TXSeries™ for Multiplatforms

CICS Application Programming Guide

Version 5.1

SC09-4460-03

TXSeries™ for Multiplatforms

CICS Application Programming Guide

Version 5.1

SC09-4460-03

Note
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 287.

Fourth Edition (March 2004)

This edition replaces SC09-4460-02.

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

Contents

Figures

Tables

About this book

This book describes the IBM® TXSeries™ CICS® application programming interface (API). It contains information needed to prepare application programs using CICS on AIX®, HP-UX, Solaris, and Windows systems. Supported programming languages include COBOL, PL/I, C, and C++. (HP-UX does not support C++; HP-UX and Solaris do not support PL/I.)

Who should read this book

The information about developing applications is intended mainly for:

- Experienced application programmers who are relatively new to CICS.
- Experienced CICS application programmers who need to know the difference between application programming for TXSeries CICS and for other CICS products.

Systems administrators and systems analysts might also find the information in this book useful.

The information about migration is intended for those responsible for planning and implementing the migration to or from TXSeries CICS. It is not intended for those users for whom TXSeries CICS is their first CICS system.

Document organization

Table 1. Road map for the CICS Application Programming Guide book

If you want to...	Refer to...
Know more about application programming, designing efficient applications, programming constraints, and coding for the presentation, data, and business services.	Part 1, "Writing applications," on page 1.
Migrate your applications to and from CICS on Open Systems	Chapter 7, "Migrating CICS applications to and from TXSeries CICS," on page 191.
Translate, compile, and link-edit your applications	Chapter 8, "Translating, compiling, and link-editing CICS application programs," on page 213.
Code your applications for problem determination, test, and debugging	Chapter 9, "Coding for problem determination," on page 233.
Understand the commands used in application programming	"CICS commands used in application programming," on page 275.
Read about the conventions used in this book.	xii.
Read about the CICS on Open Systems library and related books.	"Bibliography" on page 285.

Chapter 1 introduces concepts discussed in subsequent chapters, which describe how to develop CICS applications contain guidance material and should be used in conjunction with the *CICS Application Programming Reference* and the *CICS Intercommunication Guide*.

This book contains information about all TXSeries CICS products. Where the information is different for a specific operating system, it is presented as follows:

- If the information is brief, the information is qualified by the product name in line with the text, usually in a list or enclosed in parenthesis. Refer to the following example:
  - /usr/lpp/cics on AIX
  - /opt/cics on HP-UX
  - /opt/cics on Solaris.
- If the information is more that a few words, but smaller than a page, it is presented as shown:

> **On CICS for AIX only**
> Before you begin using the "Hello World" transaction, make sure that CICS was installed with both the development and the production system environments.

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other TXSeries documentation, send your comments by e-mail to idrcf@hursley.ibm.com. Be sure to include the name of the book, the document number of the book, the version of TXSeries, and, if applicable, the specific location of the information you are commenting on (for example, a page number or table number).

Conventions used in this book

TXSeries documentation uses the following typographical and keying conventions.

Table 2. Conventions used in this book

Convention	Meaning
Bold	Indicates values you must use literally, such as commands, functions, and resource definition attributes and their values. When referring to graphical user interfaces (GUIs), bold also indicates menus, menu items, labels, buttons, icons, and folders.
`Monospace`	Indicates text you must enter at a command prompt. Monospace also indicates screen text and code examples.
Italics	Indicates variable values you must provide (for example, you supply the name of a file for *file_name*). Italics also indicates emphasis and the titles of books.
< >	Enclose the names of keys on the keyboard.
<**Ctrl-***x*>	Where *x* is the name of a key, indicates a control-character sequence. For example, <**Ctrl-c**> means hold down the **Ctrl** key while you press the **c** key.
<**Return**>	Refers to the key labeled with the word Return, the word Enter, or the left arrow.
`%`	Represents the UNIX® command-shell prompt for a command that does not require **root** privileges.
`#`	Represents the UNIX command-shell prompt for a command that requires **root** privileges.
`C:\>`	Represents the Windows® command prompt.
>	When used to describe a menu, shows a series of menu selections. For example, "Select **File > New**" means "From the **File** menu, select the **New** command."

Table 2. Conventions used in this book (continued)

Convention	Meaning
Entering commands	When instructed to "enter" or "issue" a command, type the command and then press <**Return**>. For example, the instruction "Enter the **ls** command" means type **ls** at a command prompt and then press <**Return**>.
[ ]	Enclose optional items in syntax descriptions.
{ }	Enclose lists from which you must choose an item in syntax descriptions.
|	Separates items in a list of choices enclosed in { } (braces) in syntax descriptions.
...	Ellipses in syntax descriptions indicate that you can repeat the preceding item one or more times. Ellipses in examples indicate that information was omitted from the example for the sake of brevity.
IN	In function descriptions, indicates parameters whose values are used to pass data to the function. These parameters are not used to return modified data to the calling routine. (Do *not* include the IN declaration in your code.)
OUT	In function descriptions, indicates parameters whose values are used to return modified data to the calling routine. These parameters are not used to pass data to the function. (Do *not* include the OUT declaration in your code.)
INOUT	In function descriptions, indicates parameters whose values are passed to the function, modified by the function, and returned to the calling routine. These parameters serve as both IN and OUT parameters. (Do *not* include the INOUT declaration in your code.)
$CICS	Indicates the full path name where the CICS® product is installed; for example, **C:\opt\TXSeries\cics** on Windows® or **/opt/cics** on Solaris. If the environment variable named CICS is set to the product path name, you can use the examples exactly as shown; otherwise, you must replace all instances of $CICS with the CICS product path name.
CICS on Open Systems	Refers collectively to the CICS product for all supported UNIX platforms.
TXSeries® CICS	Refers collectively to the CICS for AIX®, CICS for HP-UX, CICS for Solaris, and CICS for Windows products.
CICS	Refers generically to the CICS on Open Systems and CICS for Windows products. References to a specific version of a CICS on Open Systems product are used to highlight differences between CICS on Open Systems products. Other CICS products in the CICS Family are distinguished by their operating system (for example, CICS for OS/2® or IBM® mainframe-based CICS for the ESA, MVS™, and VSE platforms).

Part 1. Writing applications

Table 3. Road map for Writing applications

If you want to...	Refer to...
Look up transaction processing, look up the application programming interface, see the application development tools, view a sample transaction.	Chapter 1, “Introduction to CICS application programming,” on page 3
Design efficient applications that store data within and across transactions, know about data operations, Look up CICS locking and updating files from non-CICS applications.	Chapter 2, “CICS application design considerations,” on page 21
Look up programming constraints caused by COBOL, C and C++ compilers.	Chapter 3, “Programming constraints,” on page 49
Look up coding for presentation services.	Chapter 4, “Coding for presentation services,” on page 67
Look up coding for data services.	Chapter 5, “Coding for data services,” on page 101
Look up coding for business logic parts of the application.	Chapter 6, “Coding for business logic,” on page 173.

Chapter 1. Introduction to CICS application programming

This chapter provides you with an introduction to application programming concepts used in the CICS application programming interface. The topics discussed are:

- Transaction processing with CICS from a CICS family perspective
- How CICS runs your transactions
- The CICS application programming interface (API)
- The CICS application development tools

Why use CICS?

Online transaction-processing systems (OLTP) can provide accurate, up-to-date information within seconds, from terminals that can give direct access to data held either as files or databases. Developing such a system would be a major undertaking, particularly if you had to write all your own control programs for handling terminals, files, and databases, and provide your own transaction processing mechanisms. However, CICS supplies all the transaction processing and resource management functions, allowing you to concentrate on developing application programs to meet your organization's business needs.

What does CICS do for you?

CICS controls OLTP application programs in a distributed transaction processing (DTP) environment. CICS handles interactions between terminal users and your application programs. Your programs gain access to the CICS facilities with straightforward, high-level commands.

CICS provides:

- Communication functions to terminals and systems required by application programs
- Control of concurrently running programs serving online users
- Facilities for accessing databases and files
- The ability to communicate with other CICS family members using Systems Network Architecture (SNA) and Transmission Control Protocol/Internet Protocol(TCP/IP)
- Interactive facilities to configure your system
- Recovery processing and data protection if a problem occurs

CICS transaction processing

This section describes general concepts used in CICS application programming. The CICS family of products is described, as well as transaction processing, distributed and cooperative transaction processing, and LAN-based client/server transaction processing.

The CICS family

The *Customer Information Control System (CICS)* is a general-purpose data communication and online transaction-processing system that is capable of supporting a network of many thousands of terminals. It acts as a specialized operating system that provides the environment for the execution of online

application programs, including interfaces to files and database products. The CICS family has a long history and wide acceptance in transaction processing. The key features of the CICS family are the common application programming interface and easy-to-use communications. These allow significant source compatibility and access to data and services across CICS systems. The members of the CICS family satisfy these requirements in various ways.

The CICS family members are often grouped when describing common characteristics. Those groups are:

- ***IBM CICS for Windows***
- ***CICS on Open Systems***
  - CICS for AIX offers CICS capabilities for transaction processing on the IBM RISC System/6000, the IBM workstation for scientific and commercial applications.
  - IBM CICS for HP-UX offers CICS capabilities for transaction processing on Hewlett-Packard's HP 9000 Series 800 Corporate Business Servers.
  - CICS for Solaris offers CICS capabilities for transaction processing on the SUN Solaris Operating Environment.
- ***IBM mainframe-based CICS***
  - CICS for MVS/ESA for the MVS/ESA environment provides transaction processing for large mainframe centers and for large distributed sites.
  - CICS/MVS for the MVS/XA and MVS/ESA environments also provides transaction processing for the large mainframe center.
  - CICS/VSE for the VSE/ESA environment provides transaction processing for the intermediate central site as well as distributed user locations requiring volume transaction processing.
  - CICS/DOS/VS for the VSE/SP environment provides transaction processing for small- to medium-sized businesses.

The following CICS products are also available:

- CICS/400 for the OS/400 environment provides CICS capabilities for online transaction processing on the AS/400, the IBM midrange processor for business and commercial applications.
- CICS OS/2 provides CICS capabilities for transaction processing on PCs running OS/2. It also provides transaction processing for the distributed location where the PC can access data locally or at the mainframe.

See *CICS Family: API Structure* and *CICS Family: Interproduct Communication* for related information.

Transaction processing terms and concepts

A *transaction* is a unit of processing that consists of one or more *application programs*. A transaction is initiated by a single request, often from a terminal, usually using a four-character transaction identifier. A transaction may require the initiation of one or more tasks for its execution.

A *task* is a single instance of the execution of a transaction. CICS, in conjunction with the operating system, manages the control of multiple tasks by allocating a system process to each task. Then, while one task is waiting (for example, to read a file or to get a response from a terminal), the operating system can give control to another task.

The processing done by the transaction processing system can be grouped into *logical units of work (LUWs)*. Each LUW is a set of related changes to data. For example, in an accounting system, one LUW comprises updating accounts payable, updating the books, and creating a check. The work performed by each LUW is completely separate from the work performed by any other LUW. If the LUW changes multiple resources they are either all changed successfully or none of them are. The transaction processing system ensures that, when there are multiple users accessing the resources, the partial changes made in one LUW are not made available to other users until the LUW is completed. Once an LUW has been completed, its changes are permanent.

For more information on transaction processing, see *Concepts and Planning*.

Distributed transaction processing

CICS provides a large set of intercommunication services between its family members. These services support:

- *Function shipping*, which enables application programs to access resources in another CICS system.
- *Distributed transaction processing*, which enables transactions running in one CICS system to initiate and communicate synchronously with transactions in another CICS system.
- *Distributed program link* (DPL), which allow CICS transactions to link to programs on any connected CICS system. For example, CICS applications can have access to programs that access databases on a host system.
- *Asynchronous transaction processing*, which allows a CICS transaction to initiate an independent transaction in a connected system and to pass data to it.
- *Transaction routing*, which allows operators of terminals owned by one CICS system to run transactions in any connected CICS system.
- *Advanced program-to-program communications* (APPC), which enables CICS programs to send and receive data from a program running in a remote system. Programming using APPC conversations is called Distributed Transaction Programming (DTP).

See the *CICS Intercommunication Guide* for information about writing application programs in a CICS intercommunication environment. See the *CICS Family: API Structure* and the *CICS Family: Interproduct Communication* for related information.

Developing applications within CICS

CICS is a general purpose online transaction processing (OLTP) system that provides an environment for running online transactions. You write a CICS program in much the same way as you write any other program. Most of the processing logic is expressed in standard language statements, but you use CICS commands for some functions. These commands are typically embedded in your application, preceded by the phrase **EXEC CICS**. For example:

```
EXEC CICS SET FILE(myfile) OPEN
```

The program source file is then translated (using the **cicstran** program) prior to being compiled, though translation and compilation can be done in one step with the **cicstcl** command. See Chapter 8, “Translating, compiling, and link-editing CICS application programs,” on page 213 for more information.

CICS on Open Systems and CICS for Windows support various compilers and languages.

CICS for AIX supports the following:

- IBM C and C++ for AIX
- IBM VisualAge C++
- IBM COBOL Set for AIX
- Micro Focus Server Express COBOL
- IBM PL/I
- Java Development Kit (JDK)

CICS for Solaris supports the following:

- Sun Workshop Compilers C/C++
- Micro Focus Server Express COBOL

CICS for HP-UX supports the following:

- HP-UX C/ANSI C
- Micro Focus Server Express COBOL

CICS for Windows supports the following:

- IBM Visual Age C++ for Windows
- Microsoft Visual C++
- IBM VisualAge COBOL for Windows
- Micro Focus Net Express COBOL
- IBM VisualAge PL/I Enterprise

See the *TXSeries Release Notes* for the specific versions of this software supported for the current release.

How a CICS-based application differs from a batch application

Recovery processing for an online application is more complex than for a batch system. In a batch application, input data is prepared before processing begins. The data is then supplied to the batch process in an orderly sequence, which is controlled and predictable. With a batch program, you can repeat the processing, or continue it from the point of failure. In a CICS application, input data is not prepared. The application user enters the data as needed while the application is running, and the data arrives in an unpredictable sequence. With an online application, you cannot simply rerun the application or continue from the point of failure because the state of the process is unknown. Online application programs require mechanisms to ensure that each resource associated with an interrupted online application is returned to a known state, so that processing can be restarted safely.

Application program development life cycle

Application program design is an iterative process. Decisions about the user interface affect transaction definitions, which in turn, cause a slight change in specifications, and the whole cycle begins again. A simple development life cycle can include steps to:

1. **Define the problem**: Specify broadly, the function required. This will come from the user departments within your organization.
2. **Design the transactions**: You need to define transactions and logical units of work (LUWs) to perform the defined functions. You must consider several points when designing these transactions:
   - The requirements imposed by the environment. This includes terminal type and size, data storage format, and security requirements.

- The machine usage, response times and transaction availability.
- The type, size, and quantity of records to be processed.
- The IBM 3270 Information Display System that CICS emulates. This is how your applications communicate with the application user.
- The efficiency and usability of the application. These are major goals in any application.
- The screen layout of your user interface, the consistency between screens, the number of expected keystrokes, and the number of confirmation messages issued to reassure users.
- The difference in constraints between online processing and batch applications.
- The exceptional conditions that come with new considerations for an online environment.
- The level of programming specifications and the availability of up-to-date system information.

3. **Write the application program**: Considerations include:
   - Choosing a programming language
   - Deciding how your application program uses the BMS screens
   - Defining screens with Basic Mapping Support (BMS)
   - Saving data and communicating between transactions
4. **Translate and compile the application program.**
5. **Test the application program.**

Developing client/server applications

In TXSeries CICS there are two application programming interfaces that allow non-CICS applications in a client system to use the facilities of CICS in a connected server system. Those interfaces are:

- The external call interface (ECI)
- The external presentation interface (EPI)

The external call interface (ECI)

With the ECI, you can write applications that:

- Call a CICS program in a CICS server from a non-CICS program
- Connect to several servers at the same time
- Have several outstanding program calls at the same time
- Access CICS programs, files, transient data queues, temporary storage, and transactions
- Exchange data between the client and the server

The ECI application programs make synchronous or asynchronous calls. Synchronous calls return control when the called program completes; the information returned is immediately available. Asynchronous calls return control without reference to the completion of the called program, and the application is notified when the information becomes available.

Calls may be extended, that is a single logical unit of work may cover more than one successive call, though only one call can be active for each logical unit of work at a time. The application can manage multiple logical units of work concurrently if it uses asynchronous calls.

The called program can:

- Update resources on its own system.

- Use distributed program link (DPL) to call CICS programs on other systems.
- Access resources on other CICS systems with function shipping or distributed transaction processing (DTP).

The ECI consists of three types of calls:

1. Program link calls that cause a CICS program to be executed on a CICS server.
2. Status information calls that retrieve status information about the application and its connection to the CICS server.
3. Reply solicitation calls that retrieve information after asynchronous program link or asynchronous status information calls.

Also available with the ECI is the ability to retrieve information about available servers to which the calls are directed.

The external presentation interface (EPI)

With the EPI, you can write applications that:

- Allow a non-CICS application program to be viewed as a 3270 terminal by a CICS server system to which it is connected.
- Connect to several servers at the same time.
- Have several outstanding program calls at the same time.
- Schedule transactions, where the application acts as the principal facility.

In CICS servers that support access through the EPI, other CICS transactions running in the server can use the START command to schedule transactions that will use the non-CICS application as their principal facility. When a transaction is initiated, 3270 datastreams and events are passed between the CICS server and the application. The application can present the contents of the terminal I/O to its user in any manner appropriate to the application's operating environment.

Transactions may be routed to other CICS systems by standard transaction routing. Resources on other CICS systems can be accessed with function shipping.

The EPI consists of functions, data structures, and events. The EPI functions define application programming calls, such as installing new terminals to be controlled by the process, sending data from a terminal, and terminating the process. The EPI data structures define EPI data, such as reason codes, details of events, terminal details, and sense codes. The EPI events are used to respond to events that occur against a terminal, such as when a transaction sends data and is expecting a reply.

Writing ECI and EPI application programs

ECI and EPI application programs that run on CICS on Open Systems clients may be written in COBOL, C, C++, or PL/I. Programs that do not make operating system specific calls are portable between CICS on Open Systems clients and other IBM CICS Universal Client products. Application programs can use the facilities of both the ECI and the EPI.

See the *CICS Family: Client/Server Programming* for related information.

How CICS runs your transactions

This section describes how CICS transactions and application programs are run.

Note: The term *program* can have different meanings in different contexts. Therefore, the terms *CICS program*, *application program*, *executable*, *COBOL program*, and *loadable object* are used when the use of *program* is otherwise misleading or unclear.

Components of the CICS runtime system

The major components of TXSeries CICS are:

- The IBM CICS *clients* that you use to attach to a CICS region and through which you run CICS transactions.
- The *transaction scheduler*, a component that receives requests to run transactions, prioritizes and schedules them, and dispatches them to an application server for processing.
- A pool of *application servers* that execute the transactions and interact with the CICS on Open Systems client processes to send and receive terminal input and output.
- One or more *Structured File Servers (SFS)* that house the CICS files, intrapartition transient data queues, auxiliary temporary storage queues belonging to your region, and asynchronous processing local queues.
- A *PPC Gateway server* used for intersystem communication across SNA networks. A PPC Gateway server is only required for synchronization level 2 communications across SNA.

Interacting with the CICS region

On CICS for Windows you interact with a CICS region through the CICS on Open Systems clients or by using Telnet in conjunction with the **cicsteld** process.

On CICS on Open Systems you interact with a CICS region by running either the **cicsterm** or **cicsteld** process or a replaceable CICS client process on your local terminal. **cicstermp** runs on a printer. These are all multi-threaded CICS processes.

Your local terminal does not have to be located on the same node as the region to which you wish to be attached, and it is not necessary to have all your CICS on Open Systems client processes on the same nodes. Because of this flexibility, CICS communicates between a CICS client and the region by using remote procedure calls (RPCs). If your terminal is remote to the node housing the region, network traffic is involved in sending the RPC to and from your CICS on Open Systems client process. If it is local, no network traffic is involved, but a communication path still has to be set up between the CICS on Open Systems client process and the region. The CICS on Open Systems client processes communicate using RPCs both with the transaction scheduler and the application server processing the current transaction.

See the *CICS Administration Reference* and the *CICS Administration Guide* for related information.

Transaction scheduler

This CICS component is responsible for scheduling and dispatching the transaction to be run. Each user, device and transaction has an associated priority; when you submit a transaction the scheduler computes an overall priority for it which it uses to prioritize requests in times of heavy demand. When able to do so, it dispatches the transaction request to an available application server for processing.

The scheduler communicates with the application server and the CICS on Open Systems client. The scheduler, as the controller of the overall workload, also controls the number of application servers in the region.

Application servers

An application server is a multi-threaded CICS process providing a complete environment for running a CICS transaction. If your transaction is conversational, it will be processed by exactly one application server, but if it is pseudoconversational each transaction in the sequence may in turn be executed by a different server.

When you configure CICS, you specify the minimum and maximum number of servers for your region. The transaction scheduler ensures that this minimum number is always present, and creates and destroys servers up to the defined maximum number depending on the overall workload.

All the application servers in the pool have to be located on the same node, but do not need to be on the same node as the CICS on Open Systems client. They communicate with each other using signals and shared memory, and use RPCs to communicate with the other components of the region.

Structured File Server (SFS)

Your region will use one or more SFSs, or optionally a relational database supported by your system, such as DB2 on AIX.. CICS provides several commands to allow you to access SFS files. In addition, intrapartition transient data queues, auxiliary temporary storage queues, and asynchronous processing local queues store their data in SFS files. All access to SFSs occurs through application servers; the application server, a server in its own right to CICS, becomes a client of SFS. There is no need to locate all the SFSs on the same node, or on the same node as the application servers. Communication is with RPCs.

PPC Gateway server

A region can use one or more PPC Gateway server servers to access SNA hosts. As with SFS, an application server becomes a client of the PPC Gateway server. See the *CICS Intercommunication Guide* for more information about using the PPC Gateway server to access SNA hosts.

How CICS executes your transactions

There are two steps in this process:

1. Requesting a transaction to be run
2. Executing the transaction

Requesting a transaction to be run

When you enter a CICS transaction identifier through an CICS on Open Systems client, the transaction scheduler selects an application server to run the transaction.

Executing the transaction

When you link-edit a CICS program to be run under CICS (for example using **cicstcl** to translate, compile and link), the object you create is not directly executable by the operating system both because it contains some unresolved symbols and because it expects to be run by a CICS application server. The CICS application server provides a complete environment for running the loadable objects produced by **cicstcl**. To run your transaction, CICS looks up the name and location of its first CICS program, and uses the operating system dynamic loading facility to load that program into the application server. The unresolved symbols in the program are then resolved by symbols provided by the server and the program

begins execution. This applies irrespective of the language the program is written in. Programs produced by the supported compilers are shared automatically between multiple application servers on the same machine. A copy of the program is loaded by each application server that is requested to run it.

The CICS application programming interface (API)

CICS offers a common set of programming commands that are used to request CICS services from an application program. This set of commands is referred to as the *application programming interface* (*API*). Because the API is common to all CICS family members, CICS applications can be moved from one platform to another.

The commands are statements that you include at appropriate points in your application program to perform a variety of programming functions. If you are familiar with CICS mainframe products, you should note that TXSeries CICS support command-level, but not macro-level, application programs.

You can find full reference information for each of the CICS commands available in the *CICS Application Programming Reference*. Refer also to Chapter 7, "Migrating CICS applications to and from TXSeries CICS," on page 191 for information about preparing applications developed on other platforms to run with either CICS on Open Systems or CICS for Windows.

Note: TXSeries CICS support a subset of the CICS API commands. When migrating applications, it is important to note that some commands are not available with all CICS products. Refer to the application programming information for each CICS product as well as *CICS Family: API Structure* for details.

How to split the program logic

The basis of application design is the split of program logic into modules, in much the same way as for traditional modular and structured programming. In general terms, there are three parts to an application:

- Presentation services
- Data services
- Business logic
- Problem determination logic

Presentation services

Presentation services are used for communication between the end user and the transaction processing system. Presentation services interface with the presentation management facilities of the system, which may or may not be part of CICS.

See "Summary of API presentation services" on page 12 and Chapter 4, "Coding for presentation services," on page 67 for information about the CICS commands and facilities provided for presentation services.

Data services

Data services are used to retrieve and update data. Data services interface with the CICS data management facilities.

See "Summary of API data services" on page 13 and Chapter 5, "Coding for data services," on page 101 for information about the CICS commands and facilities provided for data services.

Business logic

Business logic, which forms the bulk of the processing, performs the data

manipulation and computation required by the transaction. Subdivide your business logic in such a way that each module provides a separate service. For example, you could have modules for:

- Checking the validity of your input data
- Handling communications
- Performing data access
- Accessing system information
- Setting up your processing environment
- Requesting system services

This technique of dividing up your business logic is known as *isolation*. Designing your applications in this way can give you a number of benefits, such as:

- Enhanced portability for distribution of applications across CICS platforms.
- Enhanced programmer productivity because code can be reused in other applications.
- Reduced maintenance costs because similar functions are grouped together, making it easier to locate and modify code.
- Well-defined interfaces, making it easier to add new modules or to replace outdated ones.

See "Summary of API business logic" on page 14 and Chapter 6, "Coding for business logic," on page 173 for information about the CICS commands and services provided for business logic.

Refer also to "Using CICS-supplied transactions to test your application" on page 255.

Problem determination logic

Problem determination logic is used to handle error situations and to aid in the development and debugging of applications.

See "Summary of API problem determination logic" on page 15 and Chapter 9, "Coding for problem determination," on page 233 for information about the CICS commands and services provided for problem determination logic.

For information about application programming for distributed transaction processes, refer to the *CICS Intercommunication Guide*.

Summary of API presentation services

There are two ways to write applications that interact with terminals in CICS:

- Basic Mapping Support (BMS)
- Terminal Services

Using Basic Mapping Support (BMS) in CICS

BMS provides both device and format independence for display terminals and printers. *Device independence* means that you do not need to know the control characteristics of the terminal. *Format independence* simplifies the positioning of data on the terminal, and allows CICS to adapt displays for different terminals without any change to the application program.

If you are familiar with CICS on other platforms you will be aware that BMS provides three levels of support, called *minimum*, *standard*, and *full* function. CICS on Open Systems and IBM CICS for Windows support minimum function BMS

and some standard function (see “BMS functions supported in CICS” on page 72). This allows you to code source maps that describe the prompts and inputs on your screen, which you process using the BMS processor. You then code CICS commands in your application to send and receive these maps to and from the terminal. The commands you can use for this purpose are:

- SEND TEXT
- SEND CONTROL
- SEND MAP
- HANDLE AID
- RECEIVE MAP

See “Basic mapping support (BMS) services” on page 71 for information on these commands.

Using terminal services in CICS

Terminal services allow you to connect a wide variety of terminals to the CICS system. The terminal services commands are:

- CONVERSE
- RECEIVE
- SEND
- SEND TEXT
- WAIT TERMINAL

These commands use ASCII 3270 datastreams to communicate with the terminal.

These commands are discussed in “Terminal services” on page 67.

Note: Advanced use of these commands requires a knowledge of 3270 data streams and the various capabilities of terminals.

Summary of API data services

Data services refer to the storage of data in files, queues, journals and databases and the ways in which you can retrieve the data. Data services include the following facilities.

File services

CICS allows you to access user files managed by an Encina Structured File Server (SFS) as *VSAM* files. VSAM refers to the *Virtual Storage Access Method* that provides direct or sequential processing of fixed- and variable-length records on direct access devices.

For more information about how CICS accesses SFS files as VSAM files, see “VSAM emulation by SFS and distributed CICS” on page 108 and for more information about how CICS accesses DB2 files as VSAM files, see “VSAM emulation by DB2 and distributed CICS” on page 113.

SFS files support keyed, sequential, and relative access. Files can be defined as recoverable or non-recoverable. CICS cooperates with SFS to provide the transactional properties for accessing recoverable files.

