports the new transaction code.

One of the ways to know if a CICS program loops back is to define it in the metadata. This allows the CICS adapter to know if it can issue more than one transaction without reallocating a conversation. Also, the CICS transaction may deallocate the session when:

- The CICS adapter has no choice.
- It must deallocate locally the session and re-allocate one for a new transaction.

Reusing the same conversation drives more complexity in the CICS adapter because it knows when it can use an already open conversation. In addition, there are security issues. If the same `UserID` is not used, it starts a new conversation and there are conversations management issues.

On the other hand, if the application on the mainframe was designed to be called from a concentrating server, it may expect several transactions to be sent on the same conversation, to increase efficiency.

Security

Security has always been an important matter in mainframe. Almost every resource in a mainframe can be protected. While in the past, applications programs were doing their own security, you can now use software packages that help to protect almost all types of mainframe resources. IBM integrated a security interface in the operating system and different vendors (including IBM) uses that interface for implementation. CICS is using the same interface. In short, defining it and allowing users, with the correct profile, can protect resources. VTAM (systems network architecture controller) is also using it. CICS, like the others packages, is using its own security scheme. However, since CICS version 9.0.2, it is using an External Security Manager (ESM) to protect resources. The ESM used in CICS mainframe documentation is RACF, the IBM ESM.

LU 6.2 Security

The LU 6.2 architecture defines a number of conversation-level security option sets that include passwords, UserIDs, and profiles in allocation requests. The LU 6.2 architecture also defines a session-level security option set. The architecture requires that session-level LU-LU verification be allowed when conversation-level security option sets are enabled and when the logical units that make up the network are not physically secure (as determined by installation management). In an IBM mainframe using OS/390 for example, VTAM, in essence, is part of the systems network architecture Server and is primarily responsible for handling the LU 6.2 security. It supports session-level security and offers pass-through support for conversation-level security. the application programs are responsible for implementing conversation-level security. In this case, the application program is CICS at the mainframe.

Session Level Security

Session level security includes the following:

- Session Level Cryptography
- LU-LU Verification

Session Level Cryptography

Session level cryptography refers to the enciphering of all or selected user data, at the source logical unit, and the later deciphering that occur at the target logical unit. The encryption algorithm uses a cryptographic key, supplied by the control point, and a session seed, generated by one of the logical units when the session is started. These parameters are exchanged at session activation.

LU-LU Verification

The identity of a logical unit's partner is verified by using a LU-LU password and the Data Encryption Standard (DES) algorithm.

Conversation Level Security

Conversation-level security includes end-user verification, already-verified protocols, persistent verification, and password management. The following terms are associated with conversation level security:

- End-User Verification
- Already-Verified Protocols
- Persistent Verification
- Password Expiration Management

End-User Verification

End-user verification confirms the identity of the partner end user. When a transaction program requests access to another transaction program, it must supply adequate security information in the request to satisfy the security requirements of the other transaction program, or the request is rejected. Security information, here, could be the user ID and password supplied by the end user in its `ALLOCATE` verb initiating the Attach request between the two logical units. When a user ID and password are supplied on the request, they are verified by the logical unit that receives them. If the UserID and password combination is incorrect, the request is rejected. Also, an authorization list associated with the target transaction program can be used. The keys to search the authorization list would be combinations of the UserID and an optional profile supplied on the request, along with the name of the partner logical unit from which the request originated. The authorization list could be made up of combinations of UserID, profile, and partner logical unit name. After the UserID and password combination is verified by the logical unit, the authorization list may be searched using the received UserID and / or profile for access rights to the specific transaction program named in the request. If the additional criterion is not met, the request is rejected.

Already-Verified Protocols

A transaction program in its invocation of partner transaction programs may represent an end user whose identify has been verified locally and need not be verified at each remote partner, provided that partner trusts the invoking transaction program's logical unit. In this case, the Attach invoking the partner transaction program need not carry the already-verified password of the represented UserID. Instead, an already-verified indicator is set in the Attach request; the UserID and optional profile of the user represented by the invoking transaction program are supplied in the request. For security reasons, the password used to initiate the invoking transaction program is never saved. However, the UserID and optional profile used to initiate the invoking transaction program, are saved. The already-verified indicator can be used only if the sender of the indicator is trusted by the receiver of the indicator to have performed the proper verification of the UserID and password that initiated the sender. This level of trust is installation defined at the receiver of the indicator and communicated to the sender of the indicator during session activation in the `BIND/RSP(BIND)` exchange.

Persistent Verification

Persistent verification (PV) is one way of reducing the number of password transmissions, by eliminating the need to provide a UserID and password on each Attach (Conversation request) during multiple conversations between a user and its partner at a remote logical unit. The user is verified during a sign-on process preceding its initial conversation and remains verified until being signed off by the remote logical unit, which may occur as the result of an explicit request (triggered by a `SIGNOFF` verb issued at the remote logical unit), or because no active sessions remain between the user's logical unit and the remote logical unit.

Password Expiration Management

Password expiration management involves request and reply exchanges between two programs. The first program is a sign-on requester service transaction program and the second is a sign-on server service transaction program (called the Sign-On / Change-Password TP) identified by the registered transaction program name `X'06F3F0F1`'. The requester program invokes the server by an `Attach` carrying this registered transaction program name.

CICS Security Implementation

CICS defines APPC sessions, connections, and partners as resources, all of which have security requirements. CICS provides the following security mechanisms for the APPC environment:

- Bind-time security, or in systems network architecture terms session level security, to prevent an unauthorized connection between two LU 6.2. This security check is done when a session is opened between the two LU 6.2 sessions.
- Link security defines the authority of the remote system to access transactions or resources to which the connection itself is not authorized.
- User security checks that a user is authorized both to attach a transaction and to access all the resources that the transaction is programmed to use.

Link and User security are a CICS implementation of the APPC conversation level security.