In addition to the usual file facilities that allow you to read from, write to, and delete a file, CICS provides a file browse function. You select the record at which you want the browse to start. You can then read each record in the file in turn, either forwards or backwards through the file.

Queue services

CICS provides some special storage areas of its own, called transient data queues and temporary storage queues. The *transient data* service handles queues of data to be sent to terminals, such as printers, and to sequential files. The *temporary storage* service provides an internal scratchpad. Both of these storage types can be made recoverable.

Journal services

CICS provides facilities for creating and managing journals. A journal is a set of special-purpose sequential files and may be used, for example, to keep an audit trail or system log.

See also the *CICS Application Programming Reference*.

Relational database services

CICS allows access to relational databases that provide a programmable interface through Structured Query Language (SQL) commands in COBOL, C, C++ or PL/I. See "SQL restrictions and relational database services" on page 142 for information about usage and restrictions.

Summary of API business logic

Business logic refers to the manipulation of data from the time it is retrieved from storage to the time when it is either presented to the user or updated. The following sections list the commands used for business logic.

Program execution services

A transaction is not limited to running a single program. You can get one program to call another, and you can return a request to run another transaction when this transaction completes. The commands are:

- LINK
- RETURN
- XCTL

These commands are discussed in "Program execution services" on page 174.

Timer services

Timer services are provided that enable you to start and control transactions. The commands are:

- ASKTIME
- CANCEL
- DELAY
- FORMATTIME
- RETRIEVE
- START
- SUSPEND

These commands are discussed in "Timer services" on page 179.

Synchronization services

Synchronization services enable serialized access to resources. The commands are:

- DEQ
- ENQ

These commands are discussed in "Synchronization services" on page 181.

Storage services

Storage areas and commands that manage task storage are provided. The commands are:

- ADDRESS
- FREEMAIN
- GETMAIN
- LOAD
- RELEASE

These commands are discussed in "Storage services" on page 182.

Logical unit of work (LUW) services

Commands are available to delimit logical units of work in your transaction. The commands used for this purpose are:

- RETURN
- SYNCPOINT

These commands are discussed in "Logical unit of work (LUW) services" on page 184.

Configuration services

Commands are available to enquire upon and dynamically configure CICS runtime resource definitions, such as files and transactions. The commands you can use for this purpose are:

- ASSIGN
- INQUIRE and SET

These commands are discussed in "Configuration services" on page 186.

Refer to the *CICS Administration Reference* and the *CICS Application Programming Reference* for additional guidance information on the use of INQUIRE and SET.

Intersystem communication services

As described in "Distributed transaction processing" on page 5, CICS provides commands that enable distributed transaction processing (DTP) between a CICS region and any system supporting APPC protocol. The commands are:

- ALLOCATE
- CONNECT PROCESS
- CONVERSE
- EXTRACT ATTRIBUTES
- EXTRACT PROCESS
- FREE
- ISSUE ABEND
- ISSUE CONFIRMATION
- ISSUE ERROR
- ISSUE PREPARE
- ISSUE SIGNAL
- RECEIVE
- SEND
- WAIT CONVID

See also the *CICS Intercommunication Guide*.

Summary of API problem determination logic

Problem determination logic refers to the services provided to aid in error handling and application programming debugging.

Error handling

As described in "Error-handling services" on page 233, a number of commands allow you to handle exception conditions that might arise during the running of your applications. Those commands are:

- ABEND
- HANDLE ABEND
- HANDLE CONDITION
- IGNORE CONDITION
- POP HANDLE
- PUSH HANDLE

Debugging services

As described in Chapter 9, "Coding for problem determination," on page 233, commands are available to help you debug application programs. The commands are:

- TRACE
- ENTER
- DUMP

Performance monitoring

As described in "Performance monitoring services" on page 251, commands are available to help you analyze the performance of your system and of individual transactions, thereby helping to determine problems. The commands used for this are:

- COLLECT STATISTICS
- ENTER
- INQUIRE STATISTICS
- PERFORM STATISTICS RECORD
- SET STATISTICS

CICS application development tools

This information introduces you to the tools provided in CICS that enable you to develop and debug transactions.

Presentation interface development

Use the CICS BMS processor to translate BMS source files, which contain the definitions of map sets, to produce a symbolic map and a physical map. The *symbolic map* is a programming source language data structure (a COBOL, C, C++, or PL/I structure) used by the compiler to resolve source language references to fields in the map. The *physical map* contains the information necessary to display the map on a physical terminal, and contains instructions for embedding control characters within a datastream in order to achieve this.

Application program translation

Application programs that include CICS API commands are processed by the *command language translator (cicstran)* that translates the CICS API commands into statements in the language used. This translator accepts as input a source program written in COBOL, C, or C++, where CICS API commands are coded, and produces as output an equivalent source program where each command is translated into statements in the language of the source program. You can then compile and link-edit your programs using the COBOL, C, or C++ compilers.

Alternatively, you can request that your source program is translated, compiled, and link-edited in one step **cicstcl**. The advantage of using this alternative is that CICS uses the correct compilers and sets up the options required by CICS for translation.

The IBM PL/I compiler has an integrated CICS processor. This means that it is unnecessary (and indeed not possible) to run a separate CICS translator step for CICS programs written in IBM PL/I. You should therefore only use either **cicstcl** or the IBM PL/I invoked directly and not attempt to use **cicstran**.

Chapter 8, "Translating, compiling, and link-editing CICS application programs," on page 213 provides full details.

Application program debugging

CICS provides a transaction called the Execution Diagnostic Facility (CEDF) that enables you to debug an application program that has been preprocessed with the **-e** option (on cicstran or cicstcl) without modifying the program. The facility displays the state of the application program at the CICS interception points and allows you to interact with the debugging tool before returning control to the application code. See "Using Execution Diagnostic Facility (CEDF)" on page 257 for additional information.

The Animator tool enables you to test a Micro Focus Server Express COBOL or Micro Focus Net Express COBOL application program online without modifying the program. This tool intercepts execution of the application program at various points before displaying information about the program. Any screens sent by the application program are displayed by the tools, so that you can converse with the application program during testing just as you would on the production system. See "Using a compiler's integrated debugging tool to debug CICS applications" on page 263.

You can also debug C, C++, and IBM COBOL programs by using debugging services provided by the relevant compiler. See "Using debugging tools integrated with compilers running on CICS for Windows" on page 263.

CICS also provides supplied transactions CECI and CECS for the interpretive execution and syntax checking of commands. See "Using Command Level Interpreter (CECI) and Syntax Checker (CECS)" on page 256.

CICS for AIX only

The IBM Application Debugging Program provides the ability to debug IBM COBOL, C, C++, or PL/I programs.

There are a number of facilities to go with the debugging tool:

A CICS-supplied transaction, CDCN, that turns the debugging tool on and off. See also the *CICS Administration Reference*.

A region-wide attribute, AllowDebugging with settings **yes** and **no**, to control whether the debugging tool can be used within the region.

Two Transaction Definitions (TD) entries:
- DFHCDCN0 (for the CICS-supplied program)
- DFHCDCN (for the CICS-supplied mapset)

The **cicstcl -a** flag. See "cicstcl - translate, compile, and link" on page 281.

Using transactions to call your program

In CICS, the term *transaction* is used to describe a fundamental unit of work, consisting of one or more application programs, that is initiated by a single request. A transaction is identified by a *transaction identifier* (*tranid*) and this transaction identifier is used to call your application program in the CICS runtime environment.

Because both the transaction and program are defined to CICS (see the *CICS Administration Reference*), you can use CICS on Open Systems clients, the **cicsterm** command (CICS on Open Systems), or the **cicsteld** command to run the transaction and, therefore, run your application program. The transaction is executed under the control of CICS, which provides the services requested by each API command.

The **cicsterm** and **cicsteld** commands are described in the *CICS Administration Reference*.

Note: This is a high-level description of how to call your program from a transaction. For more detailed information, see "How CICS runs your transactions" on page 8.

Summary of commands used in application development

As an application programmer, you use the following commands:

cicsmap
(Generate map). This command takes raw Basic Mapping Support (BMS) macros and generates either logical maps, or physical maps, as specified by the map input. See "Basic mapping support (BMS) services" on page 71.

cicstran
(Translate application source program). The input to this command language translator is the COBOL, C, or C++ source code that you have written that contains embedded CICS API commands. The output is program source in which each CICS API command has been translated into statements in the supporting language. The translator ignores any embedded Structured Query Language (SQL) statements within the source program.

Notes:

1. As the *Data Language 1 (DL/I)* database access language is not supported by CICS on Open Systems or IBM CICS for Windows, DL/I commands encountered by the translator are ignored.
2. **cicstran** cannot be used to translate IBM PL/I source code.

cicstcl This command performs the translation of the application program (as performed by the **cicstran** command), and also compiles and links the generated program source.

Note: Compiling and link-editing CICS applications requires using thread-safe compilers. See "Requirements for compiling CICS application programs" on page 218 and "Thread safety" on page 50.

Summary of CICS-supplied transactions used in application development

As an application programmer, you use the following CICS-supplied transactions. For more information on any of these transactions, see *CICS Application Programming Reference*.

Data Conversion (CALF)
This transaction is used to convert data structured in the *Virtual Storage Access Method (VSAM)* format to the file format used in CICS on Open Systems or CICS for Windows.

Note: VSAM is an access method for indexed or sequential processing of fixed- and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by means of a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry sequence), or by means of a relative-record number. See "VSAM emulation by SFS and distributed CICS" on page 108 for more information.

CICS for AIX only

CICS Transaction Diagnosis Configuration (CDCN)
This transaction turns the IBM Application Debugging Program on and off.

CICS on open systems only

CICS Animator Debug Configuration Transaction (CADB)
This transaction allows users to configure CICS to enable debugging of Micro Focus Server Express COBOL application programs with Animator.

Temporary Storage Browse (CEBR)
This transaction is used to browse temporary storage queues and transient data queues. CEBR may be useful for debugging.

For more information, see "Using Temporary Storage Browse (CEBR)" on page 255.

Command Level Interpreter (CECI) and Command Level Syntax Checker (CECS)
CECI allows you to check the syntax of, interpret, and run CICS API commands. CECS transaction allows you to check the syntax of CICS API commands.

Execution Diagnostic Facility (CEDF)
The CEDF transaction allows you to use the Execution Diagnostic Facility (EDF) that enables you to debug a CICS application program.

Runtime Resources Management Facility (CEMT)
This transaction allows you to inquire about CICS resources in the runtime environment and to dynamically change their control parameters. "Using the API for trace services" on page 247 and "Dump" on page 250 describe how you can use CEMT for testing and debugging your application program. "Performance monitoring services" on page 251 describes how you can use CEMT to gather statistics.

A sample transaction

This information shows you how to write a simple CICS transaction that sends a message to your terminal. Use this exercise to perform a simple test of your system and to become familiar with some of the CICS application programming concepts.

On CICS for Windows, you can also refer to the Installation Verification Programs (IVP), which can be found in opt\cics\samples\IVP. The IVP programs are documented in *Planning and Installation Guide*.

Prerequisites for the "Hello World" transaction

Before you begin, make sure that:

- CICS is installed with both the development and the production system environments.
- You either have CICS administrator authority for the region you are working with or you are able to ask someone with this authority to add the necessary resource definitions.
- You have a DCE account. The DCE administrator can set this up for you.

To create a "Hello World" application

1. Set your PATH environment variable to the value shown in the *CICS Administration Reference*.
2. Use an editor to create the following C source file called **WORLDPRG.ccs**:

```
int main()
{
char Hello[] = "Hello world!";
EXEC CICS SEND FROM(Hello) LENGTH(12) ERASE;
EXEC CICS RETURN;
}
```

3. Run **cicstcl -lC WORLDPRG.ccs** to create an executable called "WORLDPRG" on CICS on Open Systems and "WORLDPRG.dll" on CICS for Windows.
4. Have someone with CICS administrator authority:
   a. Add a definition of the program WORLDPRG to the same running region you are allowed access to.
   b. Add a definition of a transaction called "HELO" to run this program on the same running region.

      Note: It is recommended that you have these definitions deleted when they are no longer required.

To run the "Hello World" transaction

1. Check that the PATH environment variable is set to the value shown in the *CICS Administration Reference*.
2. You are now in a position to run the transaction you have created. To do this, log on to DCE by using the dce_login utility.
3. To access CICS facilities, you need to run an CICS on Open Systems client. For example, to get this access on CICS on Open Systems, enter **cicsterm** on the command-line.
4. Now enter HELO as the transaction id. If all worked correctly, "HELLO WORLD" is displayed on your terminal.
5. It is good practice to press CLEAR before entering the next transaction.

Chapter 2. CICS application design considerations

This chapter discusses how to design CICS applications to maximize their performance and efficiency. CICS application design entails consideration of several areas including:

- CICS transaction design modes
- CICS program design modes
- Transaction data storage options
- Data management storage options
- CICS environment factors
- CICS locking function behavior
- Performance considerations for CICS developers
- Using CICS with WebSphere MQ

CICS transaction design efficiency considerations

A primary efficiency consideration for online transactions is whether to design transactions to occur in nonconversational, conversational or pseudoconversational mode.

A *nonconversational* transaction has only one input—the one that causes the transaction to be invoked. The transaction processes that input, displays a response to the screen, and terminates. Nonconversational transactions use system resources for only short time periods.

A *conversational transaction* involves more than one input from the terminal, so that the transaction and the user enter into a kind of conversation. In a conversational transaction, processor utilization times, even including the time for accessing files, are considerably shorter than the time required to transmit the input from the terminal to the processor (terminal transmission time). Furthermore, terminal transmission times are considerably shorter than user response times. Consequently, conversational transactions use storage and other resources for much longer time periods than nonconversational transactions.

A *pseudoconversational* transaction is one in which a series of nonconversational transactions are embedded in a single sequence. This sequence looks to the user like a single conversational transaction involving several screens of input. Each transaction in the sequence handles one input, sends back a response, and terminates. As a result, pseudoconversational programming uses storage and other resources for shorter periods of time compared to conversational transactions.

Before a pseudoconversational transaction terminates, it can pass data forward for use by the next transaction that is initiated from the same terminal, whenever that transaction arrives. By using the TRANSID option of the RETURN command, a pseudoconversational transaction can specify what the next transaction is to be. However, be aware that if another transaction is started for that device, it might interrupt the pseudoconversational chain that you have designed, unless you specify the IMMEDIATE option on the RETURN command. In this case, the transaction that is specified by the TRANSID command is attached, regardless of any other transactions that are queued for this terminal.

Nonconversational transactions are embedded in conversational and pseudoconversational transaction modes; therefore this discussion will focus on the two latter design modes. This section discusses the impact of a choice of conversational or pseudoconversational transaction design mode on the following areas:

- Contention resources
  - Storage use
  - Processor use
- Exclusive use resources
  - Response time constraints
  - Recovery and integrity constraints
  - Order completeness constraints
  - Uninterrupted transaction constraints
- Performance profile issues
  - Performance costs
    - Consumption of processing power
    - Throughput, response time, and predictability
    - Availability of application servers
  - Performance benefits of operational control
- Program-terminal communications
  - Screen integrity
  - Warning capability

CICS program design efficiency considerations

Efficient CICS program design includes incorporating use of the following operating system and language facilities:

- Shareable, loadable, and executable operating system objects
- EXEC CICS LINK and EXEC CICS XCTL API commands
- COBOL PERFORM and CALL commands
- C program function calls and single executable objects
- C++ Object Oriented design
- Internal and external procedures and functions in PL/I programs

Shareable, loadable, and executable operating system objects

Operating systems minimize storage requirements for normal executables by sharing program code to the extent possible. If two or more copies of the same program are running, each runs inside its own process and each is given a separate data area, the operating system loads only one copy of the program text.

Use the operating system's dynamic load facility when you call a CICS program. The operating system's dynamic load facility enables the program text being loaded to be shared by multiple processes. Although, the standard way of invoking other programs is to use the **fork** (on CICS on Open Systems) or **CreateProcess** (on CICS for Windows) or **exec** system calls (or variants), do *not* use these in CICS programs to call a CICS program because CICS programs are not normal executables.

By using the system's dynamic load facility, you minimize the amount of storage that CICS requires. It is worth doing this for programs that are likely to be heavily used by multiple concurrent users.

You can use the dynamic load facility for functions within your program. If you have a function that is common to several programs, then you can minimize your storage requirements by making it shared and dynamically loadable.

CICS links to the dynamic load facility through the **cicstcl** utility. The following command example displays the command on the AIX platform.

On CICS for AIX only
The linker called by the CICS utility **cicstcl** does not produce shareable objects by default, but you can specify this by passing the flag **-bM:SRE** through to the linker.

EXEC CICS LINK and EXEC CICS XCTL commands

The CICS commands EXEC CICS LINK and EXEC CICS XCTL are powerful CICS facilities that are similar to function calls and overlays. COBOL, C, C++ and PL/I, offer facilities similar to EXEC CICS LINK. For example, function calls in C and C++ or PERFORM and CALL usage in COBOL function similarly to the EXEC CICS LINK command. However, these languages do not offer a counterpart to EXEC CICS XCTL.

There are circumstances in which it is preferable to use EXEC CICS LINK and EXEC CICS XCTL rather than the facilities provided by the various languages. Some examples of circumstances when these commands are useful include these conditions:

- These CICS API commands are helpful when applied to abend handling, exceptional conditions, and attention identifiers. For example, if you have a common input-handling function that uses the EXEC CICS RECEIVE command, then it may be preferable to call it with EXEC CICS LINK. By using the EXEC CICS LINK command, the input-handling function can set its own attention identifier handlers.
- The EXEC CICS LINK command is helpful when it is employed to distribute the processing using the distributed program link (DPL) facility. This enables you to distribute the processing of your application to where the data resides. For example, if your region needs to apply a series of updates to a transient data queue that exists in a remote region, it may be more efficient to do this using DPL than function shipping each request individually. To achieve this without using EXEC CICS LINK will mean you will have to use a remote procedure call facility of some kind, for example the one provided by DCE.
- The EXEC CICS LINK command can be helpful in reducing the amount of code you need to write.
- The EXEC CICS XCTL command is helpful when you have a situation in which the calling function does not need to receive control back after the invoked function is called. In these circumstances it is preferable to use EXEC CICS XCTL than EXEC CICS LINK because less storage is used in the application server.
- The EXEC CICS LINK and EXEC CICS XCTL commands are helpful if you are planning on migrating the system you are writing to another CICS platform. Using these CICS API commands is preferable even at the cost of some efficiency.

COBOL PERFORM and CALL commands

In COBOL, you can use the CALL statement to execute another COBOL program as part of the calling one. The called program can either be statically linked-in with

the caller, or it can be dynamically loaded. Statically linking in the called program is faster but inflexible; dynamically loading a called program can be slower but more flexible.

Furthermore, by dynamically loading a called program, you can arrange for the called program to be shared. In this instance, the first call to it will be comparatively slow, but subsequent calls will be faster while it remains loaded. You should be aware that when you take this approach you are not descending a CICS logical level. (See "Application program logical levels" on page 174.)

Finally, in designing your COBOL applications, avoid recursion. COBOL products used with TXSeries CICS other than Micro Focus Server Express COBOL (on Open Systems) and Micro Focus Net Express (on Windows) provide only a limited support for recursion. For example, a COBOL program can be coded in such a way that it can use EXEC CICS LINK to directly LINK to itself, but a set of COBOL programs cannot be coded to indirectly LINK recursively. In an example of two programs, progA and progB, progA can directly LINK to itself (provided it has been coded to do so), but progA cannot LINK to progB and then have progB LINK back to progA.

If you are using Micro Focus Server Express COBOL or Micro Focus Net Express, you can use recursion using EXEC CICS LINK and EXEC CICS XCTL. You must specify the /DATA-CONTEXT flag when you compile the COBOL programs. Each time the program is called, it gets a new copy of working storage.

C program function calls and single executable objects

The C language encourages the division of related functions into one or more source files that are compiled separately and then combined to form an executable object. Each of the source files that you combine into a single CICS program can include EXEC CICS statements, but you should be aware that when you make a function call in C to a function that includes the CICS API, you are not descending a CICS logical level. Logical levels only apply to the use of the EXEC CICS LINK command. If you need to inherit the same set of attention identifier handlers as the calling function, you should make a function call; if your called function needs a new set, then you should make it into a CICS program and call it with EXEC CICS LINK or EXEC CICS XCTL. It is more efficient to call it as a C function.

C programs can include static data (similar to working storage in COBOL). CICS programs written in C are given a fresh copy of their static data for the only first instance of the program.

C++ Object Oriented design

The C++ language provides object oriented programming through such facilities as objects, data abstraction, inheritance, and polymorphism. Normally, each class declaration and definition is in separate header and source files, but this is not mandatory. The individual object files can be combined into one executable program or archived into shareable class libraries against which the executable is linked.

Each of the source files that you combine into CICS programs or class libraries can contain EXEC CICS statements. Neither function calls nor method invocation affect CICS logical levels, so the EXEC CICS LINK command must be used when a logical level is desired.

When using CICS statements in class templates, the statements must be translated before the template class definition is included in any program. In header files, avoid using CICS statements in inline functions and methods.

Perform all initialization of static objects in CICS C++ cached programs in the object constructors, since they are initialized only once.

Explicitly delete any dynamically allocated storage.

Note: Do not use CICS statements in static object constructors or destructors. Static constructors and destructors are called outside a CICS Logical Unit of Work, and so using CICS statements at these points can cause abends.

Internal and external procedures and functions in PL/I programs

PL/I programs can consist of multiple internal and external procedures and functions containing CICS calls.

A function or procedure call does not descend a logical level; logical levels apply only to LINKS.

Calls to external procedures and functions have their condition handling suspended across these calls.

The HANDLE ABEND LABEL command is not supported in PL/I.

Transaction data storage considerations

Storage options on a transaction system require consideration to achieve efficiency. This section discusses the effects on efficiency for storage options in the following areas:

- Storing data within a transaction
  - Transaction Work Area (TWA)
  - User storage acquired by EXEC CICS GETMAIN without SHARED option
  - COMMAREA in an EXEC CICS LINK or EXEC CICS XCTL
  - TIOA in an EXEC CICS LINK or EXEC CICS XCTL with INPUTMSG
  - Program storage
- Sharing data across transactions
  - Temporary Storage
  - Common Work Area (CWA)
  - Terminal user area (TUA)
  - COMMAREA in a RETURN
  - TIOA in a RETURN with INPUTMSG
  - Display screen
  - Intrapartition transient data
  - Your own files
  - User storage acquired by GETMAIN with SHARED option

Storing data within a transaction

Storage facilities that exist over the lifetime of a transaction include:

- Transaction work area (TWA)
- User storage (obtained by EXEC CICS GETMAIN commands issued without the SHARED option)
- COMMAREA in an EXEC CICS LINK or EXEC CICS XCTL command

- TIOA in an EXEC CICS LINK or EXEC CICS XCTL with INPUTMSG
- Program storage

All of these areas are main storage facilities and come from the same basic source: the task private pool of the application server. (See "Application servers" on page 10) None of them are recoverable, and none can be protected by resource-level security keys. They differ, however, in accessibility and duration and, therefore, each meets a different set of storage needs. The TIOA is a storage area that is associated with a terminal, and is allocated from the task shared pool.

Transaction work area (TWA): The size of the transaction work area is determined by the *TWASize* option in the Transaction Definitions (TD). If this is given a non-zero value, the TWA is always allocated, it lasts for the entire duration of the transaction, and it is accessible to all of the programs in the transaction. Processor overhead associated with using the TWA is minimal. You do not need an EXEC CICS GETMAIN command to access it, and you address it using a single EXEC CICS ADDRESS command.

The TWA is suitable for fairly small data storage requirements and for larger requirements that are both relatively fixed in size and are used more or less for the duration of the transaction. Because the TWA exists for the entire transaction, a large TWA size has much greater benefit for conversational than for pseudoconversational transactions.

User storage acquired by GETMAIN without SHARED option: User storage is available to all the programs in a transaction, but some effort is required to pass it across an EXEC CICS LINK or EXEC CICS XCTL. However, its size is not fixed, and it can be obtained (using EXEC CICS GETMAIN commands) just when the transaction requires it and returned as soon as it is no longer needed. Consequently, user storage is useful for large storage requirements that either are variable in size or have a shorter duration than the transaction. This storage is freed at task termination if it is not explicitly freed by the transaction or is shared storage.

An EXEC CICS GETMAIN command involves a large amount of processor overhead. Therefore, use it only for large amounts of storage. For smaller storage amounts, use the TWA , or group the requests together into a larger request. Although the storage acquired by an EXEC CICS GETMAIN command may be held somewhat longer when using combined requests, the processor overhead and the reference set size are both reduced.

COMMAREA in an EXEC CICS LINK or EXEC CICS XCTL command: A communication area (COMMAREA) is a command-level facility used to transfer information between two programs within a transaction or between two transactions from the same terminal. For further information, see "Passing data to other programs" on page 175.

TIOA in an EXEC CICS LINK or EXEC CICS XCTL command with INPUTMSG: A terminal data area (TIOA) is a facility that tnsfers terminal data between two programs within a transaction or between two transactions from the same terminal. For further information, see "Passing data to other programs" on page 175.

Program storage: In COBOL, when the EXEC CICS LINK and EXEC CICS XCTL commands invoke the program, a new copy of the working storage is loaded. However, programs called with the CALL statement use existing working storage

if the program has already been invoked. In C and C++, recursive invocations of a program in the same transaction share one copy of its statics and external data.

Sharing data across transactions

The following sections discuss how data can be shared intra- and inter-transaction. A common characteristic between all of these storage areas is that they perform optimally for a small-to-medium amount of data, very often of a nonpermanent nature. These transient data and temporary storage service API commands allow access to sequential and random access queues. The efficiency aspects of intrapartition transient data queues and of temporary storage are discussed in "Sharing data across transactions."

CICS facilities that exist beyond the end of a task are:

- Temporary storage
- Common work area (CWA)
- Terminal user area (TUA)
- COMMAREA in a RETURN
- TIOA in a RETURN with INPUTMSG
- Display screen
- Intrapartition transient data
- Operating System files
- Storage acquired by GETMAIN with SHARED option

The last three items provide more flexibility and function than the other listed items; therefore, their use incurs more overhead. You can use any of these methods for passing data either within transactions or across transactions.

With the exception of COMMAREA and the display screen, data stored in these facilities is available to any transaction in the system. Subject to resource-level security restrictions for transient data and temporary storage queues, any transaction may write to them and any transaction may read them.

The operating system also provides shared memory facilities which you can use to share data between transactions, but you will be reducing application portability if you use these schemes.

Temporary storage: Temporary storage is the primary CICS facility for storing data that must be available to multiple transactions. Data items in temporary storage are kept in queues whose names are assigned dynamically by the program storing the data. You do not need to define temporary storage queues in the resource definitions, unlike transient data queues. A temporary storage queue containing multiple items can be thought of as a miniature file whose records can be addressed either sequentially or directly, by item number. If a queue contains only a single item, it can be thought of as a named scratchpad area.

Temporary storage is implemented in two different ways. Which one is used for a particular queue is determined by what is specified on the command that creates the first item. MAIN means that the queue is kept in memory, and AUXILIARY means that the queue is written to disk storage. For either method, CICS maintains an index of items in main storage.

Both these methods have characteristics that you should bear in mind:

- Main temporary storage requires much more virtual storage than does auxiliary temporary storage. In general, you should use it only for small queues that have short lifetimes or are accessed frequently. Auxiliary temporary storage is

specifically designed for relatively large amounts of data that have a relatively long lifetime or are accessed infrequently.

- You can make queues in auxiliary storage recoverable, but not queues in main storage.

  Only one transaction at a time can update a recoverable temporary storage queue. If you choose to make queues recoverable, bear in mind the possibility of enqueues.
- If a task tries to write to temporary storage and there is no space available, CICS suspends it. The task is not resumed until some other task frees the necessary space in the file. This can produce unexplained response delays, especially if the waiting task owns exclusive-use resources, in which case all other tasks needing those resources must also wait. You can use the NOSUSPEND option to avoid this; see "The NOSUSPEND option" on page 37.
- It can be more efficient to use main temporary storage exclusively in very low-volume systems that have no need for recovery.