Bind Time Security

An eight character (or 16 hexadecimal digit) `password` is used by both partners to authenticate. The check is done when a new session is created. All binds can be audited, since it is recorded in mainframe SMF files. This type of security is done at the systems network architecture controller level. This security check is not associated with a UserID; both ends of the connection must have the same session key. In CICS, the key is kept in a resource definition in its ESM. On the other end, the key is defined in the remote APPC logical unit in the Microsoft systems network architecture server. The Bind security is done at Bind time (when a session is activated) and it is kept as long as the session is active.

Security For CICS in General

Each link between systems is given an authority defined by a UserID. It is important to note that users cannot access any transactions or resources over a link that is itself unauthorized to access. This means that each user's authorization is a subset of the link's authority as a whole.

To limit the remote system's access to your transactions and resources, you use link security. Link security is concerned with the single user profile that you assign to the remote system as a whole. Similar to user security in a single-system environment, link security governs the following:

- Transaction security—Controls the link's authority to attach specific transactions.
- Resource security—Controls the link's authority to access specific resources. This applies to transactions, executing on any of the sessions from the remote system, that have RESSEC(YES) specified in their transaction definition.
- Command security—Controls the link's authority for the commands that the attached transaction issues. This applies to transactions, executing on any of the sessions from the remote system, that have CMDSEC(YES) specified in their transaction definition.
- Surrogate user security—Controls the link's authority to `START` transactions with a new UserID, and to install resources with an associated `UserID`.

Security Specific to LU 6.2

Link security further restricts the resources 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 UserID has no authority. Link security can be associated with a connection or a session, depending on 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 (this is a CICS definition equivalent to the remote LU 6.2).
- To define link security for individual groups of sessions within a connection, specify the UserID in the `SESSIONS` definition as a UserID (`SESSIONS` definition in CICS is the same as `SESSIONS` in LU 6.2).

Each link between systems is given an authority defined by a link UserID. A link UserID for LU 6.2 is a UserID defined on your session's definition for this connection. If not defined, the link UserID is the `SECURITYNAME` UserID specified on the connection definition. If there is no `SECURITYNAME`, the link UserID is the default UserID. The CICS default UserID is the UserID used by CICS when a resource check has to be done and there is no other UserID that CICS can use.

User Security

User security causes a second check to be made against a user signed onto a terminal, in addition to the link security described in Link security. A conversation allocation is related to a CICS transaction and security can apply at that level. Again a UserID and a password may be used to protect the CICS transactions being invoked in the conversation. CICS also uses that UserID to protect files and other resources. The user defined for link security must also have the same access that the UserID used in the user security.

User security can be implemented in five different ways however, only one of these options can be selected:

- LOCAL—Specifies that a UserID is not to be supplied by the remote system, and if one is received, the attach fails. CICS makes the user security profile equivalent to the link security profile. You do not need to specify `ESM` profiles for the remote users. `LOCAL` is the default value.
- IDENTIFY—Specifies that a UserID is expected on every attach request. All remote users of a system must be identified to the `ESM`. If an attach request with both a UserID and a password is received on a link with

ATTACHSEC(IDENTIFY), CICS does not reject the attach request. CICS handles the attach request as if the connection was defined with ATTACHSEC(VERIFY).

- VERIFY—If a UserID and an invalid password, or a UserID and no password is received for verification, the attach is rejected. If no UserID is received, CICS applies the security capabilities of the default user. The rules that apply to the checking of the UserID for ATTACHSEC(IDENTIFY) also apply for ATTACHSEC(VERIFY). If a valid UserID is received but the password verification fails then CICS rejects the attach request.
- PERSISTENT VERIFICATION—Specifies that a UserID and a user password are required with the first attach request for a new user, but all following attach requests for the same user need supply only a UserID. (All remote users of a system must be identified to the ESM.) The first attach signs on the user, even if the attach request is later unsuccessful because the user is not authorized to attach the transaction.
- MIXIDPE—Specifies that the sign-on level for the remote user is determined by parameters sent with the attach request. The possibilities are PERSISTENT or IDENTIFY.

Synchronization of Changes

Systems network architecture defines three levels of synchronization for conversation using the APPC protocol:

- Level 0 - None—At sync level zero (0), there is no CICS support for synchronization of remote resources on connected systems. However, it is still possible, under the control of the application to achieve some degree of synchronization by interchanging data, using the SEND and RECEIVE commands.
- Level 1 - Confirm—At sync level one, you can use special commands for communication between the two conversation partners. One transaction can confirm the continued presence and readiness of the other. Both transactions are responsible for preserving the data integrity of recoverable resources by issuing synchronization point requests at the appropriate times.
- Level 2 - Syncpoint—At sync level 2, all syncpoint requests are automatically propagated across multiple systems. CICS implies a syncpoint when it starts a transaction which initiates logging of changes to recoverable resources, but no control flows take place. CICS takes a syncpoint when one of the transactions terminates normally. One abending transaction causes all to rollback. The transactions themselves can initiate syncpoint or rollback requests. However, a

syncpoint or rollback request is propagated to another transaction only when the originating transaction is in conversation with the other transaction, and sync level 2 has been selected.

Syncpoint and rollback are not limited to any one conversation within a transaction. They are propagated on every conversation currently active at sync level two.

Error Handling

This section describes the types of errors, how to handle them, and how to find errors. As there are many pieces of hardware and software involved, errors can be generated from many sources: Application program, CICS, VTAM, NCP, the systems network architecture Controller, or the CICS adapter. There are two type of errors:

- Application Error—An error detected by the application and reported in a data message returning to the CICS adapter.

  CICS program issues an `Issue Signal (APPC verb MC_REQUEST_TO_ SEND)`: this advises the partner that it wants to send data, even if it is still in `Receive` state. The partner may or may not respond to the request.

  The CICS program issues an `Issue error (APPC verb MC_SEND_ ERROR):` It requires immediate attention from the partner logical unit.

  The CICS program issues an `Issue Abend (APPC verb MC_DEALLOCATE TYPE(ABEND_PROG)`. This command not only signals a problem but also ends the conversation. It is a severe error condition.

- System Errors—Errors are usually detected by other components than the application system. Examples of system errors include:

  - CICS application program abended. Abend is a term defining a program which is terminated by a control software instance (CICS for the application program, or OS/390 for the CICS program). Usually the local LU 6.2 is advised of the condition and is able to send to the partner logical unit and application program a message telling the condition.
  - CICS itself abended—Causes all the conversations in progress to be abended).
  - VTAM problems (physical unit type five) or NCP problems (physical unit type four).
  - Systems network architecture controller, including the local LU 6.2.

6 Message Description Language Reference

This chapter describes message description language and its concepts. The following topics are discussed:

What is Message Description Language?

Message oriented technology does not have any type description which object-technologies, like Oracle9*i*AS InterConnect, require. The language specification describes the internal data format of each message buffer. The CICS adapter uses the message description language to describe the CICS buffer.

Message description language elements are message buffers sent or received by the CICS adapter mapped as message description language method arguments. The mapping allows object-oriented technologies to have a familiar view of the message buffers, with each message treated as a single argument or separated into multiple arguments. The CICS adapter automatically concatenates the arguments at run-time. The request and reply messages are grouped as a single method with input and output arguments. One message description language interface groups message description language methods (performing similar tasks) for a specific message queue.

Message Description Language Classes

To make CICS servers visible as components to Oracle9*i*AS InterConnect, you must first describe a set of methods using message description language. A method call translates into a request message and a reply message. The request message contains all the input arguments and the reply message contains all the output arguments.

The CICS adapter uses message description language `*.cls` files as the representation of component interfaces with methods having elements as arguments. For example, the message definition:

```
method GetBalance
  in BankName bank
  in CustName customer
  out Balance balance
  out CustStatus status
end method
```

defines a method containing four arguments with the type defined using message description language fixed length string types:

```
typedef string(54,' ',tail) BankName
typedef string(30,' ',tail) CustName
typedef string(20,' ',head) Balance
typedef string(20,' ',tail) CustStatus
```

Message Description Language File

An message description language file is a text file with an `*.cls` extension. It contains four types of declarations:

- Class Declaration
- Typedef Declaration
- Struct Declaration
- Method Declaration

Every message description language file requires a class declaration; it is always the first declaration in an message description language file. All other declarations are written within it. The `typedef` and `struct` declarations are optional. All declarations reside on separate lines, there are no delimiters (such as semi-columns, or commas) required in any declarations.

Class Declaration

A class describes the set of functions and class specific complex types. Classes are always the topmost level of a CLS file. They are declared with a `class` *classname* `[(default URL)] [endianness]` and `[character encoding]` declarator, and are terminated with an `end class` terminator. All class declarations should be on a separate line and the class declarator and terminator must be defined on separate lines. Interfaces may not be nested. A single public class declaration may reside within a CLS file and the name of the file must match the name of the class. Class definitions may contain:

- Type definitions (typedef declaration):
  - Structures definitions (struct declaration)
  - Method definitions (method declaration)
- Layout of a class declaration:

```
class class name [(default URL)] [endianness][character encoding]
    struct declarations...
    typedef declarations...
    method declarations...
end class
```

where:

- `class name`—The name of the class. This name must be the same as the file name.
- `(default URL)`—Specifies the default URL on the class declaration line.
- `[endianness]`—The `bigendian` and `littleendian` keywords act as endian convention `gateways`, and they specify in which format the CICS adapter sends the binary integral types. Use these keywords when the CICS adapter receives integral types regardless of its platform‘s convention. For example, if all integral types sent to the CICS adapter are always in the `bigendian` convention, prefix the message description language endainness with the `bigendian` keyword. Similarly, if all integral types sent from the CICS adapter are in the `bigendian` convention, declare message description language endianness on message description language class declarations such as `bigendian.littleendian` modifiers follow the same logic. In the absence of such keywords, the CICS adapter treats these types as opaque entities and provides them the same way as it receives them—which means the CICS adapter always expects to receive and send integral types using the platform‘s format where the CICS adapter is running.

  See Also: "Integral Types" on page 6-10 for an explanation of binary integral types

- `[character encoding]`—The ASCII and EBCDIC keywords act as character encoding convention `gateways`. They specify in which format the CICS adapter sends and receives the string type. Use these keywords when the CICS adapter receives string types of a certain convention regardless of your platform’s convention.
- *struct declarations*—A declaration for a collection of variables grouped together for convenient handling.
- *`typedef` declarations*—Provides a new name for an existing type.
- *method declarations*—Sends messages through the messaging system and receive replies by abstracting incoming and outgoing messages as method input and output parameters.
- `end class`—The class terminator.

Typedef Declaration

Typedef declarations do not create a new type; their purpose is to provide a new name for an existing type. The following is an example of a `typedef` declaration:

```
typedef composite type new name
```

Struct Declaration

Structures can contain:

- Binary
- String
- Complex Types
- Predefined Structures

Structures are declared with a structure declarator `struct` *struct name* and a terminator `end struct` which must reside on separate lines. Structures may be nested using predefined structures.

The layout of a struct declaration is as follows:

```
struct struct name
   field declaration
   field declaration
end struct
```

Note: Field declarations are specific to the declared type, and are identical to the type declarations described in "Supported Data Types" on page 6-9.

Method Declaration

Methods describe the act of sending a message through the messaging system, and of receiving replies to those messages. The concatenation of the `in` and `inout` parameters form the contents of a message sent to the destination specified by the URL. The `return`, `out`, and `inout` parameters form the reply message and the `in` and `inout` parameters form the request message. Contextual information that is part of the requests and reply messages map as `inout` parameters. When mapping the reply message, the return value precedes the `out` and `inout` parameters, in the same parameter order. Methods map to corresponding methods in other systems as synchronous calls, which emit a blocking wait for the reply message.

Methods are described in message description language as method `<method name>` `[(method properties)]` `[async]` and terminator end method, residing on different lines. They always contain a return argument description on the next line (using the keyword return), and a list of argument declarations directly following the return statement.