The following points apply to temporary storage in general:

- You must use a CICS command every time data is written to or read from a temporary storage queue, and CICS must find or insert the data using its internal index. This means that the overhead for using main temporary storage is greater than for the CWA or terminal user area. With auxiliary storage, often the most frequently used, there is usually file I/O as well, which increases overhead even more.
- You need not allocate temporary storage until it is required; you need to keep it only as long as it is required, and the item size is not fixed until you issue the command that creates it. This makes it a good choice for relatively high-volume data and data that varies in length or duration.
- The fact that temporary storage queues are named as they are created provides a very powerful form of random access to saved data. You can access scratchpad areas for terminals, file records, and so on, simply by including the terminal name or record key in the queue name.
- Resource-level protection is available for temporary storage.

Common work area (CWA): The common work area (CWA) is a single control block that is allocated at system startup time and it exists for the region. The size is fixed by specifying a value for the **CWASize** parameter in the Region Definition (RD). This means that the CWA has the following characteristics:

- There is almost no overhead in storing or retrieving data from the CWA. Transaction programs must issue one ADDRESS command to get the address of the area but, after that, they can access it directly.
- Data in the CWA is not recovered if a transaction or the system fails.
- It is not subject to resource-level security.
- CICS does not regulate use of the CWA. All programs in all applications that use the CWA must follow the same rules for shared use. The content of and access to the CWA will be one of the design issues that you will need to discuss with application developers. It is sensible to supply a copybook or include file to describe the contents.
- The CWA is especially suitable for small amounts of data, such as status information, that are read or updated frequently by multiple programs in an application.
- The CWA is not suitable for large-volume or short-lived data because it is always allocated.

- You must ensure that data used in one transaction does not overlay data used in another, by following the rules for shared use discussed above.
- You must ensure that programs do not overrun the end of the CWA as this corrupts the storage areas in the Task Shared Pool, causing other transactions to fail.

Terminal user area (TCTUA): The terminal user area is defined using the *TCTUALen* option in the Terminal Definitions (WD). If this length is not zero, the address of the area can be obtained with an EXEC CICS ADDRESS command. Terminal user areas have the following characteristics in common with the CWA:
- Minimal processor overhead (only one ADDRESS command needed)
- No recovery
- No resource-level security
- No regulation of use by CICS
- Fixed length
- Unsuitable for large-volume or short-lived data

Unlike the CWA, however, the terminal user area for a particular terminal is usually shared only among transactions using that terminal. It is therefore useful for storing small amounts of data of fairly standard length between a series of pseudoconversational transactions. Another difference is that it is not necessarily permanently allocated, because the terminal user area only exists while the Terminal Definition is set up. The Terminal User Area is allocated from system startup for nonautoinstall terminals and when the Terminal Definition is generated for autoinstall terminals. It is discarded for autoinstalled terminals when their destination is deleted.

Using the terminal user area in this way does not require special discipline among using transactions, because data is always read by the transaction following the one that wrote it. However, if you use terminal user areas to store longer-term data (for example, terminal or operator information needed by an entire application), they require the same care as the CWA to ensure that data used in one transaction does not overlay data used in another. You should also take care not to exceed the length of the allocated terminal user areas, because this causes corruption in other storage areas allocated from the CICS Task Shared Pool.

COMMAREA in an EXEC CICS RETURN command: The COMMAREA is used to pass information between application programs. For further information, see "Passing data to other programs" on page 175.

TIOA in an EXEC CICS RETURN command with INPUTMSG: The TIOA is used to pass terminal data between application programs. For further information, see "Passing data to other programs" on page 175.

Display screen: You can also store data between pseudoconversational transactions from a display terminal in the display screen itself. For example, if users make errors in data that they enter, the transaction usually points out the errors on the screen (with highlights or messages), sets the next transaction identifier to point to itself (so that it processes the corrected input), and returns to CICS.

The transaction has two ways of using the valid data. It can save it (for example, in the COMMAREA) and pass it on for the next time it is run. In this case, the transaction must merge the changed data on the screen with the data from previous entries. Alternatively, it can save the data on the screen by not turning off the modified data tags of the keyed fields.

Saving the data on the screen is easy to code, but has two limitations. First, you should not use it with screens that contain large amounts of data if the likelihood of errors is high. This is because of the additional line traffic needed to resend the unchanged data. (This does not apply to locally-attached terminals.)

Second, if the user presses the CLEAR key, the screen data is lost, and the transaction must be able to recover from this. You can avoid this by defining the CLEAR key to mean CANCEL or QUIT, if this is appropriate for the application concerned.

Data other than keyed data may also be stored on the screen. This data can be protected from changes (except those caused by CLEAR) and can be nondisplay, if necessary.

Intrapartition transient data: Intrapartition transient data has some characteristics in common with auxiliary temporary storage. Like temporary storage, intrapartition transient data consists of queues of data, kept together in a single data set, with an index that CICS maintains in main storage.

You can use transient data for many of the purposes for which you would use auxiliary temporary storage, but there are some important differences:

- Transient data queue names must be defined in the Transient Data Definitions (TDD) before CICS is started. You cannot define them arbitrarily at the time the data is created. Thus, transient data does not have the same dynamic characteristics as temporary storage.
- Transient data queues must be read sequentially, and each item can be read only once. That is, after a transaction reads an item, that item is removed from the queue and is not available to any other transaction. In contrast, items in temporary storage queues may be read either sequentially or directly (by item number). They can be read any number of times and are never removed from the queue until the entire queue is purged.

  These two characteristics make transient data inappropriate for scratchpad data but suitable for queued data such as audit trails and output to be printed. In fact, for data that is read sequentially once, transient data is preferable to temporary storage.
- Items in a temporary storage queue can be changed; items in transient data cannot.
- Transient data queues are always written to a data set. (There is no form of transient data that corresponds to main temporary storage.)
- You can define transient data queues so that writing items to the queue causes a specific transaction to be initiated (for example, to process the queue). Temporary storage has nothing that corresponds to this ‘trigger’ mechanism, although you may be able to use a START command to perform a similar function.
- Transient data has more varied recovery options than temporary storage. It can be physically or logically recoverable.
- Because the commands for intrapartition and extrapartition transient data are identical, you can switch easily between the internal CICS facility (intrapartition) and an external data set. To do this, you need only change the TDD, not your application programs. Temporary storage has no corresponding function of this kind.

Operating System files: You can also use operating system files to save data between transactions. This method probably has the largest overhead in terms of

instructions processed, buffers, control blocks, and user programming requirements, but does provide extra function and flexibility. Not only can you define files as recoverable resources, but you can log changes to them for forward recovery. You can also use resource-level security.

User storage acquired by GETMAIN with shared option: The SHARED option of the GETMAIN command can be used to acquire storage that is shared between transactions. For further information, see "Storage services" on page 182.

Data management storage considerations

This section provides an overview of the data storage facilities used in data management and suggests where to store your data.

The CICS data storage facilities are:

- **File services**

  CICS uses file services to read, update, add, delete, and browse data in local or remote files. See "File services" on page 105.
- **Queue services**

  In addition to the permanent data that is the basis of your application, you possibly need to create temporary data for purposes like processing requests and passing data between tasks or programs. CICS provides queue services for this. The advantage of CICS queue services is that they persist over multiple executions of CICS and can represent permanent data. See "Queue services" on page 130.
- **Relational database services**

  CICS allows access to relational databases that provide a programmable interface through Structured Query Language (SQL) commands in either COBOL, C, C++, or PL/I programs.
- **Journal services**

  CICS provides facilities for creating and managing journals during CICS processing. A *journal* is a set of special-purpose sequential files. Journals can contain any and all data the user needs to facilitate subsequent reconstruction of events or data changes. See "Journal services" on page 137.

See also "Data services API migration" on page 205.

Issues affecting your data management storage decisions

The list below summarizes the key issues to consider in deciding which of the CICS data storage facilities meets the needs of your application data. A more detailed discussion on each issue follows this summary list.

- **Structure and function**

  You must consider the structure of your data to ensure a match between the application programming interface, data, and your processing requirements. You also need to match the auxiliary function in the facility to your needs when managing the data.
- **Sharing and distribution**

  You need to consider:
  - How much data is shared among regions
  - How much data is accessed outside CICS
  - How much data is distributed among multiple processors
- **Data integrity**

For shared or distributed data, you must consider data integrity. Programming requirements to ensure data integrity are more complex when data is shared or distributed.

- **Data backup and recovery**

  If your data is vital, you must consider how to recover the data if it is lost. Recovery might be provided by CICS or another facility. Programming requirements to ensure recoverability are more complex when data is shared or distributed.

- **Portability**

  If you need to store or access your data from multiple platforms (AIX and MVS for example), you must ensure that your choice is supported in all potential environments. Possible future cross-platform requirements also need to be considered.

- **Performance**

  Some forms of storage are much more efficient than others. There is generally a trade-off between performance and function, so you must balance your other requirements against your performance requirements.

Structure and function: The most important difference between data stored by a database management system and data stored in either flat files or queues is the structure that the storage manager imposes on the data. This structure dictates the application programming interface to the data and determines how easy or hard it is to store and retrieve the data for a particular processing requirement. If the data is complex, the structure can be the overriding consideration.

A related difference is where knowledge of the data structure is stored. In a database management system (DBMS), the logical structure of the data resides in the DBMS. The physical structure can be changed considerably without changing application code. In flat files, the logical structure of the data is embedded in the programs using it, and logical and physical structures coincide.

DBMSs provide services and utilities for managing recovery, sharing and distribution that can be essential to your application. If your data is voluminous, recovery and other management functions can dictate that you use a DBMS.

Sharing and distribution: Another storage consideration is the potential users of the data. Will users access the data from more than one CICS platform? Is the access made concurrently or serially?

The inability to share data has historically been an impediment to application growth and change. The problems arise as follows:

- Increases occur in both the size of the data and the frequency with which the data is accessed: Growth in demands for data occur because applications grow in volume beyond expectations, or a successful application spawns other applications which make use the same data. The applications no longer fit on the processor where they began, and sometimes, they cannot fit on the largest system available. These applications have to be split into multiple CICS regions, and possibly, the data needs to be shared among these regions.
- Requirements change for data access availability: Applications originally intended to be online for only part of the day can grow to demand 24-hour availability. Data sharing becomes necessary because there is no time when CICS is not operating to run the associated batch cycle. This associated processing must be done while CICS is running.

- Reporting requirements change due to increasing time demands in the business environment: Batch reports require time to program, and results can be obtained on an established time cycle. Today's business environment can demand instant access to the data maintained by an application. In addition, business requirements can call for spontaneous manipulation of data for analytical purposes. When the data is critical to business operations, time constraints prohibit programming those reports individually. The data must be available to be shared with a query manager or report generator.

Current hardware and software makes it practical to distribute a single database among several remote locations. This distribution enables the data to be kept local to the code which most often uses it; yet, it permits access by other users. This can improve response time; however, it also introduces the need to ensure data integrity and recovery capability in the event of a communication failure.

Data integrity: Data integrity is an issue with all shared data. There is no problem if all of the parties sharing the data only read it. However, when even one party updates data, there is a risk that data can become obsolete while other parties are reading it. This loss of *read integrity* has varying importance, depending on the task. If the task is to check the data on a customer's charge card balance and authorize a large purchase, it might be critical. If the task is to analyze the customer's buying habits, it might be less so.

If more than one party can update the files, there is greater risk to data integrity. If two tasks read the same data item with intent to update, updates made by the first task are lost when the second task updates. This occurs because the second task is working from a copy of the data that does not reflect the changes made by the first. This phenomenon of the lost update almost always matters, and most systems protect you against it by enforcing *write integrity*.

Write integrity can be maintained by serializing all updates. Serializing is done by *locking* data as soon as a task expresses intent to update and delaying any other task that wants to update the same data until the holder of the lock updates and releases the lock. Read integrity can be accomplished the same way if readers, as well as updaters, lock the data.

However, since locks imply delays, full read integrity is usually optional.

To work, locks must be imposed by a program that has control over all the potential users of the data. Database managers, the access methods, CICS, and the operating system all do some locking to prevent conflicts among the users they control. Among the tasks in a single region, CICS enforces write integrity and, optionally, partial read integrity. However, CICS cannot control the effects of sharing by outside users, and the type of sharing that can be done with integrity varies significantly among the various storage facilities.

Internal data consistency: The integrity losses just described occur because tasks running in parallel can access a single item of data while it is undergoing change. Another integrity issue arises when there are processing relationships between two or more items of data. For example, you have one body of data representing accounts payable and another representing your purchase ledger. If you pay a supplier, you need to update both your ledger (to reflect the money paid out) and your accounts payable (to indicate payment has taken place), and you need to write a check. You need to do all three updates together to maintain the consistency (integrity) of your data. The single logical change to your data is made

up of all three updates together. If your task fails after making any one of the updates, you need some way to remove that all updates, so the data is internally consistent.

Database management systems generally provide integrity for the data they manage. Changes to databases are considered provisional until the task making them issues a request to commit the changes, at which time they are all made simultaneously. However, a database manager provides this kind of integrity only to data under its control. CICS extends the concept to allow you to define a set of related changes to data stored in different facilities, including direct access files, queues, and database managers.

CICS guarantees that if it cannot complete all the changes to protected data in a single logical unit of work (LUW), it makes none of them. For example, if you have made half of a sequence of related updates when a computer fails, CICS backs out the changes it has made so far in the logical unit of work before resuming. In general, you decide which data is to be protected in this way, although sometimes database managers have a say in the matter. Not all types of data are eligible, and if the integrity of your data is vital, you must consider this when you choose how to store it.

Dynamic backout and recovery:

There are two situations in which updates to resources can possibly need to be undone: an individual task can fail part way through an LUW, or the whole system can fail. (LUWs generally coincide with tasks, although it is possible to break a task into multiple LUWs; an LUW never spans tasks.)

The resources updated within a single LUW can belong to more than one resource manager. When this situation occurs, CICS recovery actions include the joint responsibility of all the resource managers involved. For example, when you are using an XA-compliant relational database management system (RDBMS), CICS acts as the XA-compliant transaction manager and controls the backout actions both of itself (as a resource manager) and of the RDBMS; on the other hand, SFS is itself a recoverable resource manager. Therefore, although CICS keeps track of such recoverable entities as transient data queue read and write pointers, it is SFS that performs the major backout and recovery work.

CICS saves the original and current state of all of its recoverable data that has been changed in the current LUW. It saves this information in the Region Pool. If the LUW needs to be *rolled back*, CICS simply discards the current state information, leaving the data in its original state. It also instructs all of the resource managers involved in the LUW to roll back their work.

When there is a failure that results in the abnormal termination of a CICS region, on restart, CICS has to process every LUW that was incomplete at the time of the failure. Whether or not other resource managers have to do likewise is dependent upon whether they also failed. For example: a region has only one SFS was running, and that server did not fail. In this case, the server is still running, and it is necessary to restart only CICS.

Restoring the state of recoverable resources after a system failure requires an external record of all the work that needs to be redone. For this purpose, CICS is configured to periodically take snapshots (*checkpoints*) of the states of all recoverable resources. On restart, CICS reads the checkpoint to reestablish the

states of the recoverable resources at the time the checkpoint was written, and then processes all relevant information held for the region.

The frequency with which this checkpoint occurs is configurable. Refer to the **CheckpointInterval** attribute described in the *CICS Administration Reference*.

Guaranteeing integrity across an LUW involves the need to protect all involved tasks so that other LUWs cannot update them until the current LUW ends. This guarantees that only the LUW having the failed transactions needs to be rolled back. Therefore, the locks that prevent concurrent updates must be extended to the end of the LUW. Extending the locks, of course, increases the associated delays.

If a task just wants to read data that has been updated in an uncommitted LUW, it might use data that is subsequently changed because the LUW that wrote it failed. To ensure read integrity in this situation, readers as well as updaters of uncommitted data must be locked out until the updating LUW ends.

Data backup and recovery:

The task of keeping backup copies is more complicated with online systems, because the data is constantly changing. One way to handle backups is to make a full copy periodically, and keep a record of subsequent changes. In this way, you can reconstruct the current data by reapplying these changes to the most recent full copy. Another way to handle backups is to write duplicate copies of critical databases. This can be done synchronously, so that the two copies are kept exactly the same, or asynchronously, where the backup copy can be slightly behind the primary one. Many users keep one copy physically remote from the normal processing site.

In addition to the time and operational complexity of making the copies, there are online costs associated with protecting your data. First, there is the overhead of recording the changes. (Recovery requires an image of each changed resource after the change.) Second, these *after images* must be recorded on an external medium by the time the task making them reaches syncpoint. This requirement ensures that even if CICS malfunctions at that time, no committed changes are lost in the process. Furthermore, this external recording of the after image requires that the task wait for the I/O to complete before the syncpoint processing can be completed.

Considerations involved in deciding how to handle backups include the following issues:

- Is your data is voluminous or volatile?
- How long does it take to make the copies?
- How frequently are backups required?
- How long can your business sustain an outage while you rebuild your data after an accident?

The business demands and the data storage facilities available in each situation will dictate your approach to backup requirements.

The following operational concerns are relevant to backup decisions:

- Who will perform the process and how much automation is required? (This is particularly important if more than one form of data storage is involved.)

- Is there sufficient operations staff to perform the day to day requirements of taking backups, maintaining the storage devices, and ensuring data availability that meets the time requirements?

Portability: The location where the transaction processing occurs is more flexible in a distributed system. This makes your design decisions more complex. We recommend that you design your transactions so that the business logic is separate from the user interface. This allows you to execute the two parts on different processors. The execution can occur entirely either under IBM CICS for Windows or CICS on Open Systems, or it can occur under other CICS products. With the proper precautions, the CICS platform on which the business logic executes can be changed without changing your application. If a platform change is likely to be required, portability must be considered when you choose how to store your data.

Refer to Chapter 7, "Migrating CICS applications to and from TXSeries CICS," on page 191 for migration considerations. Also refer to the API command descriptions in the *CICS Application Programming Reference*, where differences are noted for each individual command.

Performance: For most online applications, the major component of transaction processing time is occupied with the reading and writing of externally stored data. The task waits for physical I/O to occur, and the depth of the embedded directory level (pathlength) used for storage and retrieval consumes the greater share of the transaction time. Therefore, the greatest impact on the performance and resource requirements of your application comes from your choice of data storage facilities and the way you use any given facility.

In addition to the functional requirements, your design needs to consider the following performance trade offs:

- In general, the longer the pathlength, the more performance is reduced. Longer Pathlength are associated with the power of the language function. The more powerful language functions are associated with programming productivity. Your design often trades between performance and productivity.
- The more universal the sharing, the more performance is enhanced. But, data sharing requires significant programming effort to ensure the proper management of the data. This can make developing and maintaining the application more costly. However, designing the application to ensure easy maintenance can result in reduced performance.

There are no right answers to these trade-off decisions. It is critical to understand the issues and to make trade-off decisions consciously during the design phase. It is particularly important to understand the expectations for response time and transaction volume. These aspects, along with recovery, integrity, sharing, and other requirements, must be carefully examined to ensure that your design choices meet objectives in performance, as well as in function.

CICS environment efficiency considerations

To know how programming techniques can affect the performance and efficiency of the CICS system, it is necessary to understand a little about the environment in which CICS operates. This section discusses the following factors:

- Wait conditions
- Auxiliary trace
- The NOSUSPEND option
- Access permissions for maps and transaction programs
- CICS commands lengths

Wait conditions

In some CICS products, it is possible to inadvertently stop all CICS activity by using a facility that causes an operating system wait. This can only occur in CICS for system dumps, whether operator or software initiated.

Auxiliary trace

Efficiency can be improved by using auxiliary trace to review your application programs. It can identify many common coding problems, such as:

- unnecessary code
- too many or too large EXEC CICS GETMAIN commands
- failure to release storage when it is no longer needed
- failure to unlock records held for exclusive control that are no longer needed
- unintended logic loops.

The NOSUSPEND option

The default action for the ENQBUSY, NOJBUFSP, NOSPACE, QBUSY, and SYSBUSY conditions is to suspend the execution of the application until the required resource (for example, a queue) becomes available, and then resume processing the command. The commands that can cause these conditions are: EXEC CICS ALLOCATE, EXEC CICS CONNECT PROCESS, EXEC CICS ENQ, EXEC CICS JOURNAL, EXEC CICS READQ TD, and EXEC CICS WRITEQ TS.

With these commands, you can use the NOSUSPEND option to inhibit this waiting and to cause an immediate return to the instruction in the application program following the command. (In the case of the EXEC CICS ALLOCATE command, this option is known as the NOQUEUE option.)

If you do not use the NOSUSPEND option, the suspended applications attempt to obtain the required resource periodically until it becomes available. This can consume significant resources. For example, processor time can be wasted, or the user can be prevented from typing anything by a terminal lock until the resource is available.

Access permissions for maps and transaction programs

When you produce a transaction program or map using the CICS commands **cicstcl**, **cicsmap**, or **cicstran**, followed by the link-edit stage, you must ensure that the resultant file is either readable by the group cics or has public access.

CICS commands lengths

When a CICS command includes a LENGTH operand, the data type used usually places a theoretical upper limit of 32K bytes on the length. In practice, the limits are less than this and vary for each command. The limits depend on file definitions, recoverability requirements, buffer sizes, and local networking characteristics.

Length options: In the COBOL, C, C++, or PL/I languages, the translator deals with lengths. Refer to the individual command descriptions in the *CICS Application Programming Reference* for details of when you need to specify the LENGTH option.

In most cases, the LENGTH option must be specified if SET is used; the syntax of each command and its associated options show whether this rule applies.

For journaling commands, the restrictions apply to the sum of the LENGTH and PFXLENG values. For further details, see the *CICS Application Programming Reference*. Note that for journal records, the journal buffer size may impose a limit lower than 24KB.

Recommendation: For any command, 24KB is a good working limit for LENGTH specifications. Subject to user-specified record and buffer sizes, this limit is unlikely either to cause an error or to place a constraint on applications.

You will probably not find a 24KB limit too much of a hindrance; online programs do not often handle such large amounts of data for the sake of efficiency and response time.

Note: The value in the LENGTH operand should never exceed the length of the data area addressed by the command.

Efficiency issues for CICS locking functions

This section describes the following locking functions performed by the Structured File Server (SFS) and IBM's DB2 when processing transactions:

- Implicit locking on recoverable and nonrecoverable files
- Implicit locking on recoverable transient data queues
- Implicit locking on recoverable temporary storage queues
- Explicit locking

Note: Locking (implicit or explicit) on data resources protects data integrity, but can affect performance if several tasks attempt to operate on the same data resource at the same time. The effect of locking on performance, however, is minimized by using short Logical Units of Work (LUW).

Implicit locking on nonrecoverable and recoverable files

This section discusses the following locking functions:

- the locking provided when DB2 is used to manage CICS queues and files
- the implicit locking (exclusive control) that SFS provides when you update nonrecoverable files, and
- the extended locking actions that SFS provides when you update recoverable files

Note: DB2 handles all files as recoverable files. For nonrecoverable SFS files, SFS locks the record during an update.

Figure 1 on page 39 shows two tasks updating the same record. Task A is given exclusive control of the record between the EXEC CICS READ UPDATE and EXEC CICS WRITE commands. During this period, Task B waits.

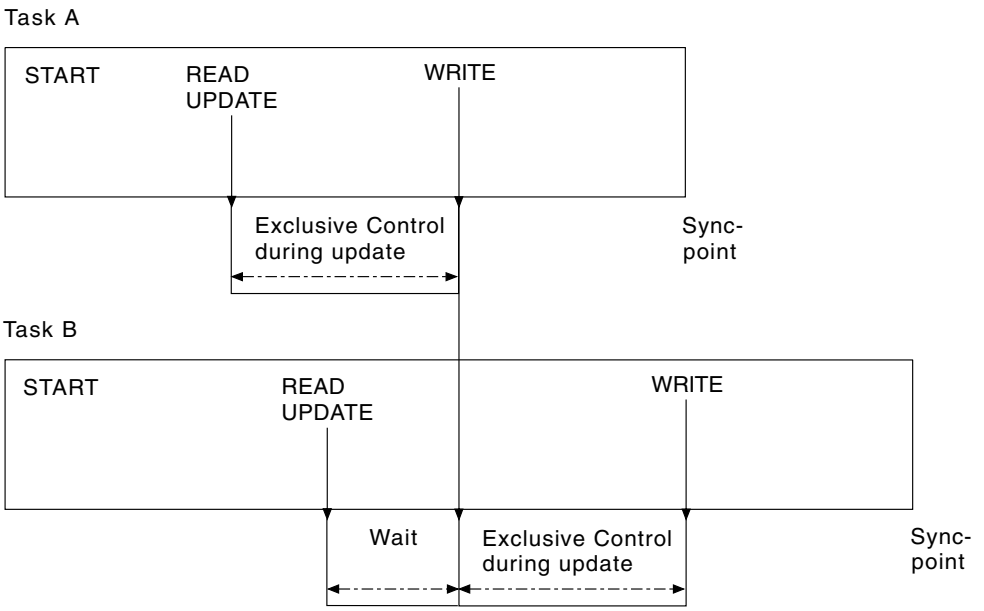


Figure 1. Locking (exclusive control) during updates to nonrecoverable files

Figure 2 illustrates two tasks updating the same record in recoverable SFS files or DB2 files. Task A is given exclusive control of the record until the update is committed (at the end of the LUW). During this period, Task B waits.

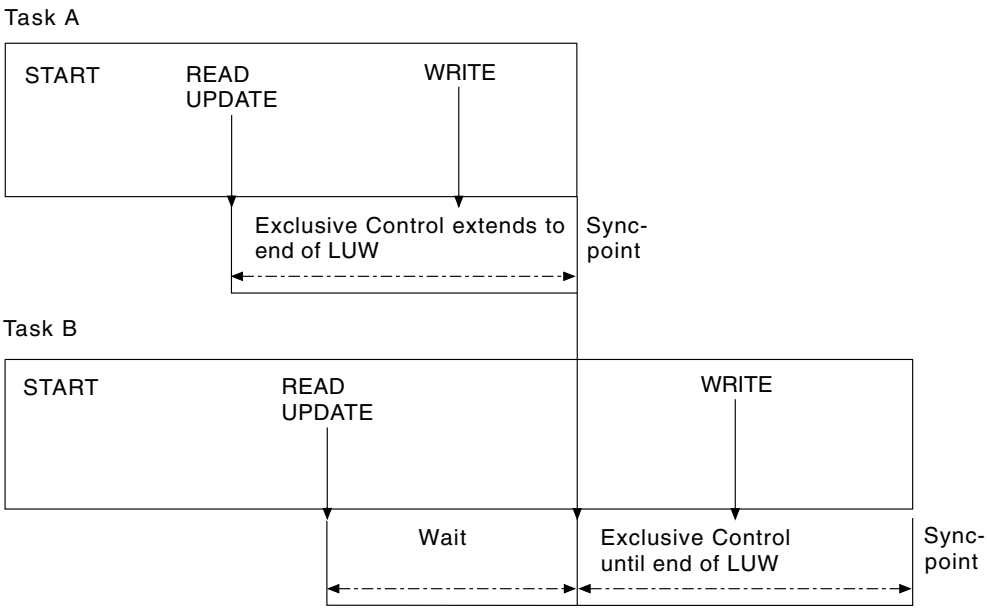


Figure 2. Locking (exclusive control) during updates to recoverable files

The extended period of exclusive control is needed to avoid the possibility of an update committed by one task being backed out by another task. Consider what could happen if you used the nonextended exclusive control shown in Figure 1. If Task B reads the data after Task A wrote it but before the syncpoint (which it could with nonrecoverable files), and Task A then backs out its changes, Task B would have the incorrect value.