The layout of a method declaration is as follows:

```
method <method name> [(method properties)] [async]
    return <return type>
    argument declarations
end method
```

where:

- *`<method name>`* is the name of the method.
- *`(method properties)`* provides the message-oriented server a list of properties. There is no property for the `lu62cpic` protocol.

The format of the properties list for the ECI protocol is:

```
(<property>=<value>...)
```

where:

```
    <property> is the name of the property.
    <value> is the value accepted by the property..
```

Table 6–1 describes method properties, an explanation, and an acceptable value.

Table 6–1 Method Properties

Method Properties	Explanation	Acceptable Values
`commarea`	The size of the commarea used by the remote CICS program. The maximum size is determined by CICS software, which is 32500 bytes	A number up to 32500

Note: The `commarea` size is calculated at run time. It is the actual size of the `in` and `inout` arguments. If the value specified in `commarea` is smaller than the computed value a warning message is included in the CICS adapter log file and the computed value is used. If a value greater than the acceptable maximum (32500) is used, a warning is also logged and the maximum is used.

- `[async]`—Indicates the Oracle9*i*AS InterConnect adapter will not wait for a server reply message and, in the case of a message Oracle9*i*AS InterConnect adapter, the CICS adapter will not send a reply message.
- *`<return type>`*—Returns a certain supported type or a void argument if is the method does not return anything.
- *`<argument declarations>`*—Arguments may consist of binary, string, and complex types, and structures.
- `End method` is the method terminator.

Return Type Declaration

Return type arguments may consist of:

- void arguments
- binary types
- string types
- complex types
- structures

It is recommended to always void for the return argument as most messaging systems do not have a notion of a return argument. If a method returns a certain supported type, the return declaration is formulated as the `return` keyword, followed by the type declaration.

See Also: "Supported Data Types" on page 6-9

For example:

```
return type declaration
```

A method that does not return anything must explicitly declare it to include the `void` keyword. For example:

```
return void
```

An EOL delimiter separates return declarations from the rest of the method declaration.

Argument Declarations

The syntax declaration of an argument declaration is the following:

```
  arg direction_type declaration
```

where:

- *`arg direction`*—Describes the argument direction and may be either of the keywords `in`, `inout` or `out`. The `in` keyword describes arguments whose contents are initialized by the client, and are useful to the server. The `inout` keyword describes parameters which may contain information for both the client and the server (a variable initialized by the client, and which may be modified by the server). The `out` parameter describes arguments initialized by the server and serves as a data recipient for information returned to the caller.

For example:

```
#describes an "in" parameter which contains data for the server
in type declaration

#describes an "out" parameter which serves as data recipient for the client
out type declaration

#describes an "inout" parameter which serves as data recipient for both the
server and the client
inout type declaration
```

Argument names are significant only to the systems to which the method that is exposed. An `EOL` (End of line) delimiter separates argument declarations.

Note: You can insert comments anywhere in an message description language file by inserting pound signs “#” at the beginning of a line. Throughout this chapter comments will precede the example.

Supported Data Types

An message description language file supports the following data types:

- Binary Types
- String Types
- Complex Types

Table 6–2 identifies the different types belonging to the three supported types:

Table 6–2 Supported Types

BINARY	STRING	COMPLEX
Floating Point	Length Prefixed	Date
Integral	Delimited	Numerical: ■ Floating Point ■ Fixed Scale, Variable Precision Numbers ■ Fixed Scale, Fixed Precision Numbers ■ Packed Decimal
	Fixed-Length Padded	Array (tables):‘ ■ Fixed Length Tables ■ Prefixed Length Tables ■ Explicitly Delimited Variable Length Sequences ■ Implicitly Delimited Variable Length Sequences
	Null Terminated	Structured Types
	Implicit	

Binary Types

Supported binary types must be in your system platform endianness, or in that endianness which is specifically indicated for the class it is used, align on a 1 byte boundary. They include integer and floating point, up to 32 bits.

See Also: "Class Declaration" on page 6-3

Binary types are all simple types of binary nature separated into two main subclasses:

- Integral Types—Binary types describing integral numbers.
- Floating Point Types—Real numbers, containing a mantissa and an exponent.

Integral Types

Integral types come in various formats: 8, 16, 32 bits signed or unsigned. They are always declared as their sign concatenated with their size in bits. For example:

```
signed8 aChar
signed16 aShort
signed32 aLong
unsigned8 aByte
unsigned16 aWord
unsigned32 aDoubleWord
```

Floating Point Types

There are two supported floating point types. 32 bit IEEE binary floating point is declared with the type specifier `single`, and 64 bit IEEE floating point is declared with the type specifier `double`. For example:

```
single myFloat
double aBigNumber
```

The run-time data format of binary floating point types always follow the IEEE binary standard.

Floating point types are generally mapped to similar (IEEE), binary entities in other systems.

String Types

All character types are assumed to be in your native system platform character set where the CICS adapter is running. The strings can be of fixed or variable length, with a length prefix or a terminating delimiter. Strings are declared with the keyword string and may come in five string styles.

Note: The presence of NULL characters in any of the following string types may cause unwanted behavior when the latter are mapped into other systems.

Length-Prefixed Strings

Length prefixed strings are variable length strings where a length specifier precedes the string at run-time. They are declared by following the string type with its length type specifier. The length specifier may be any of the numerical or integral types, and should immediately follow the string keyword. Prefixed string types do not take any parameters. For example:

```
string prefixed length type declaration_string name
```

where:

`length type declaration` may be any of the numerical or integral types, and consists in a standard type declaration.

For example:

```
#this string is length prefixed with a binary double-word
string prefixed unsigned32 myPrefixedString
#this string is length prefixed with a fixed numerical value
#for which there are 5 digits reserved for the integer part,
#no decimal digits or decimal separator enclosed in a dot
#delimited string.
string prefixed number(5, 0, none) in string(‘.‘)
  myNumPrefixedString
```

Length prefixed strings are expected to have their length prefixed with a numerical type or integral type (as described in the message description language) which should directly precede the string itself. They map such strings in other systems if they are available in the target system or as null terminated strings if they are not. The decimal component of numerical types passed as the string‘s length is always ignored if present.

See Also: "Complex Types" on page 6-13

Delimited Strings

Variable length strings for which the length is determined at run-time by the presence of a delimiter. They are declared the string type and have a type parameter for delimiter.

```
 string(delimiter ) string name
```

Delimiters are not considered part of the displayable string, and are generally replaced by null terminated strings in other systems, in which the delimiter has been removed.

For example:

```
  string(‘,‘) myCommaTerminatedString
```

Null Terminated Strings

Delimited strings map in general as null terminated strings, the EOS (End of String) delimiter is considered as an inherent part of the string and appear in the final data.

For example:

```
    string(0) myNullTerminatedString
```

Fixed-Length Padded Strings

A fixed string’s maximum length is known in advance. Declared by passing three parameters to the string type keyword, which are, respectively the size of the string, the padding character, and the padding convention. The padding convention parameter consists of the tail, head or none keywords, which indicates where the padding occurs. The none keyword indicates no padding occurs, and that the string is always assumed to take up the full fixed length (a date string, for example, may always contain a certain count of characters).

```
    string(str size, padding char, pad convention) name
```

For example:

```
#declares an 80 character wide string padded with spaces
   string(80, ‘ ‘, tail) myString

#declares a 40 character string front padded with spaces
   string(40, ‘ ‘, head) myString
```

```
#declares a 40 character wide string front padded with zeros
   string(40, ‘0‘, head) myString

#declares a 40 character wide string with no padding
   string(40, none, none) myString
```

Fixed length strings map fixed length strings if such notions exist in the target other system. They may also map into variable length strings for which the maximum length is the fixed length described in the message description language, and the actual length is always at maximum.

Implicit Strings

Implicit strings are implicit sequences where the base type is `signed8`. The declaration is string implicit. The CICS adapter assumes that meeting this type means that the entire data buffer is a string.

Implicit strings have the following limitations:

- Cannot be defined in arrays.
- If defined in a structure, it must be the only field.
- Must be the only in, out, or inout argument defined in a method.

Complex Types

Complex types come in as a composition of different or similar sets of string types and binary types.

Date Types

Date types are stored in any of the supported string types. The date type represents the date under the form of a fixed length string. The supported date formats are respectively DDMMYY, DDMMYYY, MMDDYY and MMDDYYYY. The date type parameter string defines the date field separators.

`date(`*`date format`*` ) in `*`string type`*`  myDateVar`

For example:

```
date("DD-MM-YY") in string(8,none,none) myYear2000bug
date("DD/MM/YYYY") in string (10,none, none)
  myYear2000compliantDate
```

```
date("MM DD YYYY") in string (10,none, none)
  mySpaceSepY2KCompDate
date("MM.DD.YY") in string(10,none,none) mydotSeparatedY2KBug
```

Separators are mandatory, although their nature may be of any sort.

When delimited strings are used, the `date` string should exclusively contain the date and delimiter, or have the date left aligned within the string.

Date types map other system date formats, if any exist. For example, they would map DATE structures in the case of COM, for example. If no such date formats exist, they map as variable length strings.

Numerical Types

Numerical types are formatted numbers stored within any of the supported string types (for example, numerical types are stored in ASCII). They always have a string declaration following their type declaration. The following lists supported numerical types:

- Floating Point Numbers
- Fixed Scale, Variable Precision Numbers
- Fixed Scale, Fixed Precision Numbers
- Packed Decimal

Floating Point Numbers

Represented as exponent based ANSI floats. They are declared as follows:

`number in string type` *`declaration name`*

For example:

```
number in string('*') MyStarTerminatedFloat
number in string(30,'', tail) MyFixedLengthFloat
number in string unsigned16 MyPrefixedLengthFloat

#This is a number within a prefixed string itself prefixed with a
#number within a prefixed string prefixed with a binary byte.
#This is not the best design, but still legal
number in string prefixed number in string prefixed signed8 MyNumber
```

Fixed Scale, Variable Precision Numbers

Strings in which the decimal separator may freely reside. They are declared as follows:

```
number (dec separator) in string type_name
```

where:

- *decimal separator* may be the keyword none, specifying an entirely integral number.
- *name* is the name of the defined type.

For example:

```
#declaration for the form 398.029
number('.') in string('_') myDotDecimalUnderscoreTerminatedNb
```

Note: When the decimal separator is specified, the maximum number of integral and integral precision is 7 digits. For example, 1234567.1234567

```
#declaration for the form word len "98372"
number(none) in string prefixed unsigned16
myIntegralWordPrefixedNumber
```

A fixed scale, variable precision number is always confined in the limits the string it resides in imposes. For example:

```
#This number may NOT exceed 20 characters.
#"12345678901234567890"
#The following is the declaration definition
number(none) in string(20,'0',head) my20CharNumber
```

Fixed Scale, Fixed Precision Numbers

Strings in which the number integral digits and decimal digit is constant. They are declared as follows:

```
number (Idigits, Ddigits, dec sep) in string type name
```

where:

- *Idigits* and *Ddigits* respectively represent the count of integer and decimal digits.

- *`decimal separator`* may be the keyword none, indicating that the decimal separator‘s position is implied in the number‘s format, and not expected in the run-time string.

```
name is the name of the defined type

#declaration in the form of "1234567890.12345_"
number (10, 5, ‘.‘) in string(‘_‘) myUnderscoreFixedNumber

#declaration of the form of "00000123456789012345" = 1234567890.12345
number(10,5,none) in string(20, ‘0‘, head) myNumber
```

As for the preceding `Number type`, the `fixed scale`, `fixed precision` numbers may be bounded by the string type in which case the decimal, then digits, should be truncated in order to fit inside the string constraints.

```
#declaration of the form of "123456789012" = 1234567890.12
number(10,5,none) in string(12, ‘0‘, head) myNumber
```

Packed Decimal

An internal representation of numbers. It is also called BCD (Binary Code Decimal). The field size is variable and can be determined by the number of digits divided by two (truncated) plus one. For example, if you have the number +123.45 the internal representation is 0x12345c and the length is 3 bytes. If you have the number -12.34, the internal representation is 0x1234D and the length is 3 bytes.

```
number (Ydigits, Zdigits) packed name
```

where:

- *`Ydigits`* and *`Zdigits`* respectively represent the number of integer and fractional digits.
- *`name`* is the name of the defined type.