To avoid this problem, whenever a transaction issues a command that changes a recoverable file (or reads from a recoverable file prior to update), SFS and DB2 automatically locks the updated record until the change is committed (that is, until the end of the LUW). Thus in Figure 2 on page 39, Task B would not be able to access the record until Task A had committed its change at the end of the LUW. Hence, it becomes impossible for Task B's update to be lost by a backout of Task A.

The file control commands that invoke automatic locking in this way are:
- EXEC CICS READ (for UPDATE)
- EXEC CICS WRITE
- EXEC CICS DELETE

Note: Locking can lead to transaction deadlock. If a transaction requests a record for update that is being updated by another task, the second task is locked until the first update is complete.

When a transaction issues an EXEC CICS READ UPDATE command (for any file, recoverable or not), SFS or DB2 maintain exclusive control of the record until an EXEC CICS REWRITE (or UNLOCK or DELETE) command is issued. Two EXEC CICS READ UPDATE commands for records in the same file without an intervening EXEC CICS REWRITE command will raise the INVREQ condition.

For recoverable files, you should not use unique key alternate indexes to allocate unique resources (represented by the alternate key). This is because backout failure may occur in the following set of circumstances:
- A task deletes or updates a record (through the base or another alternate index and the alternate index key is changed.
- Before the end of the first task's LUW, a second task inserts a new record with the original alternate index key, or changes an existing alternate index key to that of the original one.
- The first task fails and CICS attempts to back it out.

The backout fails because a duplicate key is detected in the alternate index. There was no locking on the alternate index key to prevent the second task taking it before the end of the first task's LUW. If there is an application requirement for this sort of operation, the File Server's locking mechanism must reserve the key until the end of the LUW.

To ensure that the data being read is up to date, the application program should issue an EXEC CICS READ UPDATE command (rather than a simple READ), thus locking on the data until the end of the LUW.

Implicit locking on logically recoverable transient data queues

CICS provides a facility for locking logically recoverable (as distinct from physically recoverable) transient data queues in a similar way to that for recoverable files.

Note that if DB2 is used as the file manager for CICS, all transient data (TD) queues are treated as logically recoverable.

Transient data control commands that invoke implicit locking are:
- WRITEQ TD
- READQ TD
- DELETEQ TD

Thus, for example:

- If a task issues an EXEC CICS WRITEQ TD command to a particular destination, CICS locks that write destination until the end of the task (or LUW). While the destination is locked:
  - Another task attempting to write to the same destination is suspended.
  - Another task attempting to read from the same destination is allowed to read only committed data (not data that is being written in a currently incomplete LUW). When trying to read uncommitted data, the task suspends until that data is committed, unless the NOSUSPEND option is used on the READQ command.
- If a task issues an EXEC CICS READQ TD command to a particular destination, CICS locks that read destination until the end of task (or LUW). While the destination is locked:
  - Another task attempting to read from the same destination is suspended.
  - Another task attempting to write to the same destination is allowed to do so. CICS locks that write destination until end of task (or LUW).

Implicit locking on recoverable temporary storage queues

CICS provides the locking protection facility for recoverable temporary storage queues in a similar way to that for recoverable files. There is one minor difference; when the file manager is SFS, locking is not invoked for EXEC CICS READQ TS commands, therefore it is possible for one task to read a temporary storage queue record while another is updating the same record (this allows the reading of uncommitted data, or "dirty reads"). To avoid this, it is necessary to use explicit locking on temporary storage queues where concurrently executing tasks can read and change queues with the same temporary storage identifier.

Temporary storage control commands that invoke implicit locking are:

- EXEC CICS WRITEQ TS
- EXEC CICS DELETEQ TS

Explicit locking (by the application programmer)

CICS provides the following explicit locking commands:

- EXEC CICS ENQ RESOURCE
- EXEC CICS DEQ RESOURCE

These commands can be useful in certain applications where, for example, the installation wants:

- To protect data that is not automatically protected by CICS for example, when the data is written into the common work area (CWA).
- To prevent transaction deadlock by locking on records that might be updated by more than one task concurrently.
- To protect a temporary storage queue from being read and updated concurrently.

To be effective, however, all transactions must adhere to the same convention. A transaction that accesses the CWA without using the agreed ENQ and DEQ commands will not be suspended, and protection will be violated. It follows that you will need installation standards.

After a task has issued an EXEC CICS ENQ RESOURCE (*data area*) command, any other task that issues an EXEC CICS ENQ RESOURCE command with the same data area parameter is suspended until the task issues a matching EXEC CICS DEQ RESOURCE (*data area*) command, or until the LUW ends, unless the NOSUSPEND option is used.

Note: The concurrent use of locks against more than one resource introduces the possibility of transaction deadlock.

See the *CICS Application Programming Reference* and the *CICS Administration Guide* for related information.

Performance considerations for CICS developers

This section identifies the performance issues that generally fall into the domain of the developer.

Improving performance of CICS application programs

Writers of CICS application programs need to consider the following points:

- The impact of storage class, see "Selecting the appropriate class of storage."
- The value of using BMS map suffixing, see "BMS map suffixing."
- The value of shared libraries, see "Resident options and shared libraries" on page 43.
- The importance of avoids locks, see "Avoiding locks" on page 43.
- The impact of reading programs, see "Reading programs" on page 44.
- The value of caching application code, see "Using the program cache" on page 44.

Selecting the appropriate class of storage

- EXEC CICS GETMAIN

  GETMAIN storage is allocated with a mechanism similar to operating system **malloc** and **free** routines, and comes from the process data segment. CICS imposes a limit, configurable in the Region Definitions (RD), on the amount of this type of storage that is given to a transaction program. This class of storage is reclaimed by CICS if the transaction program abends or terminates without releasing it.

- EXEC CICS GETMAIN SHARED

  GETMAIN SHARED is allocated from storage shared between all CICS tasks. The total available is configured in the region database.

  GETMAIN SHARED remains allocated until explicitly freed by a transaction program.

- Stack

  You can use the C language function **alloca** to allocate stack storage for automatic variables and dynamically sized automatic variables. This storage is the quickest to allocate and deallocate and is automatically cleaned up when the function exits or the transaction program abends. The application programmer must ensure that this memory is freed at an appropriate time. Because it is not simple to defend against the possibility that a transaction program will abend while holding **malloc** storage, it is normal for transaction programs to use EXEC CICS GETMAIN instead.

BMS map suffixing

Device-specific maps are found more quickly than device-independent maps. Creation of both types of maps is described in the *CICS Application Programming Guide*.

Resident options and shared libraries

On CICS for AIX only
The size of a C, IBM COBOL, or PL/I transaction program can be determined by using the **size -f** command.

On the HP-UX or Solaris platforms
The size of the C transaction program can be determined by using the **size -f** command.

When the **size -f** command is used, the fields `text` (program code), `data` (initialized data), and `bss` (zeroed data) are relevant.

Operating-system programs typically make heavy use of shared libraries to reduce code size. The use of shared libraries for programs containing CICS dependencies, such as EXEC CICS calls, is not currently supported. However, you can still possibly benefit from using shared libraries for common functions that, although called from a CICS program, do not themselves contain any CICS dependencies.

On UNIX, the size of a Micro Focus Server Express COBOL program can be determined with the **ls -l** command. Micro Focus Server Express COBOL programs are managed by the CICS Micro Focus Server Express COBOL runtime. It is possible for Micro Focus Server Express COBOL programs to invoke other Server Express COBOL programs using EXEC CICS LINK, and to access C functions by rebuilding the CICS Micro Focus Server Express COBOL runtime or by using EXEC CICS LINK.

Using EXEC CICS LINK for this purpose can cause a reduction in storage use at the expense of processing time.

Avoiding locks

Transaction programs can run concurrently as long as they do not depend on or update the same data. If they do require the same data, the second transaction must possibly wait until the first transaction reaches syncpoint.

You can design your application to minimize the amount of contention for locks. For example, suppose that you must keep a count in a file of the number of transactions executed in your system. Every transaction needs to contain statements like the following:

```
EXEC CICS READ FILE("count_file") UPDATE ... ;
count = count + 1 ;
EXEC CICS REWRITE FILE("count_file") ;
```

Therefore, every transaction needs a lock on this record.

You get better concurrency if you do this as the last action before syncpoint; that way, the lock is held for the shortest possible time. You get even better concurrency if you can maintain several counters, perhaps keyed on transaction type, or perhaps keyed on a random number, which can then be summarized when the total is required.

Reading programs

On CICS for AIX only
Programs written in C, IBM COBOL, or PL/I are mapped into storage from the file system, and so require very little paging space.

On the HP-UX and Solaris platforms
Programs written in C are mapped into storage from the file system, and so require very little paging space.

Programs written in Micro Focus Server Express COBOL are explicitly read from the file system in such a way that each application server makes its own copy the first time that application server runs that transaction program. For a system with a large number of application servers and a large number of Micro Focus Server Express COBOL transaction programs, this can significantly increase the virtual storage requirement.

Using the program cache

Caching a program improves performance because reloading costs are saved when the program is used. The following options affect program caching:

- The Region Definitions (RD) **ProgramCacheSize** attribute.

  Programs in all languages except Java and Micro Focus Server Express COBOL and Net Express are cached only if at the time a program is loaded, the Program Definitions (PD) **Resident** attribute is set to **yes** and the number of cached programs in use has not reached the maximum number.

  A resident program is not loaded again until one of the following occurs:

  – A SET PROGRAM NEWCOPY or SET PROGRAM COPY(NEWCOPY) command is issued.

    Note: On Windows systems, a file on disk cannot be removed or overwritten while a copy of it is in the cache. Therefore, new copies of programs on Windows can be used only if they are not cached.

  – The value of the **Resident** attribute is changed to **no**.

  – The cache is full and the program is removed to make room for a new entry. In this case, the least-recently used program is removed, leaving more-frequently used programs in the cache.

  Because individual programs in use are not removed from the cache, it is recommended that the cache size allow programs at every logical level to be cached when EXEC CICS LINK is used.

- The environment variable COBSW is set to l*nn*

  where *nn* is the cache size in bytes. This is a Micro Focus Server Express and Net Express environment variable that sets the length of the Micro Focus Server Express COBOL and Net Express program cache. This variable is defined in the region's environment file.

Java programs can be cached by the Java runtime controlled through the Java Virtual Machine (JVM).

Programs are loaded into the cache in the sequence in which they are accessed.

Note:

- Programs are cached for each application server, so a new copy of the program has no effect on an application server if the application server has not yet run the program.
- The SET PROGRAM NEWCOPY or SET PROGRAM COPY(NEWCOPY) commands for a Micro Focus Server Express COBOL or Net Express program remove every program previously loaded by the application server, so a fresh copy of every such program is used after one of these commands is run, not just the program for which the SET PROGRAM NEWCOPY or SET PROGRAM COPY(NEWCOPY) command is issued. These commands have no effect on Java programs.

Improving performance of database access

The following list provides some tips to help you design applications to access data files more efficiently:

- If you use non-XA databases, your applications need to issue the EXEC CICS CONNECT command to start the database server for each transaction. Such frequent connections use a lot of system resources. Applications can reuse an XA-managed database connection over and over while the application server is running without needing to call EXEC CICS CONNECT repeatedly.
- Minimize the number of interactions between a CICS transaction program and a file system such as the SFS server or the database product.

  Calls made between the CICS transaction program and the file system significantly add to the overall time it takes to complete the program. To reduce this time, you consider using dynamic SQL calls or calls to SQL stored procedures, which allow you to process several rows or several files in one activation. For instructions on how use any of these features, see the SQL documentation supplied with your database product.
- If the majority of the transactions do not need to access a database, then use a database that supports XA dynamic registration.

  If the database supports XA dynamic registration, then CICS can arrange to drive syncpoint in the database only when the database is actually updated for the given transaction. This can reduce the time it takes to syncpoint when you are running in an environment with multiple XA databases and CICS file control files. In such an environment, transactions usually update data managed by only some of the applications.
- Avoid overusing SQL operations that declare cursors or prepare dynamic SQL.

  Typically, SQL operations that declare cursors (EXEC SQL DECLARE *CursorName*) or prepare dynamic SQL commands (EXEC SQL PREPARE) are relatively expensive, compared with the SQL operations (OPEN, FETCH, EXECUTE, CLOSE). Consider using these more expensive calls only once per transaction program. A programming style such as is shown below can be more appropriate if it is supported by your database product.

```
EXEC SQL OPEN ... ;
if (SQLCODE == {cursor-not-declared} )
{
   EXEC SQL DECLARE ... ;
   EXEC SQL OPEN ... ;
}
EXEC SQL FETCH ... ;
EXEC SQL CLOSE ... ;
```

- DB2 provides a choice of two types of table spaces:
  - System Managed Space (SMS) Table Space, where operating system file manager calls are used to control the storage space.

– Database Managed Space (DMS) Table Space, where the database manager controls the storage space.

By default, DB2 uses SMS Table Space. It is recommended however, that DMS Table Space is used with CICS since, in general, it provides a performance improvement.

See the *DB2: Administration Guide* for details on how to create DMS table space. See also "Improving performance of database access" on page 45 for more information about CICS and database performance.

Improving performance of DB2 file management

A performance advantage can be gained from using DB2 single-phase commit optimization where the additional data integrity provided by the XA interface two-phase commit protocol is not required.

In general, DB2 single-phase commit optimization provides an improvement in performance over SFS file management.

Improving performance of Oracle file management

Refer to the Oracle performance tuning guide for any performance enhancements.

Using CICS with WebSphere MQ

CICS supports access to WebSphere MQ queues through XA or non-XA connections, using MQ API calls in CICS programs.

- With an XA connection to WebSphere MQ, CICS uses a two-phase commit protocol to ensure that all resources within the logical unit of work are updated together. The CICS transaction must issue an **MQCONN** call in order to access WebSphere MQ resources.
- With a non-XA connection to WebSphere MQ, the MQ API is not managed within the CICS logical unit of work, so you must use manual methods to make the two systems consistent. To do this, you must issue EXEC CICS SYNCPOINT to commit updates to CICS resources (or let this happen implicitly at the end of the transaction), and issue the relevant MQ API calls to commit updates to MQ resources.

For both XA and non-XA connections to WebSphere MQ, CICS transactions that access WebSphere MQ resources must either:

- Invoke the **MQDISC** call before returning to CICS

or

- Install the Task termination exit sample program provided by WebSphere MQ

The **MQDISC** call or the WebSphere MQ Task termination exit sample perform the necessary clean up of WebSphere MQ resources. If you install the WebSphere MQ Task termination exit sample, it is invoked by CICS during normal and abnormal task termination.

If you do not either install the WebSphere MQ Task termination exit sample, or call **MQDISC** before returning to CICS, you will experience unexpected CICS application server abends during subsequent transaction runs.

For information about the WebSphere MQ Task termination exit sample and how to install it, see the *WebSphere MQ System Administration Guide*. Refer to the *CICS Administration Guide* for information about how to configure WebSphere MQ with CICS using XA and non-XA connections, and about building CICS/MQ

applications.

Chapter 3. Programming constraints

A number of topics need to be considered when writing CICS application programs. This discusses each of the following topics in detail:
- Using COBOL compilers
- Using C and C++ compilers
- Using PL/I compilers
- Tabs in map and program sources
- The use of DCE, and operating system functions
- Names that are reserved by CICS
- Thread safety
- CICS-safe functions

General programming considerations

This section contains topics of general interest to consider when writing CICS application programs.

Tabs in map and program sources

CICS requires that map and program sources follow strict formatting rules. For example, a comma in column 72 is used to identify a continuation, and the continued line must begin in column 16.

Be aware that some editors use hidden tab characters. **cicstran** and **cicsmap** report these as errors.

The use of DCE and operating system functions

You can use functions from libraries provided by the operating system and DCE in your CICS application programs, but you should be cautious in so doing for the following reasons:

- Possible difficulties in migrating an IBM CICS for Windows or CICS on Open Systems application to run on another CICS product. These functions will need to be recoded if the other operating system does not have identical function.
- Restrictions placed on the use of these functions. The following topics discuss these functions.

Names reserved for CICS

Do not define maps, tables, or programs that begin with the letters *DFH* and do not define transactions or transient data queues that begin with the letter *C*. These are reserved for CICS internal use.

As a general rule, avoid the use of the following as names for user variables:
- EXEC
- CICS
- END-EXEC
- Or names starting with:
  - CICS
  - cics
  - DFH
  - ERZ
  - FAA

Thread safety

Compile all CICS programs by using the thread safe version of a runtime language library.

As discussed in "How CICS runs your transactions" on page 8, your application programs run in a multi-threaded process. Certain types of function may not behave correctly when called consecutively from two or more threads. These functions should either not be used, or they should only be called using a serialization technique. For example, any function that keeps static data between calls (such as **ctime**) should not be used.

On CICS on Open Systems

The operating system provides a replacement library, as an example libc_r for libc, that contains thread-safe replacements for the functions that are not thread-safe. In some cases, function names have been changed. For example, the replacement thread-safe version of ctime is called ctime_r.

On CICS for Windows

Functions should only be used from the compiler or operating system thread-safe libraries.

Note: Use only the main thread in a CICS application to perform EXEC CICS calls or do any XA work.

CICS-safe functions

Even if a function is thread safe, you cannot always use it in the CICS environment. Functions that you can use in CICS application programs without restrictions are called *CICS-safe* functions.

The functions and services below are *not* CICS safe and must not be used at all, or at best, used with caution. In many cases, CICS uses these functions and services itself; their use in application programs possibly causes CICS to behave unpredictably.

Note: The absence of a function from this list does not imply or guarantee that it is CICS safe.

Table 4. Non-CICS-safe functions

Function or Service	Restriction
Any function that is not thread safe	Do not use.
exec (without fork)	Do not use.
setlocale	Do not use.
Shared memory functions	Do not attach memory at the address specified with the Region Definitions (RD) **RegionPoolBase** attribute. CICS uses this address for region pool shared memory.
CICS internal functions	Do not use.
exit or _exit	Do not use.

Table 4. Non-CICS-safe functions (continued)

Function or Service	Restriction
fork	Refer to the DCE application programming information.
stdin, stdout, stderr	Do not use these streams.
kill	Do not send signals to any CICS process.
raise	Do not use.
assert	Do not use.
abort	Do not use.
sigprocmask	Do not use.
signals	Do not use.
cin cout cerr	Do not use these iostream objects.
DCE asynchronous cancellation	Do not use.
DCE threads	You can create threads but you must not use any CICS facility in them.
Encina TRAN	Do not use.
Encina Transactional C	Do not use.
Encina threadTid	Do not use.
catch(...) (in C++ programs)	Any exceptions not generated by the application must be rethrown (using **throw** with no argument).

See the *CICS Problem Determination Guide* for related information.

Using the COBOL compilers

Some points to consider are:

- ***The CICS API supported in COBOL***: The full CICS API, as detailed in the *CICS Application Programming Reference*, is supported for COBOL application programs.

On CICS for AIX or CICS for HP-UX

Micro Focus Server Express COBOL only

- ***The character sets supported***: Double Byte Character Set (DBCS) is supported.
- ***COMP-5***: The COMP-5 message is an *information* message, and not a *warning* message. The message is produced for the first occurrence of a COMP-5 variable found by the compiler. The message format is:

```
      02   CICS-FN-CODE          PIC S9(4) COMP-5 SYNC
**209-I*****************************************
**    COMP-5 is machine specific format. (future occurrences not
                                          indicated)
```

- ***Argument values in COBOL***: Argument values in EXEC CICS commands in COBOL programs are described in the *CICS Application Programming Reference*.
- ***Comments***: Comments are allowed in the middle of EXEC CICS commands. When these programs are translated, the comments are retained.

Default options in EXEC CICS commands for COBOL

CICS command options that have default values are shown in the following table:

Table 5. Default options in EXEC CICS commands for COBOL

Commands	Options	Defaults from
READ, READNEXT, READPREV, READQ TD, READQ TS, RETRIEVE	LENGTH	INTO
DUMP	LENGTH (unless FLENGTH is specified)	FROM
JOURNAL, REWRITE, START, WRITE, WRITEQ TD, WRITEQ TS	LENGTH	FROM
CONVERSE	FROMLENGTH (unless FROMFLENGTH is specified)	FROM
CONVERSE	TOLENGTH (unless TOFLENGTH is specified)	INTO
CONVERSE	MAXLENGTH or MAXFLENGTH with no argument	INTO
RECEIVE	LENGTH (unless FLENGTH is specified)	INTO
RECEIVE	MAXLENGTH or MAXFLENGTH with no argument	INTO
SEND	LENGTH (unless FLENGTH is specified)	FROM
LINK, XCTL, RETURN	LENGTH	COMMAREA
CONNECT PROCESS	PROCLENGTH	PROCNAME
CONNECT PROCESS	PIPLENGTH	PIPLIST
JOURNAL	PFXLENG	PREFIX
RECEIVE MAP('MAPNAME') without INTO or SET	INTO	MAP
SEND MAP('MAPNAME') without FROM or MAP ONLY	FROM	MAP

Data declarations needed in COBOL

The EXEC Interface Block (EIB) data declaration is provided automatically. The following mapping of data types in EIB fields is used:

- 16-bit binary integers are defined as PIC S9(4) COMP
- 32-bit binary integers are defined as PIC S9(8) COMP
- Character strings are defined as PIC X(*n*), where *n* is the number of bytes

The following data declarations are available in CICS-supplied COBOL copybooks:

- BMS screen attribute definitions (DFHBMSCA)
- Attention key definitions (DFHAID)

For more information about the DFHBMSCA and DFHAID COBOL copybooks, see the *CICS Application Programming Reference*.

COBOL program invocation environment (Micro Focus Server Express COBOL only) On CICS on Open Systems only

On CICS on Open Systems

CICS application programs written in COBOL language must be compiled (using **cicstcl**, for example) into either Micro Focus Server Express COBOL intermediate files (.int) or Micro Focus Server Express COBOL runtime code files (.gnt), suitable for dynamic loading by the Micro Focus Server Express COBOL run-time system. The files created are not directly executable using **cobrun**, since they contain some unresolved symbols that are expected to be provided from the invoking CICS application server (see "Application servers" on page 10). To run your transaction, CICS looks up the name and location of a CICS program, and uses the dynamic loading facility provided by the the Micro Focus Server Express COBOL run-time system to load and run the program in the CICS application server

On CICS for Windows

CICS applications written in Micro Focus Net Express COBOL can be compiled using the **cicstcl** command for use with or without Animator. The **cicstcl** command produces files with the suffix **cbmfmt**.

COBOL program invocation environment (IBM COBOL only)

CICS application programs written in IBM COBOL must be compiled (using **cicstcl** for example) into executables that are linked with thread-safe libraries. These executables must have a suffix of **ibmcob** for identification by CICS.

Calling programs from COBOL

CICS enables your COBOL programs to use the CALL statement to call other programs. With the CALL statement you can invoke a separate COBOL program whether or not it contains CICS commands or CICS dependencies.

On CICS for Windows

Dynamically called COBOL subprograms should be built as dlls and have the file suffix **.dll**.

A dynamically called COBOL program must reside in a directory that is included in the following environment variables:

- For IBM COBOL programs:

  `COBPATH`
- For Micro Focus Server Express COBOL and Net Express programs:

  `COBDIR`

Ensure that the environment variable is in the System environment for use within CICS.

COBPATH and COBDIR are defined by the compilers at installation time.

For Micro Focus Net Express COBOL programs, all COBOL calls check for the program in the PD database the first time the program is called. If there is no PD entry, the run-time looks for the program in the COBDIR directory. After the first call to this program, disabling the program has *no effect*.

On CICS on Open Systems

A COBOL program must reside on one of the following directories:

- The CICS bin directory:

  `$CICS/bin`

  Note: Refer to xii for a description of how $CICS is used to represent the product pathname.
- If it is specified as a file name or a base name in a CALL statement, it can reside in:

  `/var/cics_regions/region/bin`
- If it is specified as a full pathname in the CALL statement it can reside anywhere in the file system.

If a program that calls a CICS program is processed by the CICS translator, the CALL statement must pass the addresses of DFHEIBLK and DFHCOMMAREA as the first two parameters, whether or not any other parameters are passed. The call has the form shown in the following example:

```
CALL 'PROG' USING DFHEIBLK DFHCOMMAREA PARM1 PARM2
```

In the called program PROG, the CICS translator inserts DFHEIBLK and DFHCOMMAREA into the linkage section and into the USING list of the procedure division statement. You code the procedure division statement normally. In the previous example, this is:

```
PROCEDURE DIVISION USING PARM1 PARM2
```

The translator inserts DFHEIBLK and DFHCOMMAREA into this statement before PARM1.

Note: The CALL and PROCEDURE DIVISION statements shown above are examples only. PARM1 and PARM2 are not mandatory.

Working storage

Applications written in COBOL are given a fresh copy of their working storage when they are first used within a transaction.

It is not possible for CICS to arrange cancellation of anything but the top level CICS program. If you wish to have working storage for such programs set to the initial state both within and across transactions, you must ensure these programs are cancelled individually.

If you are using Micro Focus Net Express COBOL, every program is given new working storage if the /DATA-CONTEXT flag was specified when the program was compiled.

Recursion

Applications written in COBOL should avoid using recursion. See "COBOL PERFORM and CALL commands" on page 23 for more information.

Available memory (Micro Focus Server Express COBOL On CICS on Open Systems only)

Micro Focus Server Express COBOL allows you to set the amount of available memory for the Micro Focus Server Express COBOL run-time system (RTS) using the **-l** run-time switch. Unless all available memory has been used up, the RTS performs logical EXEC CICS CANCELs, where a logical EXEC CICS CANCEL flushes all file buffers but does not release memory. The RTS loads programs that have been logically canceled in preference to reloading from disk; the memory is used as a form of cache.

This can give performance advantages, but there is no way for CICS to force the memory to be freed. If you want to force a reload from disk in order to pick up a new copy of a program, for example, then you must set the memory switch to zero in the environment file in your region directory (for example **COBSW=-l0**) to force the run-time to reload from disk.

Alternatively, you may force a reload from disk for each separate program using CEMT SET PROGRAM(*program_name*) NEWCOPY, where *program_name* corresponds to the definition in the Program Definitions (PD).

If you experience problems running Micro Focus Server Express COBOL transactions after using NEWCOPY, try resubmitting the transaction to overcome the problem.

Mixing languages

A *run unit* is a running set of one or more programs that communicate with each other by COBOL CALL statements. It is an execution of a single entry as defined in the Program Definitions (PD). PD entries are not required for called subprograms. In a CICS environment, a run unit is entered at the start of a CICS task, or invoked by an EXEC CICS LINK or EXEC CICS XCTL command.

Be aware of three rules governing calls between COBOL and C or C++ programs under CICS.

- COBOL programs that contain CICS commands can CALL C or C++ programs as long as the called C or C++ programs do not contain any CICS commands.
- C or C++ programs that contain CICS commands can CALL COBOL programs as long as the called COBOL programs do not contain any CICS commands.
- COBOL programs can EXEC CICS LINK or EXEC CICS XCTL to a C or C++ program regardless of whether or not the C or C++ program contains CICS commands.

Therefore, if your COBOL program invokes a C or C++ program that contains CICS commands (or vice versa), use EXEC CICS LINK or EXEC CICS XCTL rather than the COBOL CALL statement.

Passing integer data between C or C++ and COBOL

If you want to pass integer data between C or C++ and COBOL programs in a COMMAREA, the data items must be decalred in COBOL as COMP-5, otherwise the byte ordering of the data is incorrect and the values are corrupted.

Returning from COBOL programs

There are various methods of returning from a COBOL program:

GOBACK
Returns control to calling program or to CICS.

EXIT PROGRAM
Returns control to the calling program when issued from a subprogram invoked by a COBOL CALL. When issued from a top level procedure division, as in a program invoked directly by CICS, or by EXEC CICS LINK or EXEC CICS XCTL, it is ignored.

STOP RUN
Must not be used in CICS programs because it causes the application server to terminate.

EXEC CICS RETURN
Terminates the program normally.

EXEC CICS XCTL
Terminates the program normally.