Array Types (Tables)

Arrays come in four variants: fixed length, length prefixed, explicitly delimited, and implicitly delimited. Each element has the same type. Unless the array is explicitly delimited, there are no special delimiters between the elements of an array itself since the delimiters of the contained data type act as implicit delimiters. Array types consist of variable or fixed sequences of a certain type. They are defined with a base type (the sequence‘s element type) and of subscript operators, in the case of fixed arrays.

The base type may be any of the structures, binary, string, complex types, or tables:

- Fixed length tables map to bound sequences in other systems.
- All variable length tables map to unbound sequences in other systems.
- Tables may be nested in any given combination provided the inner tables' definition has been defined.

For example:

```
#variable length table of bytes length prefixed with a DWord
typedef table prefixed unsigned32 of unsigned8 varByteTbl_t
#fixed length table of the preceding table type
table(80) of varByteTbl_t myNestedTables

#This is incorrect syntax because a nested array where the inner array must be
predefined cannot use the table keyword table(80) of table prefixed unsigned32
of unsigned8 myNestedTables
```

Fixed Length Tables

Fixed length tables do not have any associated run-time data overhead.

`table(`*`subscript`*` ) of `*`base type declaration  table name`*

For example:

```
table(30) of signed8 myVariableByteTable
```

Prefixed Variable Length Tables

Prefixed variable length tables have an message description language described length indicator preceding the table at run-time.

`table prefixed `*`length type decl`*` of `*`base type decl name`*

For example:

```
#table of prefixed length with a word of null terminated string
typedef table prefixed unsigned16 of string(0) myT
```

Explicitly Delimited Variable Length Sequences

Explicitly delimited variable length sequences have a delimiter between each element.

`table(`*`cont del`*` , `*`end del`*` ) of `*`base type decl  name`*

For example:

```
#each element has a comma separator between them until
#the last element is reached, where a dot appears.
#declaration for the form dword , dword  ... dword .
table(‘,‘,‘.‘) of signed32 myDWordTbl

#declaration for the form string(‘.‘) , string(‘.‘)  " string(‘.‘) .
#example: "helloworld.,helloworld2.,helloworld3.."
table(‘,‘,‘.‘) of string(‘.‘) myDelimitedStringTbl
```

One message description language-specified delimiter is used to indicate the table‘s continuation while the other indicates a terminator. For example, the run-time format is:

`elmnt 1  cont del  elmnt 2  cont del  ... elmnt n  end del`

where:

- *`elemnt 1`* is the first element and *elemnt n* is the last.
- *`cont del`* is the continuation delimiter (first type parameter).
- *`end del`* is the end delimiter (second type parameter).

Implicitly Delimited Variable Length Sequences

Implicitly delimited variable length sequences terminate at the end of the provided buffer.

`table implicit of` *`base type decl  name`*

Example:

```
#This is the same as a string implicit
table implicit of unsigned8 myByteTable
```

Some of the reply messages have items in a sequence where the length is determined by how many items are in the message buffer (there is no length prefix). COM and CORBA arrays and sequences must have a length defined before they can be filled.

Structured Types

These are structures or records containing a set of named fields where each field can have a different type. Field types are binary, string, date, or numerical types. The syntax of the fields determines the layout of the structure. There are no special delimiters for fields of a structure itself, the delimiters of the data types of the fields themselves act as implicit delimiters between fields. Fields are not named within the message itself; they are intended for systems such as CORBA and COM.

Message Description Language File Format General Syntax Conventions

This section describes the general syntax conventions for the message description language file format.

Type parameters

Certain types require fully-defined parameters, such as fixed-length `table` (array) types, requiring subscript parameters, or fixed-length strings, requiring a description of their length and formatting style. Enclose these parameters within parenthesis "()" or brackets "[]". Separate the parameters by commas "," if more than one parameter qualifies the given type. Type parameters always directly follow the type they modify. When passing characters as parameters, pass them enclosed respectively in quotes ' ' and double quotes " ". If passing non-printable characters, type their binary values instead. Parameter ordering is specific to the type concerned.

For example:

The following is a null terminated string:

```
string(0) myString
```

The following declares a fixed array of 40 bytes:

```
table(40) of signed8 myCharTable
```

The following is equivalent to the preceding line:

```
table[40] of signed8 myCharTable2
```

The following is a date declaration:

```
date("DD/MM/YYYY") in string(10,' ',tail) myDate
```

Type modifiers

There are two different classes of type modifiers, Keyword and Type Declaration modifiers.

Keyword Modifiers

Modifiers, such as specifying the alignment specifiers for fixed length strings, always follow the type itself. Type keyword modifiers never have parameters.

The following is a fixed length string, of 34 bytes, aligned to the right and padded with zeros:

```
string(34, ‘0‘, head) myString
```

Type Declaration Modifiers

Certain types require other types to be defined (declared) in order to be completely defined, for example, when the primary type aggregates, is contained in, or prefixed by another type. Insert the keywords `of`, `in`, and `prefixed`, to describe these different cases. Define the additional types following the original type declaration. Tables, for example, contain elements of another type. To fully define the table, declare the elements within the type declaration.

The following declares an array of 40 bytes:

```
table(40) of unsigned8
```

The following declares an array of 40 dot delimited strings:

```
table(40) of string(‘.‘)
```

The following declares a variable length array of (variable length strings prefixed with dot delimited fixed numbers) length prefixed with a byte:

```
table of string prefixed number(5,5,none) in string
‘.‘)prefixed signed8
```

- Types which prefix another are either `numerical` or `binary` types.
- Types which aggregate (in) another are always string types.
- Types contained in (of) may be of any type.