EXEC CICS ABEND
Terminates the program abnormally.

Releasing resources

In short-lived processes, any resources (such as file handles and memory) will be automatically released. Since CICS uses long-running application servers, this release will not be automatically done at the end of a transaction. CICS will only clean up the CICS resources that it owns (such as EXEC CICS GETMAIN storage). Therefore it is important that if a CICS program calls a function such as **open** directly, a corresponding **close** must be called before the end of the program. If **close** is not called, the system may run out of resources.

The resources affected by this consideration include memory segments, semaphores, locks and file handles.

Object-oriented COBOL support

In a CICS application program written in IBM COBOL Set for AIX, you can use the INVOKE statement to invoke methods written in IBM COBOL Set for AIX as long as the methods do not use CICS commands. You cannot use EXEC CICS LINK to access COBOL methods directly. However, you can link to another COBOL program which invokes methods directly.

Compiling EBCDIC-enabled COBOL programs

EBCDIC-to-ASCII data conversion can be necessary for communications between a CICS for Windows system and a remote mainframe or CICS/400 system. For example, if you use function shipping to access file records from a mainframe, the data must be converted from EBCDIC to ASCII in order to be usable by the ASCII program on the CICS for Windows workstation. Often, resource definition templates must be defined to identify the type of conversion to be applied to the data.

To avoid the need to set up these conversion tables or to ensure the collating sequence compatibility of mainframe applications, you can use the **cicscobinsert** utility to compile EBCDIC-enabled programs to run on a CICS for Windows workstation. Such EBCDIC-enabled programs are supported by Micro Focus Net Express version 3.0 or later.

Note: Use in EBCDIC-enabled programs of the Front-End Programming Interface (FEPI), Basic Mapping Support (BMS) macros, and pretranslated copybooks is not supported. Support of EXEC SQL calls is provided through Micro Focus Net Express.

To create an EBCDIC-enabled program, first write the source code on the workstation in ASCII or download the source code from the mainframe, using EBCDIC to ASCII conversion in the usual way. Then set the Micro Focus Net Express option CHARSET to EBCDIC, as shown in the following example:

```
set COBOPTS=/CHARSET(EBCDIC)
```

Then translate the program on the workstation. Use the **cicscobinsert** utility to provide a necessary conversion for alphanumeric literals during interaction between applications and CICS. See "Using Micro Focus Net Express to compile EBCDIC-enabled COBOL programs" on page 227 for information on how to use this utility with the **cicstcl** or **cicstran** commands.

The Micro Focus Net Express compiler converts alphanumeric literals to EBCDIC. It does not convert character constants represented in hexadecimal format. As a result, these constants must exist as their hexadecimal equivalent EBCDIC values. See your compiler documentation for more information. Programs compiled with the EBCDIC directive run as EBCDIC programs; for example, PIC 9 fields hold values from X'F0' through X'F9'.

When an application is run and an EXEC CICS call is made, character values are converted from EBCDIC to ASCII and from ASCII to EBCDIC as required. Examples of fields and arguments that are converted in this way include the following:

- EIBTRNID and EIBTRMID field values.
- Arguments to EXEC CICS commands.

The content of data areas passed to EXEC CICS commands is not converted.

Restrictions

You should not use COBOL statements to request operating system functions that can be requested from the CICS API.

You should also not use the following COBOL statements:
- STOP RUN
- DISPLAY

If the PROGRAM-ID statement is used, the program name must be the same as the filename.

The PROGRAM-ID is used by the COBOL compilers as the entry-point name, defaulting to upper-case.

On CICS on Open Systems you can use compiler options to allow a mixed-case entry-point name, but CICS does not support mixed-case for Micro Focus Server Express COBOL, and supports mixed case for IBM COBOL only if you compile and link in separate steps and do not use **cicstcl**.

On CICS for Windows, CICS does not support mixed-case entry-point names for IBM COBOL or Micro Focus Net Express COBOL.

Micro Focus Net Express programs are never unloaded from memory.

See the *CICS Application Programming Reference* for related information.

Using the C and the C++ compilers

If you are writing CICS application programs in C and C++, you must use a compiler that can link with thread-safe libraries (see "Thread safety" on page 50). It is recommended that you use **cicstcl** to compile your application programs because this CICS command uses the thread-safe compilers.

Note: For C and C++ transaction programs, the main() function must be contained in a **.ccs** file because it must be translated.

The full CICS API, as detailed in the *CICS Application Programming Reference* , is supported for C and C++ application programs, with the exception of those commands related to nonstructured exception handling:
- EXEC CICS HANDLE CONDITION (with or without a label)
- EXEC CICS HANDLE AID (with or without a label)
- EXEC CICS IGNORE CONDITION
- EXEC CICS PUSH HANDLE
- EXEC CICS POP HANDLE
- EXEC CICS HANDLE ABEND (label)

Use of these commands is diagnosed by the translator.

In a C and C++ application, every EXEC CICS command is treated as if it has the NOHANDLE option specified. This option overrides the default action when an un-handled condition occurs, which is to map the condition to a transaction abend, and means that control always flows to the next instruction after an EXEC CICS command. The application must test for a normal response and take appropriate action if any condition has occurred.

EXEC CICS HANDLE ABEND PROGRAM commands are allowed, but you cannot use EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE.

Argument values in C and C++

Argument values in EXEC CICS commands in C and C++ programs are described in the *CICS Application Programming Reference*.

When CICS requires a pointer argument for data-area or pointer-ref argument types, the translator will insert the ampersand (&) if you have not already specified it.

New time arguments are supplied for use with C and C++ programs that cannot have a packed decimal data type. The commands affected are:

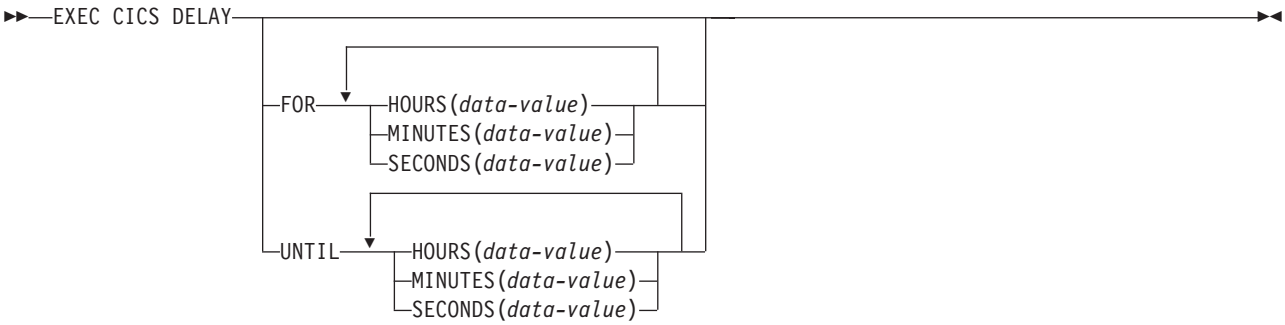
- EXEC CICS START
- EXEC CICS DELAY

These time fields have previously been defined as data-type *hhmmss* - packed decimal. These fields are of two kinds:

1. An elapsed time interval
2. A time-of-day value, relative to the preceding midnight.

Delay processing the task (EXEC CICS DELAY)

DELAY



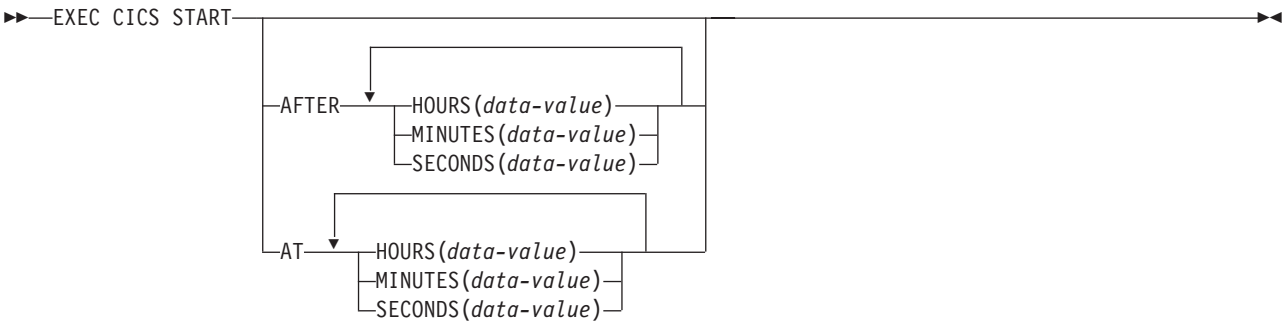
FOR has the same meaning as the INTERVAL option used with COBOL programs (interval of time). UNTIL has the same meaning as the TIME option (absolute time).

The following is an example of the EXEC CICS DELAY command:

```
EXEC CICS DELAY UNTIL HOURS(2) MINUTES(15) SECONDS(37);
```

Start a task (EXEC CICS START)

START



AFTER has the same meaning as the INTERVAL option (interval of time). AT has the same meaning as the TIME option (absolute time).

Time arguments

HOURS(hh)
The option specifies the hours time argument. The argument (hh) is defined as a 32 bit binary data value.

MINUTES(mm)
The option specifies the minutes time argument. The argument (mm) is defined as a 32 bit binary data value.

SECONDS(ss)
The option specifies the seconds time argument. The argument (ss) is defined as a 32 bit binary data value.

If more than one of the time components (HOURS, MINUTES, SECONDS) is specified, the omitted ones will be assumed to have a zero value and the permitted ranges will be:
1. Hours - 0 to 99
2. Minutes - 0 to 59
3. Seconds - 0 to 59

If only one time component (HOURS, MINUTES, SECONDS) is specified, the permitted range will be 0 to the equivalent of 99 hours 59 minutes and 59 seconds.

Defaulting options in CICS commands

The LENGTH option must always be supplied in CICS commands in C or C++ language application programs, except on EXEC CICS SEND MAP, EXEC CICS RECEIVE MAP, EXEC CICS READ, EXEC CICS READNEXT, EXEC CICS READPREV, EXEC CICS REWRITE, and EXEC CICS WRITE.

You can omit the FROM option on the EXEC CICS SEND MAP command and the INTO option on an EXEC RECEIVE MAP command, provided the MAP option is a literal string. The defaults are shown in the following table:

Table 6. EXEC CICS default options for C and C++

Commands	Option	Defaults to
EXEC CICS SEND MAP('MAP1')	FROM	&map1.map1o

Table 6. EXEC CICS default options for C and C++ (continued)

Commands	Option	Defaults to
EXEC CICS RECEIVE MAP('MAP1')	INTO	&map1.map1i

Data declarations needed in C and C++

The EXEC Interface Block (EIB) data declaration is provided automatically. Within a C application program, fields in the EIB are referred to in lowercase and fully qualified as, for example, *dfheiptr->eibtrnid*, in contrast to EIBTRNID as used in other CICS applications.

The following mapping of data types in EIB fields is used:
- 8-bit unsigned binary integers are defined as cics_ubyte_t
- 16-bit unsigned binary integers are defined as cics_ushort_t
- 16-bit signed binary integers are defined as cics_sshort_t
- 32-bit unsigned binary integers are defined as cics_ulong_t
- 32-bit signed binary integers are defined as cics_slong_t
- Single character fields are defined as cics_char_t
- Character strings are defined as cics_char_t arrays
- Boolean data is defined as cics_bool_t

The following data declarations are available in CICS-supplied C or C++ include files:
- BMS screen attribute definitions (dfhbmsca.h)
- Attention key definitions (dfhaid.h)

For more information about the dfhbmsca.h and dfhaid.h header files, see the *CICS Application Programming Reference*.

C and C++ program invocation environment

Two arguments are normally passed to a C or C++ main procedure: *argc* and *argv*. The *argc* argument is the number of command line arguments with which the program was invoked, and the *argv* argument is a pointer to an array of character strings that contain the command-line arguments, one per string. CICS programs do not have genuine command-line arguments; instead, the following scheme is used:

```
argc     4
argv[0]  The transaction id under which the program is running
argv[1]  A pointer to the EIB
argv[2]  A pointer to the commarea (or NULL)
argv[3]  The commarea length
```

Note that this scheme differs from other approaches used by the CICS family where only the *transaction identifier (tranid)* is provided and the value for *argc* is set to **1**.

When you link-edit an application program to be run under CICS (using **cicstcl**, for example), the object you create is not directly executable by the operating system both because it contains some unresolved symbols and because it expects to be run by the CICS application server. The CICS application server (see "Application servers" on page 10) provides a complete environment for running the loadable objects produced by **cicstcl**. To run your transaction, CICS looks up the name and location of a CICS program, and uses the operating system dynamic

loading facility to load that program into the application server. The unresolved symbols in the program are then resolved by symbols provided by the server and the program begins execution.

On CICS for AIX only

If you are compiling a program that uses non ANSI extensions, compile using the extended mode. Also, the same options apply to **cicstcl**. If you are compiling a transaction program that uses non-ANSI extensions, add the following to your **cicstcl** arguments:

```
CCFLAGS=-qlanglvl=extended
```

IBM C++ executables must have a suffix of **.ibmcpp** for identification by CICS.

Restriction in cached programs using variables in static storage

When C or C++ programs are cached, variables in static storage are not reinitialized when the program is re-invoked. This is useful where database connections (among other things) are stored as variables in static storage and are required to be kept between successive invocations as this provides a large performance benefit.

BMS maps may be stored as static, which allows previous data to be shown on the screen. If you do not want this because of the possiblity of security exposures, either change the code so that the variables in static storage are not used, or do not cache the program.

EXEC CICS address COMMAREA

The address of the communication area is not passed as an argument to a C or C++ main function. This means that C functions must use EXEC CICS ADDRESS COMMAREA to obtain the address of the communications area.

Calling programs from C or C++

CICS C or C++ programs are only given a fresh copy of their static data for the first instance of the program. You should not use static data in programs that are called recursively.

Mixing languages

A run unit in CICS C or C++ is a single CICS C or C++ program. A CICS transaction can consist of many run units, each of which can be in a different language, provided that programs written in one language communicate with programs written in another language only by using EXEC CICS LINK or EXEC CICS XCTL commands.

EXEC CICS address EIB

The ADDRESS of the exec interface block (EIB) is not passed as an argument to a C or C++ main function. This means that C functions must use EXEC CICS ADDRESS EIB to obtain the address of the EIB.

If you specify the RESP option or the RESP2 option or both in an EXEC CICS statement, you must specify EXEC CICS ADDRESS EIB(`dfheiptr`) beforehand. The name `dfheiptr` is required. Omitting this step causes the transaction to abend with an abend code of ASRA.

Releasing resources

In short-lived operating system processes, any resources (such as file handles and memory) will be automatically released. Since CICS uses long-running application servers, this release will not be automatically done at the end of a transaction. CICS will only clean up the CICS resources that it owns (such as EXEC CICS GETMAIN storage). Therefore it is important that if a CICS program calls a function such as **open** directly, a corresponding **close** must be called before the end of the program. If **close** is not called, the system may run out of resources.

The resources affected by this consideration include memory segments, semaphores, locks and file handles.

String handling

CICS does not generally support the C convention of delimiting strings using a null byte. Instead, strings are padded with spaces to their maximum length. All strings output from CICS are always space padded, and therefore cannot be used directly by C string manipulation functions.

You should take care in passing strings terminated with a null to the CICS interface. This will work in situations where the option concerned is not of a fixed length, but will not work for fixed length options. For example, a CICS file name can be up to eight characters in length, so you can open the literal “ABCD” and CICS will handle this correctly. But, temporary storage queue names (including REQID on certain commands) must be exactly 8 bytes. Therefore, CICS will take whatever bytes happen to be after the null following “ABCD” as part of the TSQ name.

C++ considerations

Do not use iostream objects in CICS programs.

Do not place CICS statements in header files as part of an inline function definition. CICS statements in class templates must be translated before the inclusion of the template class definition in any program. C++ allows the creation of new objects on the heap and it is important that these objects are deleted after use to avoid memory leaks. Ensure that all initialization is done within the object constructors.

You should also ensure that your *prodDir* is set to the path of any class libraries used in your CICS environment file.

Do not use CICS statements in static object constructors or destructors. Static constructors and destructors are called outside a CICS Logical Unit of Work, and so using CICS statements at these points can cause abends.

See the *CICS Application Programming Reference* for related information.

Returning from C and C++ programs

There are various methods of returning from a C or C++ program:

return Returns control to calling program or to CICS.

exit, abort, and _exit
These must not be used in CICS programs because they cause the application server to terminate.

EXEC CICS RETURN
Terminates the program normally.

EXEC CICS XCTL
Terminates the program normally.

EXEC CICS ABEND
Terminates the program abnormally.

Using the IBM PL/I compiler

CICS application programs written in IBM PL/I must be compiled (using **cicstcl** for example) into executables that are linked with thread-safe libraries. These executables must have a suffix of **ibmpli** for identification by CICS.

The full CICS API, as detailed in the *CICS Application Programming Reference* , is supported for PL/I application programs, with the exception of HANDLE ABEND LABEL.

Restriction in cached programs using variables in static storage

When PL/I programs are cached, variables in static storage are not reinitialized when the program is re-invoked. This is the desired behavior as database connections (among other things) are stored as variables in static storage and it is required that these to be kept between successive invocations as this provides a large performance benefit.

BMS maps may be stored as statics and this may allow previous data to be shown on the screen. If this behavior is not desired, such as the possibility of security exposures, either the code should be changed so that variables in static storage are not used, or the program should not be cached.

Default options in EXEC CICS commands for PL/I

CICS command options that have default values are shown in the following table:

Table 7. Default options in EXEC CICS commands for PL/I

Commands	Options	Defaults from
READ, READNEXT, READPREV, READQ TD, READQ TS, RETRIEVE	LENGTH	INTO
DUMP	LENGTH (unless FLENGTH is specified)	FROM
JOURNAL, REWRITE, START, WRITE, WRITEQ TD, WRITEQ TS	LENGTH	FROM
CONVERSE	FROMLENGTH (unless FROMFLENGTH is specified)	FROM
CONVERSE	TOLENGTH (unless TOFLENGTH is specified)	INTO

Table 7. Default options in EXEC CICS commands for PL/I (continued)

Commands	Options	Defaults from
CONVERSE	MAXLENGTH or MAXFLENGTH with no argument	INTO
RECEIVE	LENGTH (unless FLENGTH is specified)	INTO
RECEIVE	MAXLENGTH or MAXFLENGTH with no argument	INTO
SEND	LENGTH (unless FLENGTH is specified)	FROM
LINK, XCTL, RETURN	LENGTH	COMMAREA
CONNECT PROCESS	PROCLENGTH	PROCNAME
CONNECT PROCESS	PIPLENGTH	PIPLIST
JOURNAL	PFXLENG	PREFIX
RECEIVE MAP('MAPNAME') without INTO or SET	INTO	MAP
SEND MAP('MAPNAME') without FROM or MAP ONLY	FROM	MAP

PL/I program invocation environment

When you link-edit an application program to be run under CICS (using **cicstcl**, for example), the object you create is not directly executable by the operating system both because it contains some unresolved symbols and because it expects to be run by the CICS application server. The CICS application server (see "Application servers" on page 10) provides a complete environment for running the loadable objects produced by **cicstcl**. To run your transaction, CICS looks up the name and location of a CICS program, and uses the operating system dynamic loading facility to load that program into the application server. The unresolved symbols in the program are then resolved by symbols provided by the server and the program begins execution.

PL/I executables must have a suffix of .ibmpli.

Calling programs from PL/I

CICS PL/I programs are only given a fresh copy of their static data for the first instance of the program. You should not use static data in programs that are called recursively.

Data declarations needed for PL/I

The EXEC Interface Block (EIB) data declaration is provided automatically. The following mapping of PL/I expressions in EIB fields is used:

- Halfword binary values are defined as FIXED BIN(15)
- Fullword binary values are defined as FIXED BIN(31)
- Character strings are defined as CHAR(*n*), where *n* is the number of bytes

For more information, see the *CICS Application Programming Reference*.

OPTIONS(MAIN) specification

If OPTIONS(MAIN) is specified for a PL/I program, the program can be the first program of a transaction, or control can be passed to it by means of an EXEC CICS LINK or XCTL command.

If OPTIONS(MAIN) is not specified, the program cannot be the first program in a transaction, nor have control passed to it by an EXEC CICS LINK or XCTL command.

The definition of the EIB (DFHEIBLK) is generated in each program, based upon the pointer variable DFHEIPTR. In programs declared with OPTIONS(MAIN), the DFHEIPTR variable is set up to address the EIB on entry. In programs other than those declared with OPTIONS(MAIN), addressability to the EIB is the user's responsibility.

Addressability can be achieved by using the command:

```
EXEC CICS ADDRESS EIB(DFHEIPTR)
```

Addressability can also be achieved by passing the EIB address or the values of particular EIB fields as arguments to the CALL statement that invokes the external procedure.

Mixing languages

A run unit in CICS PL/I is a single CICS PL/I program. A CICS transaction can consist of many run units, each of which can be in a different language, provided that programs written in one language communicate with programs written in another language only by using EXEC CICS LINK or EXEC CICS XCTL commands.

Returning from PL/I programs

There are various methods of returning from a PL/I program:

EXEC CICS RETURN
: Terminates the program normally.

EXEC CICS XCTL
: Terminates the program normally.

EXEC CICS ABEND
: Terminates the program abnormally.

Releasing resources

In short-lived operating system processes, any resources (such as file handles and memory) will be automatically released. Since CICS uses long-running application servers, this release will not be automatically done at the end of a transaction. CICS will only clean up the CICS resources that it owns (such as EXEC CICS GETMAIN storage). Therefore it is important that if a CICS program calls a function such as "open" directly, a corresponding "close" must be called before the end of the program. If "close" is not called, the system may run out of resources.

The resources affected by this consideration include memory segments, semaphores, locks and file handles.

See the *CICS Application Programming Reference* for related information.

Chapter 4. Coding for presentation services

This chapter describes how to write application programs that use the CICS presentation services.

What are the presentation services?

CICS provides two API services that handle the presentation interface with the end-user. These can be thought of as a *low-level interface* and a *high-level interface*:

Terminal services
The low-level interface is called *terminal services*. This interface allows you to communicate with the terminal using 3270 datastreams. See "Terminal services."

Basic mapping support (BMS)
The high-level interface is *Basic Mapping Support (BMS)*. This facility allows you to define your presentation logic in terms of maps containing fields. You define these maps externally to the applications program with BMS macros. On CICS for Windows and CICS on Open Systems, BMS macros are coded manually. See "Basic mapping support (BMS) services" on page 71 and "Using the BMS macros to code BMS map sets" on page 97.

Terminal services

The CICS terminal services allow user-written application programs, terminals, and logical units to communicate using API commands. Terminal services handle data translation, transaction identification, synchronization of input and output operations, and the session control needed to read from or write to a terminal. This frees the application from the responsibility of controlling terminals.

During processing, an application program is connected to one terminal for one task. The terminal services monitor which task is associated with which terminal, and determine which task is to be initiated.

In intercommunications, terminal services are also used to control communication with *logical units (LU)* or with another CICS system. An LU can represent a terminal either directly, or indirectly, through a program stored in a subsystem controller.

CICS for Windows supports terminals with a process known as a *CICS local terminal*. The client process is responsible for reading user input, communicating with an attached region to run transactions, receiving output from those transactions, and displaying it to the end-user. The supplied CICS client **cicsterm** (CICS on Open Systems) and **cicsteld** processes contain an IBM 3270 Information Display System datastream emulation which supports a subset of the 3270 protocol.

CICS on Open Systems clients, **cicsterm**, and **cicsteld** however, cannot emulate all I/O devices, such as card readers. To enable you to use these devices from CICS, and to allow you to write enhanced 3270 emulations, the CICS on Open Systems clients **cicsterm** and **cicsteld** processes are replaceable. Information about CICS on

Open Systems clients **cicsterm** and **cicsteld,** and about writing your own versions of the replaceable code is in the *CICS Administration Reference*.

You can use terminal services to communicate with a remote system by means of distributed transaction processing (DTP), which is described fully in the *CICS Intercommunication Guide*.

You can use the following API commands to use the terminal services (provided they apply to your terminal or logical unit):

RECEIVE
Read data from a terminal or logical unit.

SEND Write data to a terminal or logical unit.

SEND TEXT
Write text to a terminal or logical unit that is formatted in such as way that words are not split across lines.

CONVERSE
Converse with a terminal or logical unit.

WAIT TERMINAL
Synchronize terminal input/output for a transaction.

ISSUE SIGNAL
Send an asynchronous interrupt.

Other services available in response to terminal services commands apply to specific types of terminal. Because CICS supports many different terminal types, it provides a number of special services. In particular, there are many commands for communicating with display devices such as those members of the IBM 3270 Information Display System family.

Use the EXEC CICS SEND TEXT command to send text to a terminal. The text is split into lines of the same width as the terminal, such that words are not broken across line boundaries. If the text exceeds a page, it is split into pages that fit on the terminal.

The data area containing the text to be sent is specified in the FROM option. The LENGTH option specifies the length of this area. To help control the format of the display, the text may contain embedded new-line characters and embedded blanks.

How text is formatted

When formatting the text, BMS splits it into lines of length less than or equal to the terminal page width. BMS pads the ends of lines with blanks rather than splitting words. BMS starts each line with a single blank corresponding to the 3270 attribute byte. On a 3270, the attribute byte is set to unprotected, autoskip, and normal intensity.

If a line of text ends with a non-blank character, and the next character is a blank, BMS processes the data as if it were a sentence, that is, the blank is removed and the next character is positioned in the second column of the next line, which is the starting column for text.

Where a line of text ends with a blank and the next character is also a blank, BMS honors all blanks to process the data as if it were in table format.

If the FROM data area contains more text than can fit on a single screen, BMS creates another screen of formatted text and overwrites the previous screen. If the ERASE option is used, then the previous screen is erased before the new screen is written, thus improving the presentation of the text.

Printing the text

CICS for Windows:

See the *CICS Clients: Administration* for information about using CICS on Open Systems client services to print the text.

CICS on Open Systems:

Printed text is formatted such that words are not broken across line boundaries and, if the text exceeds a page, it is split into pages. To print text:

1. Start **cicstermp**, the CICS printer emulator, as shown:

   ```
   cicstermp -r region -n netname
   ```

   where *netname* is the network name specified in the Terminal Definitions (WD) entry that represents the printer. Refer to the *CICS Administration Reference* for additional information, such as how to specify a print command when **cicstermp** is started.
2. Use either EXEC CICS START or CECI START, as shown:

   ```
   EXEC CICS START TRANSID(transid) TERMID(termid) END-EXEC.

   CECI START TRANSID(transid) TERMID(termid)
   ```

   where *transid* is the application that issues the SEND TEXT PRINT command (see Figure 3 on page 70), and *termid* is the printer model entry defined in the WD. If **cicstermp** is running, these commands send the text to the printer specified by the principal facility named with the TERMID(*termid*) option. If **cicstermp** is not running, then the START command is queued until **cicstermp** is started.

```
01 TEXT-LENGTH PIC S9(4) COMP-4     VALUE +792.
*
01 PRINT-LINE.
    05  FILLER              PIC X(40)            VALUE
        'ONCE-UPON A TIME, THERE WERE THREE BEARS'.
    05  FILLER              PIC X(40)            VALUE
        ', DADDY BEAR, MUMMY BEAR, AND LITTLE BAB'.
    05  FILLER              PIC X(40)            VALUE
        'Y BEAR. ONE DAY, EARLY IN THE MORNING BE'.
    05  FILLER              PIC X(40)            VALUE
        'FORE BREAKFAST, THEY ALL WENT OUT FOR A '.
    05  FILLER              PIC X(40)            VALUE
        'WALK, LEAVING THE PORRIDGE TO COOL ON TH'.
    05  FILLER              PIC X(40)            VALUE
        'E KITCHEN TABLE. WHILE THEY WERE AWAY, G'.
    05  FILLER              PIC X(40)            VALUE
        'OLDILOCKS CAME UPON THE HOUSE WHILE SHE '.
    05  FILLER              PIC X(40)            VALUE
        'WAS SKIPPING THROUGH THE FOREST AND, UPO'.
    05  FILLER              PIC X(40)            VALUE
        'N SEEING THE PORRIDGE SITTING INVITINGLY'.
    05  FILLER              PIC X(40)            VALUE
        ' ON THE TABLE, DECIDED THAT SHE WAS HUNG'.
    05  FILLER              PIC X(40)            VALUE
        'RY AND PROMPTLY ATE EVERY BOWL OF PORRID'.
    05  FILLER              PIC X(40)            VALUE
        'GE IN SIGHT. FEELING TIRED, SHE THEN RET'.
    05  FILLER              PIC X(40)            VALUE
        'IRED TO THE BEDROOM AND FELL ASLEEP IN T'.
    05  FILLER              PIC X(40)            VALUE
        'HE BABY BEARS BED. WELL, YOU CAN IMAGINE'.
    05  FILLER              PIC X(40)            VALUE
        ' THE COMMOTION WHEN THE BEARS GOT BACK. '.
    05  FILLER              PIC X(40)            VALUE
        'IT WAS ALL SORTED OUT, THOUGH, WHEN THEY'.
    05  FILLER              PIC X(40)            VALUE
        ' WENT UPSTAIRS AND FOUND GOLDILOCKS - TH'.
    05  FILLER              PIC X(40)            VALUE
        'EY DECIDED THAT SHE''D TASTE MUCH BETTER '.
    05  FILLER              PIC X(40)            VALUE
        'THAN THE PORRIDGE ANYWAY SO THEY ATE HER'.
    05  FILLER              PIC X(32)            VALUE
        ' WITH SOME FRIED WILD MUSHROOMS.'.

 LINKAGE SECTION.
 01  DFHCOMMAREA                  PIC S9(4) COMP.
 PROCEDURE DIVISION.
 EXEC CICS SEND TEXT FROM(PRINT-LINE)
    LENGTH(TEXT-LENGTH)
    FREEKB
    PRINT
    END-EXEC.
```

Figure 3. COBOL example using SEND TEXT to print data

The output will read as shown in Figure 4 on page 71 (in 80 column mode):

```
ONCE UPON A TIME, THERE WERE THREE BEARS, DADDY BEAR, MUMMY BEAR, AND LITTLE
BABY BEAR. ONE DAY, EARLY IN THE MORNING BEFORE BREAKFAST, THEY ALL WENT OUT
FOR A WALK, LEAVING THE PORRIDGE TO COOL ON THE KITCHEN TABLE. WHILE THEY WERE
AWAY, GOLDILOCKS CAME UPON THE HOUSE WHILE SHE WAS SKIPPING THROUGH THE FOREST
AND, UPON SEEING THE PORRIDGE SITTING INVITINGLY ON THE TABLE, DECIDED THAT SHE
WAS HUNGRY AND PROMPTLY ATE EVERY BOWL OF PORRIDGE IN SIGHT. FEELING TIRED, SHE
THEN RETIRED TO THE BEDROOM AND FELL ASLEEP IN THE BABY BEARS BED. WELL, YOU
CAN IMAGINE THE COMMOTION WHEN THE BEARS GOT BACK. IT WAS ALL SORTED OUT,
THOUGH, WHEN THEY WENT UPSTAIRS AND FOUND GOLDILOCKS - THEY DECIDED THAT SHE'D
TASTE MUCH BETTER THAN THE PORRIDGE ANYWAY SO THEY ATE HER WITH SOME FRIED WILD
MUSHROOMS.
```

Figure 4. Example output from SEND TEXT

Terminal services design considerations

To ensure the efficiency of applications using terminal services, you should consider the following guidelines:

- *Keep the length of the datastream short*.

  Good screen design and effective use of the 3270 features, significantly affect the number of bytes to be sent in the remote procedure call (RPC) and potentially therefore across the network.

  It is particularly important to keep the number of terminal transmissions as small as possible as, in most cases, this may be the slowest part of the path a transaction takes. The efficiency of the datastream therefore affects both response time and line usage.

- *Use CONVERSE rather than SEND/RECEIVE*.

  If you program it to be conversational, use CONVERSE rather than SEND/RECEIVE (or a (SEND WAIT)/ RECEIVE sequence). They are functionally equivalent, but CONVERSE only crosses the CICS services interface once, which saves processor time.

- *Avoid using unnecessary transactions*.

  For example, avoid situations that may cause users to enter an invalid transaction or to use the CLEAR key unnecessarily, thus adding to terminal input, task control processing, terminal output, and overhead. Good screen design and standardized PF/PA key assignments should minimize this.

See the *CICS Application Programming Reference* for related information.

Basic mapping support (BMS) services

Basic mapping support (BMS) is an interface between CICS and CICS application programs that move 3270 datastreams to and from a terminal. BMS formats input and output display data in response to BMS commands and programs using device information defined in the Terminal Definitions (WD) and formatting information from the maps prepared for the application program.

On both TXSeries CICS you can code map definition macros and use the **cicsmap** command to generate the maps. This is described in "Using the BMS macros to code BMS map sets" on page 97.

Developing applications that use BMS services

BMS provides a front-end interface to 3270 protocol. Because of this, you don't need to know the details of 3270 protocol. However, because some 3270 protocol

terminology is used in the following discussions, it would be helpful to refer to the *IBM 3270 Information Display Programmer's Reference*.

BMS lets you separate the tasks of display design and CICS application programming by interpreting generalized *device-independent* application program output commands, and by generating *device-dependent* datastreams for specific output devices. BMS also transforms incoming datastreams into a form acceptable to application programs. BMS determines the format of the datastream for the terminal from the device on which the task is running, not from the application program.

You can use the same BMS input or output commands in your application program for different types of devices. A single BMS command in your program applies equally to various devices because BMS interprets commands differently for different device types.

BMS commands are quite simple, because all the low-level formatting information is held separately, in *maps*. Consequently your application programs are easier to write and less affected by changes to the system or its devices. You can make changes independently of your application programs just by changing the maps.

BMS functions supported in CICS

The following table shows the functions supported with minimum function BMS in TXSeries CICS:

Table 8. Functions supported with minimum function BMS

BMS function	**Is the function provided?**
Basic 3270 displays and printers	Yes
Default and alternate screen sizes	Yes
Extended attributes	Yes
Formfeed control	Yes
Command-level requests	Yes
Non ACCUM SEND MAP TERMINAL	No
RECEIVE MAP and RECEIVE MAP FROM	Yes
map set suffixing	Yes
GDDM coordination	No
Aligned and unaligned maps	No
Out of sequence input maps	Yes
Block data	Yes
Automatic setting of WCC character line width	Yes
ERASE, ERASEUP, FORMFEED, CURSOR, and WCC on BMS SENDs	Yes

In addition to minimum function support, TXSeries CICS also support the use of the EXEC CICS SEND TEXT command with the following options:

- FROM(data-area)
- LENGTH
- CURSOR
- FORMFEED
- ERASE
- PRINT

- FREEKB
- ALARM
- NLEOM

SEND TEXT is included in standard function BMS support.

The existing maps you migrate from other CICS family members are processed within the TXSeries CICS environment with the limitation that minimum function BMS and SEND TEXT with the limited options is supported. This supports the IBM 3270 and IBM 3270-like range of displays and printers, except for *SNA character string* printers. (*SNA character string* refers to, in SNA, a character string composed of EBCDIC controls, optionally intermixed with end-user data, that is carried within a request or response unit.)

How BMS affects programming

Different versions of a display map can exploit the features of different devices. By having the screen data in *fields* (that is, defining data as having *field* format), you can address predefined fields in a display symbolically by name from within your application, without knowing the actual screen positions of those fields.

Changing field data to and from its displayable form is called *mapping*.

Although the same fields must appear in all versions of a display, you can move them around in different versions. A suffixing mechanism enables BMS to associate a display version with the kind of device to which it applies.

The BMS processor

The BMS processor accepts files containing a series of BMS macro instructions that were coded in a map source file. The input file name must have extension .bms. The following describes the map source files and associated input files.

BMS maps: Maps specify to BMS how field data is to be formatted. Maps are not needed for text data. Every BMS mapping command names a map that contains formatting instructions. Each map has two forms: physical and symbolic.

BMS formats a display for a given device by embedding control characters in the datastream. A *physical map* tells BMS how to do this.

A *symbolic map* is a source language data structure that is used to resolve source program references to fields in the map. It is also known as a *logical map*.

Note: C language symbolic maps must be byte-packed.

Maps must belong to a map set. You usually group related maps together into one map set. You define a map set by coding a series of BMS macro instructions. BMS maps are generated by the BMS map processor from BMS source files containing three types of macros. The first of these macros defines the map set itself, the second defines the first or only map, the last defines fields within those maps. The field macros define the field size, shape, position (the row and column), potential content, and characteristics (such as protected or unprotected, and bright or dark).

You define map sets, maps, and fields within maps with the following macros:

DFHMSD
: Defines a group of related maps, known as a map set.

DFHMDI
: Defines a single map within a map set.

DFHMDF
Defines a single field within a map.

To map screen display information into the application program, you use the EXEC CICS RECEIVE MAP command. In order to send data from an application program to a display screen or printer you use the EXEC CICS SEND MAP command. You use the EXEC CICS SEND CONTROL command to transmit device control orders.

See the *CICS Application Programming Reference* for related information.

CICS also provides the EXEC CICS HANDLE AID command. This command passes control within an application program as a result of an *attention identifier (AID)* being received from a display device.

On TXSeries CICS the **cicsmap** command processes a source file containing BMS macros, and generates either a symbolic map or maps, or a physical map or maps, or both as specified by the map input. You use command line options to control the generation of symbolic or physical maps. See "cicsmap - generate BMS map files" on page 276.

The map sets required in your region are defined in a Program Definitions (PD) entry as described in the *CICS Administration Reference*.

For the following descriptions, the term *processed* means *processed with the **cicsmap** command*, and the term *defined* means *defined in a Program Definitions (PD) entry*.

Symbolic Map
A symbolic description map set definition is processed and defined in the subdirectory. The member name is usually the same as the map set name, but it need not be. Alternatively, the symbolic map can be copied or inserted directly into the application program.

Physical Map
A physical map set definition is processed and defined in the CICS region's bin subdirectory.

When you define the physical map, you should consider whether to add a suffix to its name. See "Map set suffixing" for more detail. The reason for suffixing a map is that you might wish to produce alternative versions of it for different emulator models.

Note: The **cicsmap** command takes no notice of DFHMSD TYPE=*operand*. Symbolic and physical maps are generated depending on the options supplied to the processes.

Map set suffixing: If you want to execute the same transaction from more than one type of emulator, you might need to use BMS map set suffixing. If you are prepared to use the same map to format data for all your emulators, you need not read the rest of this section. If however, you wish to organize output data according to the emulator in use, making best use of its features, you ought to consider suffixing map sets.

To avoid problems at the assembly stage, do one of the following:

- Use SUFFIX or TERM on your DFHMSD maps (in which case, you can safely use the same name for your map set and your maps).
- Make sure you use different names on your DFHMSD and DFHMDI macros.

When a BMS command requests a mapping operation for an 80-column terminal, CICS adds a suffixed 'M' to the map set name specified in the command, and attempts to load a map set with that suffixed name. For example, if the terminal is 80 columns wide, and the map set name is MYMAP, CICS will use a map set with the name of MYMAPM. If MYMAPM is not found, MYMAP is used. Terminals that are 132 columns wide do not use a suffixed map set.

Finally, you should ask your system programmer to ensure that your physical maps are defined with the correct suffixes. In particular, you may need to know the following points about suffixing:

- If you specify TERM or SUFFIX on your DFHMSD macro, you should ensure that the physical map set is defined using the correctly suffixed name.
- You can code SUFFIX, instead of TERM, on DFHMSD if you need to create a special version of a map.
- You should ensure that the Region Definitions (RD) **SufficesSupported** attribute is set to **yes** to ensure suffixed maps are loaded at run time.

The physical mapset name is made up from the mapset name on the DFHMSD macro suffixed by a 1-character value determined from the TERM or SUFFIX operand of the macro.

For example:

```
TESTMAP DFHMSD MODE=INOUT,CTRL=(FREEKB,FRSET),TERM=3270,               *
               LANG=COBOL,TIOAPFX=YES,EXTATT=MAPONLY,COLOR=BLUE
```

In this case, M is appended because TERM=3270 is specified. For further information about the DFHMSD macro, see the *CICS Application Programming Reference*.

In this way, if you generate a base mapset TESTMAP as well as a suffixed mapset TESTMAPM when you issue a mapping command such as EXEC CICS SEND MAP(...) MAPSET(TESTMAP), CICS will use the map from the unsuffixed mapset unless the DEVICE TYPE is a 3270 when it uses the map from the suffixed mapset. In this way you can modify your maps within the different mapsets to suit different device types.

How to use the BMS processor: The BMS processor accepts map source files containing a series of BMS macro instructions. You can create these BMS macro instructions by:

- Migrating them from another CICS family member
- Typing the macros in with a standard editor

The source file consists of a map set which in turn is broken into a number of maps and fields. You declare the map set using a map set macro, you declare each map using the map macro, and you declare the fields using the field definition macros. The BMS processor processes the macros to produce the symbolic and physical map files.

The symbolic map file is a programming source language data structure (for CICS a COBOL DATA DIVISION definition or a C structure or a PL/I Structure) which you include in your application program. The symbolic map allows you to make symbolic references to display fields and attributes. CICS loads the physical map file into the runtime environment, and uses this map file to generate the display control data to drive a particular display device.

You invoke the BMS processor directly from the command-line, by entering the required options for the **cicsmap** command. For example:

```
cicsmap mapset.bms
```

where *mapset* is the one to seven character map set name.

You can prefix the input map source file name with an optional pathname, but the file must have the extension .bms on it. The BMS processor places the output physical map file in the current working directory, overwriting any previous physical map file for the same map source file. The output file is named:

```
.map
```

If you have used the SUFFIX or TERM option, the output file is named:

```
.map
```

where *x* is replaced with the value given in the SUFFIX and TERM operands in the map source file.

The BMS processor places the symbolic map file in the following for COBOL:

in the following for C or C++:

```
mapset.h
```

and in the following for PL/I :

```
mapset.inc
```

The symbolic map file is placed in the current working directory, overwriting any previous symbolic map file for the same map set. No maps are generated if the BMS processor detects any errors in the map source file.

The contents of the map source file alter the operation of the **cicsmap** program. The LANG option you associate with the map set macro (DFHMSD) determines the output of **cicsmap**. The LANG option does not affect the production of the physical map file.

To generate the symbolic map file where the target language is COBOL, set the option LANG in the map set macro in the map source file equal to 'COBOL'. You can use the output file as a COBOL copybook file using the COBOL verb COPY.

To generate the symbolic map file where the target language is C, set the option LANG in the map set macro in the map source file equal to 'C'. You can use the output file as an include file.

To generate the symbolic map file where the target language is PL/I, set the option LANG in the map set macro in the map source file equal to 'PLI'. You can use the output file as a PL/I include file.

3270 terminal emulation

The 3270 datastream conveys both displayable data characters and nondisplayable control characters between the host processor and an emulator. Using BMS commands, you do not have to understand the format of the datastream. Nevertheless, you need to know the range of things the datastream allows you to do. This section describes the features of 3270 emulators, and discusses how you can use them.

Input operations: When you have typed data on to a display, you will probably want to send it to the host processor. You do this by:
- Pressing the ENTER key
- Pressing a program function (PF) key

Although the display will send modified data when you press PF keys, the keys are not normally used for this. Generally, you assign a specified meaning to the key itself.

If you want to send data from a terminal without the user having to enter it explicitly, you can set the modified data tag (MDT) for the required field in the output to the terminal.

An attention identifier (AID) character is always sent to the host processor whenever a 3270 input operation is performed. This indicates the cause of the input operation.

CICS ensures that an application program receives input data intended for it. The AID allows the application program to react differently, depending on the input operation. The effect of different combinations of data and AIDs depends entirely upon the design of the application program.

Output operations: An emulator can receive data from an application program, as well as send data to it. Some of the data can be displayed, the rest consists of device controls. By building datastreams containing device controls, you can, for example:
- Unlock the keyboard for input
- Reset the modified data tag (the MDT) of each field
- Erase all unprotected fields
- Position the cursor

The way you use these features is up to you. However, they can improve the usability of your application program.

Display field concepts: An application program can divide a screen into more than one field. The fields combine to produce a complete screen of data.

A field starts with an attribute character, continues with data characters, and ends at the next attribute character. A field can contain only a single character or it can span several lines, as the last character on a line is logically followed by the first character on the next line.

If the screen width is the same as the map width, BMS allows a field to wrap around from the end of one line to the start of the next. Because of the dependence on resource definitions, it is not recommended that an application design should depend on this function.

Normally, a display is divided into several fields by the program, but it is possible to have a display with no fields (no attribute characters). This occurs when you press the CLEAR key; such unformatted displays are not supported by BMS and the use of the CLEAR key causes a MAPFAIL in BMS.

An application program can use the HANDLE AID command to detect the use of the CLEAR key. An application programmer can use the HANDLE CONDITION

command to detect a MAPFAIL condition. (See the *CICS Application Programming Reference* for related information.) An attempt to read from a cleared screen raises the MAPFAIL condition.

Attribute character: The attribute character is always the first character of a field. It occupies a character position on the screen but appears as a blank.

Attribute characters can convey the following field attributes:

Unprotected
You can enter any keyboard character into an unprotected field.

Numeric-only
A numeric-only field is unprotected. On a data entry keyboard, a numeric-only field causes a numeric shift to occur.

Protected
Data cannot be entered in a protected field. If the operator attempts to enter data, the keyboard is locked. Stopper fields following variable-length data fields are normally defined with protected attribute characters. If the operator attempts to enter more characters than the variable-length data field can contain, the stopper field following it will cause the keyboard to be locked.

Autoskip
An autoskip field is a protected field that automatically skips the cursor to the next unprotected field. Keyword fields and stopper fields following fixed-length data fields are normally defined with autoskip attribute characters.

Normal intensity
A normal intensity field displays the data at the normal operating intensity.

Bright intensity
A bright intensity field displays the data at a brighter than normal intensity. This is often used to highlight keywords, errors, or operator messages.

Nondisplay
A nondisplay field does not display the data on the screen for operator viewing and does not print the field data. This might be used to enter security data when the screen is visible to others. This attribute characteristic should be used with care, as the operator loses the ability to verify the data entered in a nondisplay field. This field might also be used to store messages on the screen. The messages can be displayed later by changing the attribute character to bright or normal intensity.

Base color
A **base color** image is produced by using the PROTECT and INTENSIFY attributes of the 3270 standard datastream to select four colors:
white bright, protected
red bright, unprotected
blue normal, protected
green normal, unprotected

The protect attribute retains its protect function when conveying color information. This characteristic only applies to color monitors, and if you have not set the color attributes for the field in BMS, the display defaults to a standard setup which varies from terminal to terminal (for example, green for normal intensity, white for bright).

Extended color

Extended color attributes in an extended data stream determine the colors of display elements. The datastream can specify the colors of multicharacter fields. Seven colors can be selected: blue, red, pink, green, turquoise, yellow, and neutral.

As soon as an extended color attribute is received, the display treats the whole image as an extended color image. Fields that have no color attribute adopt the default colors (green for normal intensity, white for bright). If the color control switch has been set to base color, the part of the image that has already been displayed will change from base color to default color. Such a change, which could disturb an operator, can be avoided by applying an extended color attribute to the first field in any image that uses extended color.

The device interprets extended color attributes to determine the colors of fields in an image.

Extended highlighting

Extended highlighting can be applied to characters, or character fields, in a display that uses the extended data stream. It can take one of three forms: BLINK, REVERSE, or UNDERSCORE.

Modified data tag (MDT)

The modified data tag is turned on when fields are modified by the operator. When the operator presses the ENTER key or a PF key, only fields that have been modified by the operator or selected by the cursor select are transmitted to the processor. The program may send fields to the terminal with the modified data tag already on to guarantee that the field will be returned with the next inbound transmission.

Insert-cursor indicator

The insert-cursor indicator is not a field attribute. Instead, it places the cursor under the first data character of the field. If the insert-cursor indicator is specified for more than one field, the cursor is placed under the first data character of the last field specified.

Not all devices support all the attributes. BMS ensures that attributes which are not supported by the device (as specified in the emulator definition or determined by an automatic query of the device following logon) are ignored when building the datastream.

Note: The unprotected, protected, and autoskip characteristics of the attribute character are mutually exclusive. Only one may be selected for each field. The normal, bright, and nondisplay characteristics of the attribute character are mutually exclusive. Only one may be selected for each field.

Programmed symbols: As well as the standard display symbol sets, some devices can have optional additional symbol store. Support for this feature is limited in CICS for AIXand IBM CICS for Windows to selection of the default Double-Byte Character Set for the device (PS=8). This feature uses the extended data stream.

Field Outlining: Field outlining allows lines to be included above, below, to the left, or to the right of a field. You can use these lines in any combination to construct boxes around fields or groups of fields.

Screen layout design

CICS provides an emulation of the features of the IBM 3270 Information Display System. These features allow you to design screen layouts for operator convenience and efficiency. The success of an online system depends on its ease-of-use, screen clarity, and terminal operator acceptance.

The following features of some IBM 3270 Information Display System screens make it easier for the layout designer to fulfil these requirements:

- Color
- Field highlighting
- Programmed symbols
- Easy correction
- Field delimiters or stoppers (to control the length of data entered)

The first step in designing 3270 screen layouts is to divide the screen into functional areas such as a title area, an application data area, and a message area.

The CICS local terminals on Windows, **cicslterm**, the CICS 3270 Terminal Emulator, and the CICS on Open Systems client on Open Systems, **cicsterm**, support the following screen sizes:

- 80 columns by 24 rows
- 80 columns by 32 rows
- 80 columns by 43 rows
- 132 columns by 27 rows

Title area: The title area of a screen should identify the program that displays the data. Data fields from the same file can appear in the same screen locations for different applications, permitting the operator to become familiar with fields by their screen location. You can use a title to help the operator recognize the application. The title area is normally the top one or two lines of the screen and may contain a page number (if you require more than one page), field headings, and other data besides the title.

Application data area: The application data area comprises the main portion of the screen. Data from one or more records in the same file or multiple files is entered by you, or displayed for you, depending on the application requirements.

Three kinds of field are usually found in this area: keyword, data, and stopper.

Keyword fields

Contain constant data sent by the program to identify the contents of a data field. For example, a keyword field containing:

```
ACCOUNT BALANCE:
```

can precede and identify a data field containing:

```
$129.54.
```

A keyword field can also be used in a data entry application to identify the data being entered. For example:

```
ENTER QUANTITY:
```

Data fields

Contain data that the application program retrieves and displays. The data may appear exactly as stored in a file, or it may be changed by the application program. Data fields may also be left blank for the user to enter data. The application program can use the entered data to make

changes to a record or to alter the processing of the program. In some cases, it may be appropriate for the program to display characters in an entry data field to guide the operator in entering the data. For example:

```
DATE: MMDDYY
```

means enter month, day, year, each having two characters.

Stopper fields
On data entry screens restrict the length of the data fields. Stopper fields containing no data are used to define the space between data fields and to stop the operator from entering too many characters in a field.

For example, a field containing a street address may be 20 characters long, but for screen layout reasons the screen layout designer provides an entire line of the display for this field. To prevent you from keying more than 20 characters on this line, the layout designer defines a stopper field starting in the twenty-first position of the line. The stopper field is protected, restricting the operator to the 20-character field.

Terminating reverse video: If you have a field with reverse video on, and you want to delimit the characteristics of the field before it reaches to the next field, you can define a stopper field in between.

For example, on CICS on Open Systems suppose you want a map to look like this, where both fields are in reverse video and there is nothing in between:

```
Hello World

Bye World
```

You define the source bms map as follows:

```
MAP1    DFHMSD TYPE=&SYSPARM,MODE=INOUT,LANG=COBOL,STORAGE=AUTO,       X
               TIOAPFX=YES
MAP1    DFHMDI SIZE=(30,60),MAPATTS=(COLOR,HILIGHT),DSATTS=(COLOR,     X
               HILIGHT),LINE=1,COLUMN=1,COLOR=GREEN,HILIGHT=REVERSE
        DFHMDF POS=(6,10),LENGTH=11,ATTRB=(UNPROT,NORM),CASE=MIXED,    X
               INITIAL='Hello World'
        DFHMDF POS=(6,22),ATTRB=(ASKIP,NORM),LENGTH=0
        DFHMDF POS=(10,10),LENGTH=9,ATTRB=(UNPROT,NORM),CASE=MIXED,    X
               INITIAL='Bye World'
        DFHMDF POS=(10,20),ATTRB=(ASKIP,NORM),LENGTH=0
MAP1    DFHMSD TYPE=FINAL
```

Message area: You use the message area of a screen to send instructions or messages to assist the user in processing a transaction. You should separate the message area from the application data area to allow communication with the user, without disturbing the application data. The message area is normally the bottom one or two lines of the screen.

See the *CICS Application Programming Reference* for related information.

Using BMS services in application programs

Application programs use EXEC CICS SEND and EXEC CICS RECEIVE commands to send and receive data. The following describes the syntax of these and other commands and demonstrates their use.

Symbolic map data structures

The symbolic map data structures that result from executing map and field definition macros contain extended versions of the fields, each one consisting of

subfields. Each subfield can be referred to by its name, which is the name assigned to the field, plus a single-letter suffix. Each kind of subfield has a different suffix.

Furthermore, the whole input or output data structure can be addressed by its suffixed name. The suffixed name of an input map is its original name extended by the suffix I. The corresponding suffix for the output map is O.

Input map data structures: The suffixes used to address subfields, and the contents of those subfields, in input maps are:

Table 9. Suffixes used for input map data structures

F	A flag byte. This is normally set to X'00'. If the field has been modified but no data is sent (that is, the field is cleared) the flag byte is set to X'80'.
I	Input data read from the display. It is set to X'00' if no data is entered for that field.
L	A 16-bit binary length value. This defines the number of characters that are typed into the data field before it is read by BMS.

Input field suffixes: Having read data, a program can process it by issuing ordinary application programming commands that address fields by name.

Consider a field, called INPUT, in an input map. A program can test that either its length field INPUTL contains a value greater than zero (data has been entered) or that its flag byte INPUTF indicates that the field has been cleared. If INPUTL contains a value greater than zero, you can, for example, move the first INPUTL characters from INPUTI to another data area.

The suffix on the data structure for the whole map enables you to manipulate the whole data structure. For example, you can write simple commands to copy the whole structure into another data area.

Output map data structures: The suffixes used to address subfields, and the contents of those subfields, in output maps are:

Table 10. Suffixes used for output map data structures

A	An attribute byte defining the characteristics of the field (for example, protected or unprotected).
C	An attribute byte specifying the color of the field.
E	(C only). This suffix is appended to the field name to derive a structure which occurs *n* times (where *n* is specified by the OCCURS operand of the DFHMDF macro).
H	An attribute byte defining the highlighting to be used within a field in a display.
M	An attribute byte defining that *SO/SI* creation is to be used. SO/SI (shift in/shift out) refers to code extension characters that are used to substitute graphic characters in standard character sets.
O	Output data to be sent to the display. The program usually stores data in such a field before sending the map. If the contents of the field begin with a null character, the whole field is ignored, the contents of the display field being taken from the physical map. If you want to send a blank field, you must store blanks in the symbolic map data structure. Being non-null, this overrides the contents of the physical map.
P	An attribute byte defining the programmed symbol set to be used within a field in a display.

Table 10. Suffixes used for output map data structures (continued)

U	An attribute byte defining the outline to be used.

If you want to use programmed symbols, you must ensure that a suitable symbol set has been sent to the device.

If MODE=INOUT is specified, the "fieldnameA" subfield is defined in the input map data structure. (If you are using COBOL, you will find that compiler errors occur if a MOVE statement modifying an attribute byte is qualified to refer to the output map.)