Expression

Expressions are not supported by the message description language parser. For example, the following expression is valid:

```
string(25)myString
```

The following expression is invalid:

```
string(10 + 50)myOtherString
```

Alias

To create an alias for a given type, use a `typedef`:

```
typedef type declaration  typedef name
```

For example:

```
typedef string prefixed number(5,0,none) in string('.')
   myString_t
  struct myStruct
     MyString_t structName
   table(80) of myString_t myAddressList
end struct
```

`Typedef` declarations are always global to the file where they are defined. Their declarations must precede their usage.

Comment Insertion

Insert comments anywhere in the file by inserting pound signs "#" at the beginning of a line. For example:

```
   #commented line
   signed16 myShort this part is NOT commented
```

Case Sensitivity

The message description language is case sensitive, and its reserved keywords are always lower cased. Keywords with different case conventions as identifiers are legal. For example:

```
#valid statement
signed16 Signed16
#valid method
method Method
```

Message Description Language File Example

The following is an example of a message description language file called CLS.

```
#MyClassName.CLS
#Tabs (or spaces) and Extra EOLs are here only for readability.
#Keywords are in bold.
   class MyClassName
   typedef string(‘.‘) string_t
   typedef number(5,5,‘.‘) in string(11,‘ ‘,none) number_t
       #this structure declaration is private to MyClassName
struct Account
       string_t structName
       string_t clientName
       unsigned16 accntID
       number_t balance
   end struct
       typedef table(‘,‘,‘.‘) of Account AccountList_t
       struct BankInfo
       string_t structName
       string_t bankName
       AccountList_t accountList
       table(80) of string_t debtorNameList
       date("DD-MM-YY") in string_t lastModificationDate
  end struct
  #Get account info method
   method GET
       return void
       in string_t clientName
       out Account account
   end method
   #Add new account method
       method ADD
       return BankInfo bankInfo
       out string_t result
    end method
end class
```

7

Using the Configuration Editor

This chapter describes how to use the Configuration Editor to configure the CICS adapter. The Configuration Editor is only used at runtime. The following topics are discussed:

- Using the Configuration Editor
- Configuration Editor Login
- Configuration Editor Security

Using the Configuration Editor

Using the Configuration Editor, you can customize the settings to specify how the CICS adapter and Service Provider components interact with your system. You can change these settings by accessing the Editors through the Configuration Editor.

Note: Profiles and Deployment are sensitive to the Master Key setting. If using a shared machine, before accessing the Configuration Editor, ensure the Master Key is set to either that of User1 or create a new Master Key for your profiles.

To configure settings for the CICS adapter you must access the CICS Configuration Editor as follows:

1. Change directories to the `.../oai/9.0.2/config/configeditor` using a DOS prompt.
2. Type **configeditor** and press **Enter**.

   The Configuration Editor displays.
3. Click **Profile** and select **iStudio**.

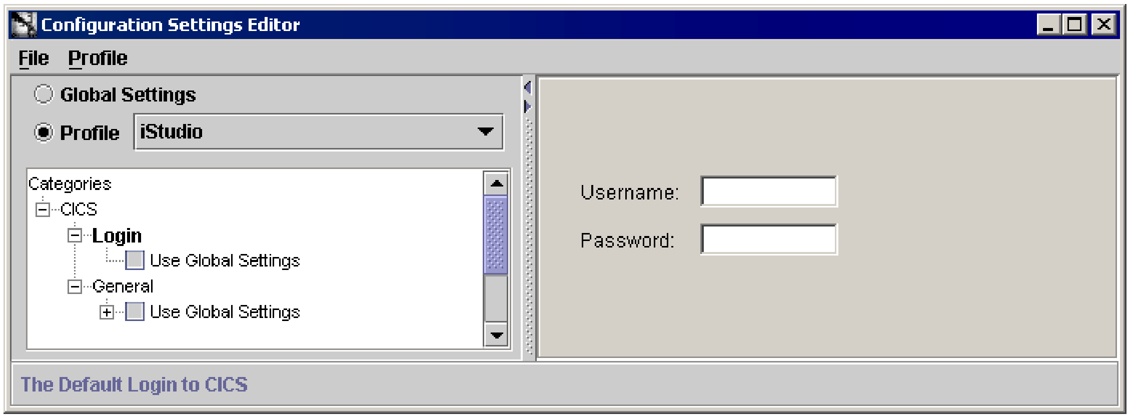
Note: Under some circumstances you may wish to run your adapter under a profile other than iStudio. This may be needed, for example, if you want to run two instances of the CICS adapter on the same machine. You may want to have two instances of the same type of adapter if these instances need to connect to different backend system installations. To accomplish this you need to create a new profile using the configuration editor and fill in the settings for this new profile. The name of the new profile should be the same as the name of the application. For example, if your application is called APP2, create a profile called APP2. Now APP2 will use the settings in the profile called APP2, whenever it runs.

---

4. Double-click on **CICS** to edit the CICS configuration settings for iStudio profile.
5. Click to expand the **Login** node.

6. Click to expand the **General** node.

Figure 7–1 Configuration Settings Editor - Expanding the General folder



Global Settings

● Profile iStudio

Categories
- CICS
  - Login
    - Use Global Settings
  - General
    - Use Global Settings

Username:

Password:

The Default Login to CICS

7. Click to unselect **Use Global Settings** in both the Login and General dialogs.

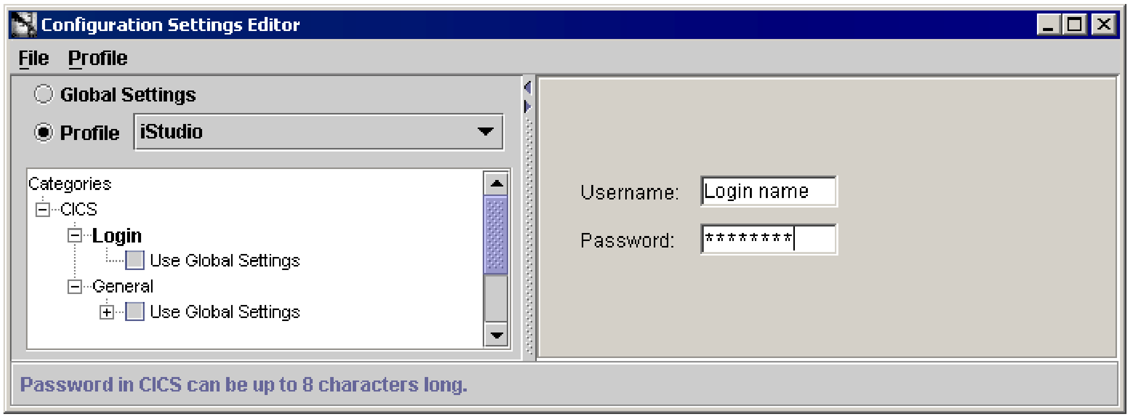
Configuration Editor Login

In the CICS adapter, the configuration dialog allows the user to set the login and password for the CICS region.

To set the login information:

1. Click **Login** on the Login dialog.

Figure 7–2 Configuration Editor Login Screen



iStudio

Username: Login name

Password: ********

Password in CICS can be up to 8 characters long.

2. Enter a **username** in the **Username** field. A username in CICS can be up to eight characters long.
3. Enter a **password** in the **Password** field. A password in CICS can be up to eight characters long. The password is stored encrypted in the registry.

General

General settings only apply to the SNA LU 6.2 protocol. This section will explore the General settings that need to be set in the Configuration Editor for the CICS adapter. In the General Setting section, you can define the Synchronization Level and Security. From the Configuration Settings Editor dialog:

1. Expand the **General** branch.
2. Expand the **Use Global Settings** branch.

Mode Name

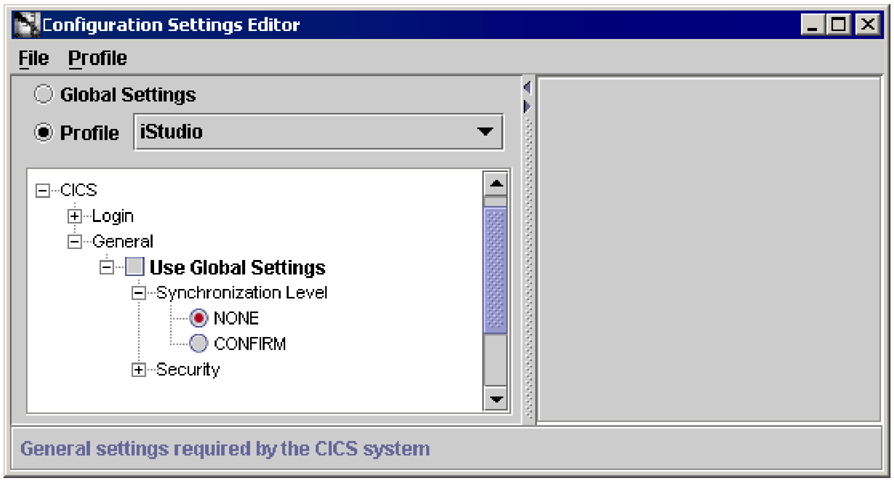
To use a different Mode Name, enter the name of the Mode Name. The Mode Name must be defined in both your local and remote system network architecture servers. If you have any questions, please refer to your communication system administrator.

Synchronization Level

In the Synchronization Level section, you can define the type of confirmation the remote CICS system requires when exchanging buffers with the CICS adapter.

1. Expand the **Synchronization Level** branch.

Figure 7–3 Configuration Settings Editor - Synchronization Level



General settings required by the CICS system

2. Click **None** if there will be no confirmation when exchanging buffers with the CICS adapter.

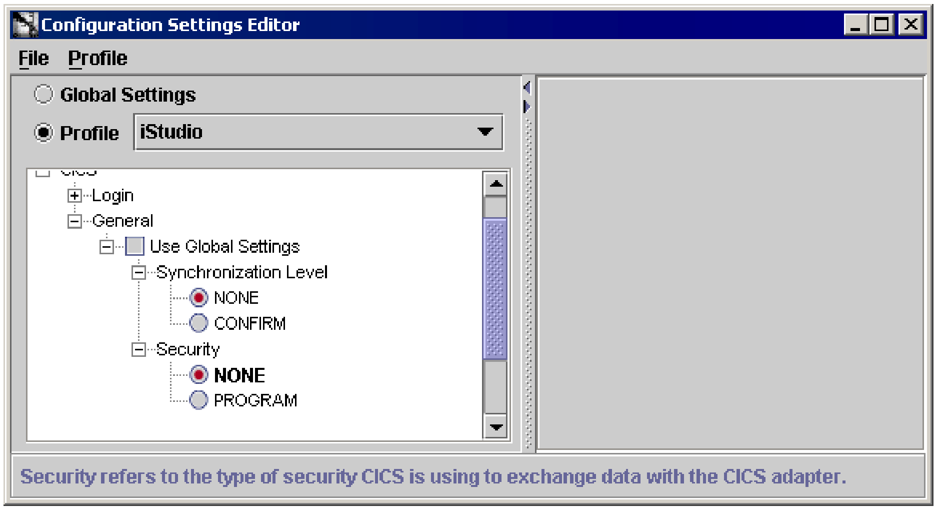
   Click **Confirm** if there will be confirmation.

Configuration Editor Security

In the Security section, you can define the type of security CICS is using to exchange data with the CICS adapter.

1. Expand the **Security** branch.

Figure 7–4 Configuration Settings Editor - Security section



Global Settings [✓ Profile] iStudio

CICS
Login
General
Use Global Settings
Synchronization Level
[✓ NONE] CONFIRM
Security
[✓ NONE] PROGRAM

Security refers to the type of security CICS is using to exchange data with the CICS adapter.

2. Click **None** if there will be no security when exchanging data with the CICS adapter.

   Click **Program** if there will be security when exchanging data.

Index