Subfields with suffixes H, P, C, U, and M are only generated if the corresponding attribute types are included in the DSATTS operand of the DFHMDI or DFHMSD macros.

As with input data fields, a program can address individual subfields in an output field, verifying or changing their contents. For example, an application program can check a calculated data value, say BALANCE. If the value is found to be negative, the highlight attribute constant (BALANCEH) in a field called BALANCE can be set to produce highlighted characters when displayed. The data value in the field occupies subfield BALANCEO.

You can also manipulate the whole output data structure using its suffixed name. For example, you can copy data into it from another area.

Note: You must set this area to nulls before using its corresponding physical map in an output operation. Otherwise, you can obtain unpredictable results. By doing this, you ensure that fields and attributes in the output display inherit the default contents of the physical map, not whatever happens to be in the symbolic data structure.

The following examples shows how you might do this in COBOL, C and PL/I:

```
COBOL example:    MOVE LOW-VALUES TO MAPO

C example:        memset(&mapo,0x0,sizeof(mapo));
PL/I example:
DCL STR BASED CHAR(32767);
 ...
SUBSTR(ADDR(MAPO)->STR,1,STG(MAPO)) = LOW(STG(MAPO));
```

Attribute constants: Subfield suffixing allows an application program to change the data within a data structure. However, the bit patterns representing particular attributes are difficult to remember, so CICS provides a list of named standard attribute bytes. You can code these names in a program instead of their hexadecimal equivalents. To use them, you must copy the list (a supplied copybook stored in the system source library at installation) into your program, using the name DFHBMSCA. For information about the attribute constants and their meanings, see the *CICS Application Programming Reference*.

Using attribute constants and subfield suffixing, a program can modify field attributes using simple commands. The following examples show how you could (1) put data into an output data field and (2) set the highlighting attribute of the output data field:

```
COBOL example:
       MOVE CUSTNO TO ACCOUNTO.... (1)
       MOVE DFHBLINK TO ACCOUNTH.. (2)
```

```
C example:
       accounto=CUSTNO;.......... (1)
       accounth=DFHBLINK;......... (2)
PL/I example:
       accounto=CUSTNO;.......... (1)
       accounth=DFHBLINK;......... (2)
```

Refer to the *IBM 3270 Information Display Programmer's Reference* for information about determining the value of an attribute constant.

Invalid data: BMS does not check the validity of attribute and data values in the symbolic data structure. However BMS does ensure that attributes are not sent to emulators that do not support them. Invalid data may be transmitted to the emulator. Some emulators can detect this invalid data and send error information to CICS.

Sending data to a display: You use the EXEC CICS SEND MAP command to send mapped data to a display. (See the *CICS Application Programming Reference* for related information.) You can send three kinds of data, depending on what options you specify, as follows:

- **Constant display data** (with attributes) such as headings, footings, prompt fields, and comments.
- **Variable display data** (with attributes) such as user data or warning messages.
- **Device control data** such as instructions to clear the screen, or activate an alarm, before displaying data.

The MAP option names the map that is used to format the data, and the MAPSET option names the map set to which the map belongs.

If the MAPSET option is omitted in an EXEC CICS SEND MAP command, the name in the MAP option is taken as the map set name.

In its simplest form, the EXEC CICS SEND MAP command is used as follows:

1. The application program assigns values to variables named in the symbolic description map.
2. The program issues an EXEC CICS SEND MAP command. This uses the application data in the application data structure to replace default data and attributes in the physical map, and sends the modified map to the display.

For example, if the first map in a map set called DISPLAY is an output map of the same name, the map can be displayed using the command:

```
SEND MAP('DISPLAY')
```

However, the omission of the MAPSET option in an EXEC CICS SEND MAP command is not recommended.

Another map, called ERROR, in the same map set can be displayed by:

```
SEND MAP('ERROR') MAPSET('DISPLAY')
```

By default, BMS displays application data or attribute data from the application data structure rather than default data from the physical map. To override this for a given field, your program must set the corresponding subfield in the data structure to hexadecimal zeros (X'00'). before issuing an EXEC CICS SEND MAP command, or use the MAPONLY option.

Composite displays: If your program sends a succession of maps to a display, the final form of the display depends on both the design of the maps, and the form of the EXEC CICS SEND MAP command. For example, if the final map fills the screen, or the EXEC CICS SEND MAP command includes the ERASE option, it obliterates all previous output. However, if you design your maps to occupy different parts of the screen, or to overlay each other only partially, you can combine them to produce the final display.

Refreshing and modifying displays: You use the MAPONLY option of the EXEC CICS SEND MAP command to build a display using data from the physical map, without inserting user data. This can be useful when sending a menu to a display, as no data is sent with the map, and input data fields regain their default data values (perhaps blank).

You use the DATAONLY option to modify the variable data in a display that has already been created by an EXEC CICS SEND MAP command. BMS transmits variable data but no physical map data.

No data is sent for fields that you have cleared to hexadecimal zeros (X'00'). You can use EXEC CICS SEND MAP DATAONLY to ensure that only changed fields are sent.

Getting storage for a data structure: You have now seen how to map data from one or more data structures. Depending on how you define your map sets, a program might have to issue commands to acquire main storage for the data structures it uses. It does this by issuing EXEC CICS GETMAIN commands. You can usually avoid having to code EXEC CICS GETMAIN commands by coding STORAGE=AUTO on the DFHMSD macro.

It has been assumed so far in this information that every output map has its own data structure. However, you might decide that this uses too much storage. To save storage, you can specify that different maps are to use the same storage area. You do this by coding BASE=name (or nothing at all), instead of STORAGE=AUTO, on the DFHMSD macro. This section describes what happens when you code each operand for each language, and how it affects application programs. For information about using the BASE operand, see the *CICS Application Programming Reference*.

Remember that, however you acquire storage, you should clear its contents before issuing an EXEC CICS SEND MAP command. If you do not do this, existing data in storage can modify the output display unpredictably. If you use EXEC CICS GETMAIN to acquire storage, you can clear the storage by coding the INITIMG option. Refer to Table 11 on page 86.

Table 11. Rules for acquiring storage

The rules for COBOL are:	When STORAGE=AUTO, the data structure must be copied into the working-storage section. CICS acquires storage automatically for every map; you do not have to code an EXEC CICS GETMAIN command. When BASE=name, the map set must be copied into the linkage section. You must code an EXEC CICS GETMAIN command to acquire enough main storage to contain the largest map in the set. When nothing is specified, or when the map set is copied into the working-storage section, you don't have to code an EXEC CICS GETMAIN command, but you should place the largest map first in the set. If the map set is copied into the linkage section, you must code an EXEC CICS GETMAIN command to get storage for it. When you use EXEC CICS GETMAIN to get main storage for a COBOL map, you must ensure that you establish addressability for the map.
The rules for C or C++ are:	STORAGE=AUTO specifies that the symbolic description maps are to be declared as having the AUTOMATIC storage class. If STORAGE=AUTO is not specified, they are declared as pointers. You cannot specify both BASE=name and STORAGE=AUTO for the same map set. If STORAGE=AUTO is specified and TIOAPFX is not, TIOAPFX=YES is assumed. *TIOAPFX* is the 12-byte terminal input/output area (TIOA) prefix. When BASE=name, you must code an EXEC CICS GETMAIN command that gets at least enough main storage to contain the largest symbolic map in the map sets sharing this base. The name specified in the BASE operand is used as the name of the pointer variable on which the symbolic description map is based.
The rules for PL/I are:	When STORAGE=AUTO, CICS acquires storage automatically for every map; you do not have to code an EXEC CICS GETMAIN command. When BASE=name, you must code an EXEC CICS GETMAIN command that gets at least enough main storage to contain the largest symbolic map in the map sets sharing this base. The name specified in the BASE operand is used as the name of the pointer variable on which the symbolic description map is based. If you omit this operand, the default name (BMSMAPBR) is used for the pointer variable. You must establish addressability for the based structures. When nothing is specified you must code an EXEC CICS GETMAIN command that sets the pointer BMSMAPBR to the address of the acquired data area. The EXEC CICS GETMAIN command must get at least enough storage to contain the largest symbolic map in the map sets.

Alternative data structures: The examples so far have shown EXEC CICS SEND MAP commands that contain literal field map names. If the field map name referenced by your program is to be a variable, you need to code the additional options, FROM and LENGTH, on the EXEC CICS SEND MAP command. Also you can use your own data area rather than the data structures from the symbolic

description map, even when use you use a literal map name. FROM enables you to display data stored in a data area other than the data structure for the symbolic description map.

FROM and MAPONLY are mutually exclusive.

LENGTH specifies the length of the data string stored in the FROM data area. You must specify the LENGTH option if the data to be mapped is shorter than the data area expected by the map.

Device control options: As well as transmitting application data to a display, BMS can relay device control commands. An application program uses options of the EXEC CICS SEND command to specify which controls are to be activated. Alternatively, it can use the BMS EXEC CICS SEND CONTROL command, which transmits device control commands without also sending application data. For example:

```
SEND MAP('ERROR') MAPSET('DISPLAY')
ERASE
```

erases the screen before data is displayed.

You can code one or more of the following device control options in an EXEC CICS SEND MAP command; they are shown in Table 12.

Table 12. EXEC CICS SEND MAP options

ALARM	Sound audible alarm on displaying data.
CURSOR	Specify position of cursor after output. The cursor position is a 16-bit binary value, representing the absolute screen address of the cursor. However, you need not always specify a value.
ERASE	Erase screen and place cursor in top left-hand corner of screen before output. The first EXEC CICS SEND MAP command of any CICS application program should specify ERASE. This ensures that the size of the screen is set to default.
ERASEAUP	Erase all unprotected fields before output.
FORMFEED	Send a form feed character as the first character in the device-dependent datastream.
FREEKB	Unlock the keyboard for data input.
FRSET	Reset all modified data tags (to “not modified” state) before output.
PRINT	Start printing (when emulator is a printer).

The CICS local terminals, **cicslterm**, The CICS 3270 Terminal Emulator, and **cicsterm** support the following screen sizes:

- 80 columns by 24 rows
- 80 columns by 32 rows
- 80 columns by 43 rows
- 132 columns by 27 rows

Cursor positioning: You can control the positioning of the display cursor in three different ways, as described in Table 13 on page 88.

Table 13. Rules for cursor positioning

Normal cursor positioning	You can specify a two-byte cursor position on the BMS EXEC CICS SEND commands. This enables you to specify the absolute value of the cursor position on the screen after the SEND has been performed. Note that the first location on the display screen is address zero. You specify the address in parentheses after the CURSOR keyword, as follows: `CURSOR(44)`
Insert cursor attribute	If you omit the CURSOR option, BMS will search the map for a field with the IC attribute. (You would have given it this attribute by coding ATTRB=IC on the DFHMDF macro for the field.) If there is more than one field with the IC attribute, BMS places the cursor at the beginning of the last one. If there is no such field, BMS places the cursor at the cursor position from the map, which is screen address zero. If you omit the CURSOR option from the EXEC CICS SEND CONTROL command, the cursor position remains unchanged.
Symbolic cursor positioning	You can use symbolic cursor positioning instead of coding an explicit value on the CURSOR option of the EXEC CICS SEND MAP command. To do this: 1. Specify MODE=INOUT in the DFHMSD macro. 2. Set the length of the field (to which the cursor is to be positioned) to -1. 3. Execute the SEND command, specifying CURSOR without an argument. CICS then places the cursor under the first data byte in the field on the output screen. If the length of more than one field is set to -1, the cursor is placed at the beginning of the first of those fields. If you use symbolic cursor positioning with EXEC CICS SEND CONTROL, the cursor is always positioned at position zero of the panel.

Accessing data outside the program: Sometimes your program needs access to information held by CICS. The ASSIGN command allows it such access.

Some ASSIGN options apply exclusively to BMS. For information about these ASSIGN options, see the *CICS Application Programming Reference*.

However, you can only use the ASSIGN options that are concerned with the position or size of the maps. Those EXEC CICS ASSIGN options are shown in Table 14.

Table 14. EXEC CICS ASSIGN options

MAPLINE	Requests the number of the line, on a display, that contains the origin of the most recently sent map.
MAPCOLUMN	Requests the number of the column, on a display, that contains the origin of the most recently sent map.
MAPWIDTH	Returns the width of the most recently sent map.
MAPHEIGHT	Returns the height of the most recently sent map.

Receiving data from a display: You use the EXEC CICS RECEIVE MAP command to receive data from a display. The data from the display is mapped into a data area in an application program.

For information about the full syntax of the EXEC CICS RECEIVE MAP command, see the *CICS Application Programming Reference* . The MAP option names the map that is used to convert the data to its unformatted form, and the MAPSET option names the map set to which the map belongs. The effect of omitting the MAPSET option is the same as explained for a EXEC CICS SEND MAP command.

For example, in its simplest form, the EXEC CICS RECEIVE MAP command is coded as:

```
RECEIVE MAP('DISPLAY')
```

This command tells BMS to map the input data into a symbolic map data structure called DISPLAY. The example assumes that the name of the map set is also DISPLAY.

Another map, MENU, in the same map set can be read by:

```
RECEIVE MAP('MENU') MAPSET('DISPLAY')
```

This command tells BMS to map the input data into a symbolic map data structure called MENU.

After an EXEC CICS RECEIVE MAP command, your program can determine the inbound cursor position by inspecting the value stored in EIBCPOSN. To do this, the application program must be informed of the physical layout of the screen, although BMS separates the screen layout from the application for other interfaces.

Refer to the following list:

- Receiving data into an alternative data structure

  The sample EXEC CICS RECEIVE MAP commands shown above use a literal for the name of the map or map set. You can also use a variable for these names, in which case you must use one of the options INTO or SET.

  If you code INTO, display data is mapped into the named data area rather than into the data structure for the symbolic description.

  If you code SET, BMS acquires a data area for you, maps the display data into it, and stores the address of the data area in the named pointer reference. Note that this data area includes the 12-byte *terminal input/output area (TIOA)* prefix, if present. (The TIOA prefix is present when TIOAPFX=YES is coded with the DFHMSD macro.) This option specifies that BMS should include a filler in the symbolic description maps to allow for the unused TIOA prefix that occurs with command-level application programs. If this operand is not specified, the BMS processor issues a warning message and assumes TIOAPFX=YES. For application portability, however, you should always code it. Refer also to the EXEC CICS GETMAIN STORAGE=AUTO description in “Getting storage for a data structure” on page 85.

  BMS sets the receiving area to hexadecimal zeros (X'00') before performing the EXEC CICS RECEIVE operation, so you should save any data in this area before performing an EXEC CICS RECEIVE operation. Furthermore, if you depend on BMS to set a data area to hexadecimal zeros (X'00') for you during an EXEC CICS RECEIVE operation, you should be aware of the MAPFAIL condition. If this arises, BMS does not set the input map to hexadecimal zeros (X'00').

If an operator types into a BMS input map, but does not fill one of the fields, BMS justifies the input data, and pads the empty part of the field according to predefined rules. These depend upon what you specify with the JUSTIFY operand of the DFHMDF macro.

The MAPFAIL condition can occur unexpectedly after an EXEC CICS RECEIVE MAP command. For example, it occurs if the emulator operator presses a program access key (such as PA1 or PA2) when CICS is waiting to perform an EXEC CICS RECEIVE command. Therefore, you should always consider using the RESP option and inspecting the returned code, or coding an EXEC CICS HANDLE CONDITION command for the MAPFAIL condition.

- Uppercase translation

  By default, the data to be mapped is assumed to come from an emulator. The emulator control table entry for the terminal can specify that all input data is to be translated to uppercase. You can override this for any individual EXEC CICS RECEIVE command by specifying ASIS. Note, however, that ASIS has no effect on the first EXEC CICS RECEIVE MAP command of a transaction. (This means that ASIS is irrelevant to pseudoconversational transactions, which issue only one EXEC CICS RECEIVE MAP command.)

- Mapping data from another data area

  Sometimes, you need to perform an input mapping operation in two stages; accepting and storing the input data in one stage, mapping it in the second. For example, your program might receive (but not map) data using an emulator control EXEC CICS RECEIVE command. It would then have to map the data from CICS storage.

  You use the FROM and LENGTH options of the EXEC CICS RECEIVE MAP command to specify that data is to be mapped from a data area instead of from an emulator. FROM names the data area; LENGTH indicates the number of bytes of data to be mapped. If the data is produced by an emulator control EXEC CICS RECEIVE command, the LENGTH value of the EXEC CICS RECEIVE MAP command must match that specified in the original EXEC CICS RECEIVE command.

  After an EXEC CICS RECEIVE MAP, the program can determine the type of attention identifier (AID) by inspecting EIBAID. You cannot issue the EXEC CICS RECEIVE MAP command in a task not associated with an emulator because BMS needs to refer to emulator information to analyze the datastream.

  For information about the emulator control EXEC CICS RECEIVE command, see the *CICS Application Programming Reference*.

  Note: The data obtained from an EXEC CICS RECEIVE BUFFER command cannot be mapped since the data will not contain SBA (set buffer address) orders and a MAPFAIL condition will be raised.

Responding to emulator input: Some operator actions cause an AID to be sent to CICS. Each such action generates a different AID. The AID is a one-byte character, and can be tested by an application program by inspecting the contents of the EIBAID field and comparing it to the values supplied in the DFHAID copybook. This can be used as a mechanism for controlling program flow. The EXEC CICS HANDLE AID command controls conditional branching caused by attention identifiers. If either the RESP, RESP2, or no EXEC CICS HANDLE option has been specified, the HANDLE AID function is suspended for that command.

- Exception conditions

  On input, you are only likely to encounter a MAPFAIL exception condition when using minimum function BMS, as follows:

- If the data to be mapped has a length of zero. This happens if a PA key or the CLEAR key is pressed.
- If an AID has been pressed and no data has been entered and no fields have FRSET.

You should remember, however, that an exception condition is not necessarily an error condition. Sometimes you might even want to treat an exception condition as part of the normal course of events. Use the RESP option or the EXEC CICS HANDLE CONDITION command to respond to exception conditions.

For information about the BMS commands, and the default system action they invoke, refer to the list of conditions documented with each command. The commands are documented in the *CICS Application Programming Reference*.

- EIBAID field

  A program can examine the value of the EIBAID field in the EIB to find out which attention key has been pressed. The 3270 emulator transmits an AID character, which is stored in field EIBAID. The program can compare the contents of EIBAID with the constants supplied in the CICS copybook DFHAID. Using EIBAID is particularly suited to a structured programming environment. For information about DFHAID, see the *CICS Application Programming Reference*.

- HANDLE AID command

  Instead of examining the contents of EIBAID, you can use the EXEC CICS HANDLE AID command to pass control to a specified label when CICS receives an AID from a display device; control is passed after the input operation is completed. In the absence of an EXEC CICS HANDLE AID for an AID, control returns to the application program at the point immediately following the input request.

  You can suspend the EXEC CICS HANDLE AID command using the PUSH and POP commands. Note that RESP (which invokes NOHANDLE) suspends the EXEC CICS HANDLE AID function in the same way as it does with EXEC CICS HANDLE CONDITION, and is better suited to a structured programming environment.

  An EXEC CICS HANDLE AID command takes precedence over an EXEC CICS HANDLE CONDITION command, unless the exception condition stops receipt of the AID. If an AID is received during an input operation for which a EXEC CICS HANDLE AID is active, control passes to the label specified in the EXEC CICS HANDLE AID command, regardless of any exception conditions that occur.

  An EXEC CICS HANDLE AID command for a specified AID remains active until the task is terminated or until another EXEC CICS HANDLE AID is issued for that AID. (If no label is specified in the new request, the existing EXEC CICS HANDLE AID command is suspended.)

  An EXEC CICS HANDLE AID command is valid only for the program in which it is issued. Each new program in a task starts without any active EXEC CICS HANDLE AID settings. When control returns to a program from a program at a lower logical level, the EXEC CICS HANDLE AID commands that were active in the higher-level program before control was transferred from it are reactivated, and any EXEC CICS HANDLE AID commands activated in the lower-level program are deactivated.

  If an AID covered by the general option ANYKEY is received and there is no active EXEC CICS HANDLE AID command for the specified AID but there is an active EXEC CICS HANDLE AID ANYKEY command, control passes to the label specified in this command. An EXEC CICS HANDLE AID command for an AID overrides the EXEC CICS HANDLE AID ANYKEY command in relation to that AID.

The following example shows an EXEC CICS HANDLE AID command that specifies one label (LAB1) for the PA1 key AID, a second label (LAB2) for the PA2 and PA3 key attention identifiers, all of the PF key attention identifiers except PF10, and the CLEAR key AID:

```
HANDLE AID
PA1(LAB1)
ANYKEY(LAB2)
PF10
```

You cannot code more than 16 options in a single EXEC CICS HANDLE AID command.

Copying symbolic description maps into your application program

In "Developing applications that use BMS services" on page 71, we described how to process and define the symbolic version of a map set. The defined version of a map set (the symbolic storage definition) is an application data structure that must be copied into any application program that refers to fields in its maps.

The following examples show you how to copy these structures for each programming language. In these examples, *mapset1*, *mapset2*, and *mapset3* (COBOL examples), *mapset1.h*, *mapset2.h*, and *mapset3.h* (C examples), and *mapset1.inc*, *mapset2.inc*, and *mapset3.inc* (PL/I examples) are the names of the files that contain the BMS symbolic map set definition. These files are generated by the **cicsmap** command.

- A COBOL program must contain a COBOL COPY statement for each symbolic map set definition. Generally, you should code the COPY statements in the working-storage section of a program. This saves you from having to acquire storage for them:

  ```
  WORKING-STORAGE SECTION.
  COPY mapset1.
  COPY mapset2.
  COPY mapset3.
  .
  .
  ```

  where *mapset* is the one to seven character name of the map set.
- A C or C++ program must contain an #include statement for each symbolic storage definition:

  ```
  #include mapset1.h
  #include mapset2.h
  #include mapset3.h
  .
  .
  ```

  where *mapset* is the one to seven character name of the map set.
- A PL/I program must contain a %INCLUDE statement for each symbolic storage definition:

  ```
  %include mapset1;
  %include mapset2;
  %include mapset3;
  ```

where *mapset* is the one to seven character name of the map set.

How to obtain BMS printed output

CICS for Windows

To obtain printed output from CICS, use **cicsprnt**, which is provided by the IBM CICS Universal Client. Applications need to know the *termid* of the printer, so that the application program can direct the output to **cicsprnt**. For more information about **cicsprnt** see the *CICS Clients: Administration*.

CICS on Open Systems

To obtain printed output from CICS, attach a **cicstermp** to a printer. Applications need to know the *termid* of the printer, so that your application program can direct the output to the **cicstermp** process.

Note: TXSeries CICS do not support the CICS local copy key.

Very often you will want printed output (hard copy) as well as, or instead of, the screen images produced by a transaction. You have a choice of methods of producing such output. Which one you choose depends on your requirements. This section describes one method available, which is asynchronous page build transaction.

A 3270 printer contains a page buffer. BMS moves data into this page buffer when instructed to do so by EXEC CICS SEND MAP or EXEC CICS SEND CONTROL commands. The page buffer is printed only when BMS receives an EXEC CICS SEND MAP or EXEC CICS SEND CONTROL command containing the PRINT option. Likewise, it is erased only if BMS receives an EXEC CICS SEND MAP or EXEC CICS SEND CONTROL command that specifies the ERASE option. These properties of the printer make it possible for a program to build a single page of printed output from a series of maps.

Note: Each time you issue an EXEC CICS SEND CONTROL PRINT command, the output is spooled to a file in the /tmp directory. The file is queued to the printer after a syncpoint has been taken or when the transaction completes.

Two ways of printing a page built from multiple maps are:

1. Using the interval control START command.

   You use the START command to initiate a secondary CICS task. This is a print task if the TERMID option of the command names a printer as its *principal facility*. Your initial transaction can pass data to the print task by specifying the FROM and LENGTH options of the START command. If the primary transaction has already created a series of output data structures in the FROM area, the secondary transaction can map the data into the printer buffer, then initiate printing using a BMS EXEC CICS SEND with the PRINT option.
2. Using a transient data queue with a trigger level.

You can send symbolic map data structures to a transient data queue using the WRITEQ command. CICS can be made to initiate a print transaction when a specific number of records have been written to the queue. The name of the transaction to be initiated, the identifier of the printer that is to be its principal facility, and the trigger level at which it is started, are defined in the Transient Data resource definition.

Note, however, that output from several instances of your transaction may be interleaved on the transient data queue. This can be avoided if all the data to be printed by an instance of your transaction is stored in a single transient data queue item. Alternatively, each instance of your transaction can get exclusive control of the transient data queue by ENQ and DEQ commands.

Blank lines and 3270 printers: Every line in a map for a 3270 printer must contain field data (blanks if necessary), because the 3270 does not print empty lines (that is, lines of null characters).

Setting the printer page width: BMS builds device-dependent datastreams for 3270 printers by computing set buffer address (SBA) orders based on the page width specified by the **NumColumns** attribute in the Terminal Definitions (WD). If you are using **cicstermp**, you must set **NumColumns** to 132, because **cicstermp** emulates a 132 column printer.

Form feed characters: You can code an option, called FORMFEED, on the EXEC CICS SEND MAP and EXEC CICS SEND CONTROL commands. This generates a form feed character at the start of the datastream. If you code this option for an emulator that does not support form feed, CICS simply ignores the request.

The form feed character occupies screen position 1 (the top left-hand corner) on a 3270 display or printer. It can be overwritten by other data sent to the emulator, in which case form feed does not occur.

Be careful when using the FORMFEED option on an EXEC CICS SEND CONTROL command. The EXEC CICS SEND CONTROL command always generates a complete blank page. Thus an EXEC CICS SEND CONTROL FORMFEED skips to a new page and also sends this as a blank page. However, as described earlier, 3270 printers sometimes suppress null lines so that a blank page is printed as a single line.

See the *CICS Application Programming Reference*, the *CICS Administration Reference*, and the *CICS Clients: Administration* for related information.

BMS design considerations

To ensure the efficiency of applications using BMS, consider the following points:

Good screen design and effective use of the 3270 features significantly effects the number of bytes to be sent in the remote procedure call (RPC) and potentially therefore across the network. Consideration should be given to minimizing the number of transmissions to a terminal. The efficiency of the datastream affects both response time and line usage.

Sending unformatted data

If your output to a terminal is entirely, or even mostly, unformatted you can send it using native terminal control commands rather than BMS (that is, using SEND without the MAP option). This command is much more efficient in terms of processor overhead.

Do not use square brackets [] in maps or native terminal control commands if the transaction might be run, with transaction routing, from an EBCDIC system. CICS does not support square brackets in non-ASCII terminals.

Sending formatted data

When building a formatted datastream with BMS, you should bear in mind the factors described in the following list:

Avoid turning on modified data tags unnecessarily

The modified data tag (MDT) is the bit in the attribute byte that determines whether a field should be transmitted from a terminal to CICS.

The MDT for a field is normally turned on by the 3270 emulator (the CICS **cicsterm** and **cicsteld** processes) when the user enters data into a field. However, you can also turn on the tag when you send a map to the screen, either by specifying FSET in the map or by sending an override attribute byte that has the tag on. You should never set the tag on in this way for a field that is constant in the map, or for a field that has no label (and therefore is not sent to the program that receives the map).

Also, you do not normally need to specify FSET for an ordinary input field. This is because the MDT is turned on automatically in any field in which the user enters data. This is then included in the next SEND. These tags remain on, no matter how many times the screen is sent, until explicitly turned *off* by the program (by FRSET, ERASEAUP, ERASE, or by an override attribute with the tag off).

You can store information, between inputs, that the user did not enter on the screen. This is a reason for turning the MDT on by program. However, this storage technique is appropriate only to small amounts of data, and is more suitable for local than for remote terminals, because of the transmission overhead involved. For example, this technique is particularly useful for storing default values for input fields. In some applications, the user must complete a screen in which some fields already contain default values. A user who does not want to change a default just skips that field. The program processing the input has to be informed what these defaults are. If they are always the same, they can be supplied as constants in the program. If they are variable, however, and depend on earlier inputs, you can simply save them on the screen by turning the MDT on with FSET in the map that writes the screen. The program reading the screen then receives the default value from a user who does not change the field, and the new value from a user who does.

Note: The saved values are not returned to the screen if either the CLEAR, PA1, PA2, or PA3 key is pressed.

Use FRSET to reduce inbound traffic

If you have a screen with many input fields, which you may have to read several times, you can reduce the length of the input datastream by specifying FRSET when you write back to the screen in preparation for the next read. FRSET turns off the MDTs, so that fields entered before that write are not present unless the user reenters them the next time. If you are dealing with a relatively full screen and a process where there may be a number of error cycles (or repeat transmissions for some other reason), this can be a substantial saving. However, because only *changed* fields are sent on subsequent reads, the program must save input from each cycle and merge the new data with the old. This is not necessary if you are not using FRSET, because the MDTs remain on, and all fields are sent regardless of when they were entered.

Do not send blank fields to the screen

Sending fields that consist entirely of blanks or that are filled out on the right by trailing blanks to the screen usually wastes line capacity. The only case in which

BMS requires you to do this is when you need to erase a field on the screen that currently contains data, or to replace it with data shorter than that currently on the screen, without changing the rest of the screen.

This is because when BMS builds the datastream representing your map, it includes blanks but omits nulls. This makes the output datastream shorter. BMS omits any field whose first data character is null, regardless of subsequent characters in the field.

BMS requires you to initialize to nulls any area to be used to build a map. BMS uses nulls in attribute positions and in the first position of data to indicate that no change is to be made to the value in the map. If you are reusing a map area in a program, you should take special care to clear it in this way.

Use the MAPONLY option when possible

The MAPONLY option sends only the *constant* data in a map, and does not merge any variable data from the program. When you send a skeleton screen to be used for data entry, you can often use MAPONLY.

Send only changed fields to a screen that is not new

Sending only changed fields is important when, for example, a message is added to the screen, or one or two fields on an input screen are highlighted to show errors. In these situations, you should use the DATAONLY option to send a map that consists of nulls except for the changed fields. For fields in which only the attribute byte has changed, you need send only that byte, and send the remaining fields as nulls. BMS uses this input to build a datastream consisting of only the fields in question, and all other fields on the screen will remain unchanged.

It may be tempting to ignore this advice and send an unnecessarily long datastream. For example, when a program that is checking an input screen for errors finds one, there are two options. It can simply add the error information to the input map (highlighted attributes, error messages, and so on) and resend it, or it can build an entirely new screen, consisting of just the error and message fields. The former is slightly easier to code (you do not need to have two map areas or move any fields), but it may result in very much longer transmissions because the output datastream contains the correct input fields as well as the error and message fields. In fact, it may even be longer than the original input stream because, if there were empty or short fields in the input, BMS will have replaced the missing characters with blanks.

Design data entry operations to reduce line traffic

Often, users are required to complete the same screen several times. Only the data changes on each cycle; the titles, field labels, instructions, and so on remain unchanged. In this situation, when an entry is accepted and processed, you can respond with a SEND CONTROL ERASEAUP (or a map that contains only a short confirmation message and specifies ERASEAUP). This causes all of the *unprotected* fields on the screen (that is, all of the input data from the last entry) to be erased and to have their MDTs reset. The labels and other text, which are in protected fields, are unchanged, the screen is ready for the next data-entry cycle, and only the necessary data has been sent.

Compress data sent to the screen

When you send unformatted data to the screen, or create a formatted screen outside BMS, you can compress the data further by inserting set buffer address (SBA) and repeat-to-address (RA) orders into the data stream. SBA allows you to

position data on the screen, and RA causes the character following it to be generated from the current point in the buffer until a specified ending address. SBA is useful whenever there are substantial unused areas on the screen that are followed by data. RA is useful when there are long sequences of blanks on the screen.

Note: If you wish to insert a Set Buffer Address (SBA) into the datastream, then the values must be ASCII and not EBCDIC.

Use nulls instead of blanks

You should note that, outside BMS, nulls have no special significance in an *output* datastream. If you need a blank area on a screen, you can send either blanks or nulls to it; they take up the same space in the output stream. However, if the blank field is likely to be changed by the user and subsequently read, use nulls, because they are not transmitted back.

Use methods that avoid the need for nulls or blanks

For any *large* area of a screen that needs to be blank, you should consider methods other than transmitting blanks or nulls; for example, using BMS, putting SBA and RA orders directly into the data stream, or using the ERASE and ERASEAUP options. If there are no explicit references to color in the BMS map, the colors specified in the keymap file (.3270keys) are used in the 4-color mode operation.

On CICS for Windows, see the *CICS Clients: Administration* for default color definitions.

On CICS on Open Systems the default .3270keys file has the color definitions:

background
: black

low intensity protected
: cyan

low intensity unprotected
: green

high intensity protected
: white

high intensity unprotected
: red

These colors display the 4 different combinations of protect and intensify field attributes. For example, the default color for a field defined as **unprotected** and **high intensity** is red; if the characteristics of this field are changed to **unprotected**, the color changes to green (the default value for **unprotected**). Note also that if you specify COLOR=NEUTRAL for a field, the field is displayed in white, whereas fields that do not have the COLOR attributes specified, display the default color.

Using the BMS macros to code BMS map sets

This section describes the three macros DFHMSD, DFHMDI, and DFHMDF, that are used to define BMS map sets, maps, and fields. It show to use the macros to define a simple map set, and how to catalog this mapset for use by application programs.

The following macros are used to build map set definitions:

Map set definition macro (DFHMSD)
: The DFHMSD macro defines a map set. A map set contains one or more maps.

Map definition macro (DFHMDI)
The DFHMDI macro defines a map within the map set defined by the previous DFHMSD macro. A map contains zero or more fields.

Field definition macro (DFHMDF)
The DFHMDF macro defines a field within a map defined by the previous DFHMDI macro. A field name can be up to thirty characters long.

You process a BMS map set definition to generate a symbolic description map or a physical map (or both) using the BMS processor **cicsmap** command.

Note: Before CICS can load a physical map, you must define a Program Definitions (PD) entry for it with the **ProgType** attribute set to **map**.

The following information provides guidance on how to use these macros to define map sets. See the *CICS Application Programming Reference* for syntax descriptions of these macros.

Defining a map set

Use the DFHMSD macro to define the map set. This macro consists of operands that allow you to define the basic characteristics of the map, or maps, that make up the map set. Some of the DFHMSD operands establish defaults for the DFHMDI and DFHMDF macros.

Defining maps within a map set

Each map in a map set is defined using the DFHMDI macro. This macro is similar in form to DFHMSD and specifies defaults for fields within the map. It allows you to override some of the options inherited from DFHMSD, and to specify some new ones.

A map set definition must contain at least one map definition. Where you have more than one map, code their definitions one after another.

The number of maps per map set is limited to a maximum of 9998. (This is based on a map size of 8 characters.)

All maps in a map set are loaded whenever any one of them is used. If all the maps in a map set are used during a single invocation of the program, the single load of all maps is more efficient than loading each map as it is required. You should ensure that you use unique names for maps within a map set, or within multiple map sets that are copied into one application program.

Another reason for loading several maps at the same time is that more than one of them can appear on the screen at one time. This is because a map definition can specify where a map is to be placed on the screen. When BMS sends a map to a display, it does not erase the existing contents of the display unless you code the ERASE option. Instead, it uses your program data, plus constant map data, to overlay part of the screen. Therefore, if you design your maps so that they occupy different parts of a screen, you can display them at the same time. Alternatively, you can design some maps in a map set so that they overlay one another. In this way, you can erase parts of the contents of the screen without affecting the rest.

A map usually consists of one or more data fields. Each field contains display data, and has a set of associated attributes that are initialized by coding operands in a

DFHMDF macro. All field definition macros following a map definition macro belong to that map. The end of one field definition is indicated by the beginning of another.

Defining fields within a BMS map

The DFHMDF macro is used to specify initial attributes to be given to fields within a map.

Defining field groups

Very often, an output data display field has to contain several subfields that share the same display attributes, each of which might have to be modified separately. On output, subfields that have not been modified by the program can adopt default data values from the output map. For example, a display can include a date field with a *day* subfield, *month* subfield, and *year* subfield (shown later in the example of a map definition with the names DD, MM, and YY respectively). The contents of the year subfield remain constant over a relatively long period; its value can safely be taken from a map. However, the day value and month value must be updated regularly. Similarly, on input the terminal operator can enter data in each subfield separately.

You use the GRPNAME operand to define a group of subfields that combine to produce a field. The start of the group is indicated by a DFHMDF macro with the GRPNAME operand. This operand defines the first subfield, and specifies the attributes and name of the group. It is followed by other DFHMDF macros, one for each of the other subfields. Each of these must specify the group name, but cannot specify attribute values. The definition of the group is terminated by a DFHMDF macro that specifies a different group name, by one that specifies no group name, or by a DFHMDI or DFHMSD macro.

Briefly, a group of fields in a map might appear as follows in the map definition:

```
MAPSET DFHMSD....
       .
       .
MAP    DFHMDI....
       .
       .
DD     DFHMDF GRPNAME=DATE,POS=(6,40),LENGTH=2,ATTRB=...
       .
MM     DFHMDF GRPNAME=DATE,POS=(6,46),LENGTH=2
       .
YY     DFHMDF GRPNAME=DATE,POS=(6,52),LENGTH=2
       .
FIELD  DFHMDF LENGTH=5,COLOR=GREEN,...
       DFHMSD TYPE=FINAL
```

The POS operand specifies the position of the attribute byte of the field even though subfields of a group (other than the first) do not have attributes. If the subfields are positioned contiguously with no intervening blanks, the POS operand of the second and succeeding subfields must specify the position of the last character of the previous subfield.

Terminating a map set definition

A map set definition ends with a DFHMSD macro of the form:

```
[mapset] DFHMSD TYPE=FINAL
```

mapset is optional, but if used it must be the same as that on the DFHMSD macro that began the map set.

Coding the BMS definition macros

When coding the BMS macros, you must ensure that the information appears in the correct columns so that it can be processed by **cicsmap**.

Three column positions are significant:

StartCol
starting column (default 1).

ContCol
column in which continuation lines must begin (default 16).

EndCol
last column before continuation marker (default 71).

The following general coding rules apply:

- The label for each BMS macro must start in StartCol.
- The macro name must be separated from its label and first operand by one or more spaces.
- Operands must be separated by a single comma (,); spaces should be used with care because they cause the remainder of the line to be treated as a comment.
- Lines containing many operands can be continued after any comma, by placing a nonblank character (usually X) in the position after EndCol and then starting a new line with the first character of a new operand in ContCol.
- Blank lines and tab characters are not allowed.
- Comment lines are indicated by placing an asterisk (*) in StartCol.
- All macros and their operands (except INITIAL and GINIT data) must be in uppercase.

Chapter 5. Coding for data services

This chapter describes how to write application programs that use the CICS data services. The discussion covers

- "Relationship between CICS and file managers" discusses the relationship between CICS and file managers
  - SFS consistency, isolation, and locking
  - DB2 concurrency and locking
  - CICS and SFS performance with large files
- "Mixed resource manager applications" on page 105 discusses the use of mixed resource manager applications.
- "File services" on page 105 discusses how CICS maps VSAM requests onto SFS files or DB2 tables.
- "Queue services" on page 130 discusses the use of queues in CICS applications.
- "Journal services" on page 137 discusses the role journal services plays in CICS applications.
- "SQL restrictions and relational database services" on page 142 discusses relational database issues.
- "File processing using EXTFH with non-CICS applications" on page 157 discusses file processing using the External File Handler.

Relationship between CICS and file managers

You can manage CICS user files (files defined with the File Definitions (FD)) on either a DB2 or an Encina SFS file manager. Both file managers permit two or more regions or regions and non-CICS applications (such as batch processing programs) to share data.

When designing systems to exploit these facilities, consider the networking costs of transferring data between the File Manager and its client, and the benefits of distributing the processing and disk access load between several machines.

CICS attempts to minimize network traffic by avoiding unnecessary interactions with the file manager. For example, at the end of a transaction, CICS retains an SFS *open file descriptor (OFD)* for possible reuse where a region uses SFS for file access services. Such OFDs are released when a file is closed, either by an explicit EXEC CICS SET FILE CLOSED request, or when CICS is shut down.

Note: If CICS abends or is canceled, OFDs may not be released, including those for temporary storage queues (TSQs) and transient data queues (TDQs).

SFS consistency, isolation, and locking

The variety of attribute settings and locking modes available with SFS OFDs offer design efficiencies to CICS applications. Consider:

- OFD consistency
- Isolation level
- Lock modes

Refer to the *Encina SFS Programming Guide* for a detailed explanation of these concepts.

CICS uses SFS OFD attribute settings for consistency and isolation level differently based on the value of the **Recoverable** attribute in the File Definitions (FD):

- If a file is specified as not recoverable, then the following OFD attributes are requested by CICS:
  - Consistency : *sfs_nonTransactional*
  - Isolation : *sfs_nonTranCursorStability*

  Operations using OFDs specified as not recoverable do not participate in user transactions. As a result, an EXEC CICS SYNCPOINT ROLLBACK does not undo changes made with such OFDs. Such changes are immediately visible to other processes reading such records.
- If a file is specified as recoverable, then the following OFD attributes are requested by CICS:
  - Consistency : *sfs_Transactional*
  - Isolation : *sfs_cursorStability*

  Operations using OFDs specified as recoverable participate in user transactions. Changes made via such OFDs do not become visible to other processes until the transaction reaches EXEC CICS SYNCPOINT. Greater computing resources are required by operations using OFDs specified as recoverable than for those operations using OFDs specified as not recoverable.

SFS offers lock modes to use when records are accessed. CICS uses *sfs_noLock* mode when reading records and *sfs_writeLock* mode when reading records to update them. These modes allow multiple transactions to concurrently read a record, but prevent simultaneous updates of the record. The lock modes prevent a transaction from reading data for update that has been written via a recoverable OFD until an EXEC CICS SYNCPOINT is reached. However, a straight read-only operation can see the changes prior to the EXEC CICS SYNCPOINT. A record updated via a recoverable OFD is locked for writing until the transaction is resolved by reaching an EXEC CICS SYNCPOINT. As a result, greater interprocess contention occurs when a recoverable OFD is used.

DB2 concurrency and locking

When DB2 is used to manage CICS queue and user files, it treats all files as recoverable, regardless of the value set in the FD for the **RECOVERABLE** attribute.

CICS file services access DB2 using cursor stability isolation level and with record locking enabled. This configuration impacts the behavior and concurrency of CICS applications in a number of ways.

- An application does not retrieve uncommitted changes performed by another application. Changes made by a CICS transaction do not become visible to other applications until the transaction reaches EXEC CICS SYNCPOINT.
- When a transaction writes records to a file in which a browse operation has already been started, the newly inserted records are not visible to the browse operation. The browse operation displays a snapshot of committed table data at the time when the browse operation is started.
- Similarly, changes committed to a file by other applications are not visible to a browse that has already been started.
- Records can be concurrently read, but they cannot be simultaneously updated.
- An updated record is locked until an EXEC CICS SYNCPOINT is reached.

Oracle concurrency and locking

When Oracle is used to manage CICS queue and user files, it handless all files as recoverable, regardless of the value that is set in the FD for the RECOVERABLE attribute.

CICS and SFS performance with large files

The BufferPoolSize attribute, which is in the CICS SSD definitions, sets the size of the I/O buffer that is in the SFS server. This buffer exists only to allow a memory image of sections of the SFS data volume, so that the SFS process does not have to wait for the data volume disk. The information that is given here might help you improve the performance of SFS:

- A large I/O buffer might seem desirable, but that buffer is in the private data segment of the SFS process, which also contains memory that is allocated to manipulate the internal structures in SFS. These internal structures can use a large amount of memory. For example, 320 000 records, each of size 128 bytes, 4 indexes of size 8 bytes, and 1 index of 32 bytes uses approximately 110 MB of disk space. To load a file with data, you need about 160 MB of memory in the SFS private data segment. If, in this example, a BufferPoolSize attribute that is greater than 73 000 is used, this error is returned:

  ```
  Not enough memory in InitiateBufWrite
  ```

  The following command would set the size of the buffer to 70000:

  ```
  cicsupdateclass -w -c ssd -k "/.;/cics/sfs/myServ" -a BufferPoolSize –n 70000
  ```

  Note that the SFS server name has a semicolon where normally you would use a colon. Also, the server name must be inside quotes. The server must be warm started in either case to get the change to take effect.

  Note: The **svmon -P <pid>** command is very useful if you want to monitor the how much memory the data segment uses. This command shows the total amount that is allocated (in the InUse column), and also the address range that is allocated (in the Addr Range column). The data segment normally grows from both ends. When these ends meet, either an InitiateBufWrite error, or another memory error occurs.
- Before the I/O buffer is full, the SFS process can continue to run efficiently; that is, it uses at least 100% of one CPU if the load on the operating system allows it. The disks show that they are running at 100% capacity. If they are running only write operations, they are not actually slowing down the SFS process; they are catching up with the I/O buffer asynchronously. However, when the I/O buffer is full, the SFS process must wait for the data to be written, and it must reread data from disk because it cannot hold it all in the buffer at the same time. At this point, many read operations occur on the data disk, the CPU uses SFS less, and the loading rate decreases greatly.
- To help ensure that your disk write operations are the fastest possible, you can add extra data volumes to an SFS server. Although the basic CICS SSD definitions define only one data volume for an SFS server, you can add more after the initial cold start of the server. To add more data volume, type the following:

  ```
  tkadmin map lvolsfs_newVol sfs_newVol 64
  tkadmin enable lvol sfs_newVol
  sfsadmin add lvol sfs_newVol
  ```

  where `sfs_newVol` is the AIX logical volume name that was already defined through the normal AIX facilities (for example, smitty). For this action to be

effective, assign the AIX logical volume to a different physical disk from the disk that contains the first data volume for the SFS server. By using different physical disks, you allow write operations to run in parallel. You can then assign the secondary indexes for a file to volumes that are separate from the actual data and primary index (which always stay together). For example:

```
sfsadmin create clusteredfile mult.indices2 f1 byteArray 8 f2 byteArray 8 \
i1 -unique 1 f1 sfs_firstVol
sfsadmin add index mult.indices i2 1 f2 -volume sfs_newVol
```

If you ever cold start the SFS server, you must rerun the **lvol** commands to add the extra volumes.

- If the primary index has already put into sequence the input data for a particular file, and no secondary indexes are active, that file should load quickly. SFS slows if it has to recalculate unsequenced indexes.
- To flush the I/O buffer to disk, type:

  ```
  tkadmin flush lvol
  ```

  This command flushes all data volumes for the server. You can specify which data volume is to be flushed, if you have more than one.

  When a log volume becomes almost full, SFS tries to compress it. This operation can take a long time, and does not necessarily create enough free space. In this condition, you might find the **tkadmin force checkpoint** command useful. Use it between **sfsadmin** and **cicssdt** commands if the log volume is likely to become full. This command completely clears the log file only if all data has been flushed to disk.
- To determine how many pages to allocate in the SFS Server, you can use the rough formulas for size estimating. For example:

  If number of records to store is N, the width of the record is w, and the width of an Index is x:

  ```
  Pages (that is, 4096 bytes) for data + primary index = N * ( 28 + x)
  / 2650 + N * w / 4060 Pages for secondary index = N ( 20 + x) / 2650
  ```
- For fast loading:
  - Use a generous log volume size. It might need to be a few times larger than the size of the largest file.
  - Increase the soft data segment size for the userid to 52 4288 bytes (in 512 byte blocks) so that the full 256 MB segment can be used.
  - If you try to load a large file, and all indexes are active, that file needs, in private storage, a buffer that is about 1.5 times size of the disk data.

    To determine the maximum I/O buffer size that you can use, assume the starting size to be 240 MB. (It could be from approximately 240 MB to 256 MB depending on how much other modules in the process use the remainder of the memory. The ideal maximum is 256 MB, the segment size.)

    If the result gives a buffer that is smaller than the disk data size, you might find it just as fast to deactivate some of the indexes (which means more memory available for the I/O buffer), load the file, then reactivate the indexes.
  - Put one or more secondary indexes into separate data volumes. If possible, the initial load should have at least one nonsequenced index being built on each disk.
- Multiple SFS servers would give better performance if, in production, these conditions exist:
  - A particular machine has real memory that is not going to be used (that is, the machine is not regularly going to swap pages to the paging space).

– The I/O buffer that is possible with the 256 MB data segment cannot hold all the pages of SFS files that are in frequent use.

Multiple SFS servers can together provide more than 256 MB of data segments, and they hold more data in memory that they do on disk

Mixed resource manager applications

CICS coexists and complements other resource managers that access their resources through COBOL, C, C++, and PL/I programs. Typically, these resource managers are relational database management systems (RDBMSs) using the SQL language embedded in COBOL, C, C++ and PL/I programs.

You can write applications that access CICS and RDBMS resources in the same program. Refer to your RDBMS manuals for information on how to develop these applications. Generally:

- RDBMSs encourage prototyping and suggest to novice users that there is no need to follow conventional design strategies. This is definitely **not** recommended for online transaction processing. In particular, you should carefully undertake a full design of your data requirements.
- SQL is a set-level language (that is, it operates on a group of records rather than individual ones). When performing update operations, a large number of records could be locked until a syncpoint is reached. You need to carefully consider performance aspects when designing update operations.

File services

File services commands perform the functions needed to read, update, add, delete, and browse data in local or remote files.

When used locally, these commands provide access to files managed on either an SFS or a DB2 database. Used remotely, the commands provide access to the underlying file manager of the remote system. CICS provides an interface to a generic record-oriented file manager, regardless of the file manager, the type of file, or its physical organization.

Using a VSAM perspective to examine distributed CICS

If you are moving to CICS for Windows or to CICS on Open Systems from an IBM mainframe-based CICS platform, you possibly are familiar with *Virtual Storage Access Methods (VSAM)* data sets. VSAM is an access method for direct or sequential processing of fixed-length and variable-length records on direct access devices.

VSAM is not supported by your operating system, but the CICS-supported file managers provide similar facilities. As described in “Record handling in CICS files” on page 106, TXSeries CICS emulate VSAM.

For reasons of portability and commonality with other CICS platforms, the commands used for TXSeries CICS file services are described in terms of VSAM and not in terms of the file managers. The VSAM terminology can be confusing to new users of CICS. Similarly, users migrating applications from mainframe systems are equally unfamiliar with the file managers. Because of this, the VSAM discussions define the most important VSAM terms and correlates them to their CICS-supported counterparts. This correlation shows how CICS file services map onto the file managers used by CICS.

Record handling in CICS files

The records in a VSAM data set or file can be organized in any of three ways: logical sequence by a key field (key sequence), the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number. TXSeries CICS file services support:

- Fixed-length and variable-length records
- Multiple access paths to the same file
- Large records that can span system boundaries (control intervals, disk blocks)
- Record-level locking (or in the case of VSAM, control-interval locking).

If you are familiar with VSAM, you need to be aware of a major difference between VSAM and TXSeries CICS file services. You do not just define files and records to the file managers, as is done using VSAM. When using TXSeries CICS file services, you also have to define the record and each index on the file in terms of its constituent fields.

Each field can be of fixed or variable length. The index is defined with the File Definitions (FD) **IndexName** attribute. Failure to define the index results in error messages when the file is used.

CICS cannot access a file that has more than one variable-length field in a record. Also, if a record contains a variable-length field, this field must be the last field in the record.

In IBM mainframe-based CICS, the CICS transactional support overlays VSAM; some of the restrictions imposed on CICS file control reflect this. The file managers themselves provide full transactional support; several of these restrictions could be lifted in CICS for Windows or CICS on Open Systems. For example, neither file manager insists that a transaction is limited to a single update per file per LUW, whereas VSAM does. In order to preserve application portability, both to and from IBM mainframe-based CICS, TXSeries CICS have retained most of these restrictions.

Using primary and alternate indexes to access files

You must be able to access each record in your file by a unique key. For example, you can have records in a personnel file that have as their key an employee number. No matter how many Smiths there are, each has a unique employee number. The employee number can be the primary key to the base file. The base file must be defined to CICS in the File Definitions (FD).

Sometimes, however, you want to access the same set of records in different ways. For example, if you were producing a telephone directory from the file, you would want to list people by name rather than employee number. You can identify records in a file with an alternate or secondary key instead of the primary key described above. So the primary key is the employee number, and the employee name is the *alternate key*. Alternate keys are just like the primary key in Key-sequenced data set (KSDS) fields of fixed length and fixed position within the record. (The KSDS file type is discussed in "Key-sequenced data set (KSDS)" on page 107.) You can have any number of alternate keys and, unlike the primary or base key, alternate keys need not be unique. To continue our personnel example, the employee's department code might be defined as a further alternate key.

If you delete an Entry-sequenced data set (ESDS) record via an alternate index, an attempt to read the deleted record via the index, results in the condition NOTFND.

(The ESDS file type is discussed in "Entry-sequenced data set (ESDS)" on page 108.) An attempted read on the same deleted record via the RBA results in the condition ILLOGIC.

A separate FD entry is needed for each index by which you access a file. So in the example above, you would have one entry for access by the primary index using the employee number, and a second entry for access by the alternate index using the employee name.

CICS treats an alternate index as though it were a KSDS, although the keys need not uniquely identify the record. Whenever you update a record, all keys in alternate indexes are automatically updated; primary keys cannot be altered.

A CICS application program disregards whether it is accessing a file by its primary or alternate key.

See the *CICS Administration Guide* and the *CICS Application Programming Reference* for related information.

The types of files used by CICS

CICS uses three types of files:

- Key-sequenced data set (KSDS)
- Entry-sequenced data set (ESDS)
- Relative record data set (RRDS)

Both SFS and DB2 provide emulation for these file types.

Key-sequenced data set (KSDS)

A *key-sequenced data set (KSDS)* has each of its records identified by a key. The *key* of each record is simply a field in a predefined position within the record. Keys within a KSDS file need not be unique.

The physical order of the records depends on the collating sequence of the key field. This also fixes the order in which you retrieve records when you browse through the data set. An *index* relates the key of each record to the record's relative location in the data set. When you add or delete records, this index is updated accordingly.

When you write a KSDS record, you must specify the complete key, and when browsing, you must provide a record identification field (RIDFLD) sufficiently large to hold the complete key.

When you use a generic key, you must specify its length in the KEYLENGTH option. You must also specify the GENERIC option on the command. A generic key cannot have a key length equal to the full key length. You must define it to be shorter than the complete key.

You can also specify the GTEQ option on certain commands, for both complete and generic keys. The *GTEQ option* causes the command to position at, or apply to, the matching record in the data set. If no match is located, the command then positions at the record with the next higher key. When accessing a data set by way of an alternate index path, the record identified is the one with the next higher alternate key when a matching record cannot be found.

Even when using generic keys, always use a storage area for the RIDFLD equal in length to the length of the complete key. During a browse operation, after retrieving a record, CICS copies the actual identifier of the record retrieved into the RIDFLD area. CICS returns a complete key to your application, even when you specified a generic key on the command. For example, a generic browse through a KSDS returns the complete key to your application on each EXEC CICS READNEXT and EXEC CICS READPREV command.

Entry-sequenced data set (ESDS)

An *entry-sequenced data set (ESDS)* is one in which each record is identified by the address assigned to it when the record is first entered into the data set. This address is known as its *relative byte address (RBA)*.

Records are held in an ESDS in the order in which they were first loaded into the data set. New records added to an ESDS always go after the last record in the data set. You can not alter the length of a record, and you can not delete a record via the RBA base. The only way to delete a record is via a alternate index. After a record has been stored in an ESDS, its RBA remains constant. The browsing function retrieves the records in the order in which they were added to the data set.

You use the *RBA option* on most file control services commands to specify that the RIDFLD contains the RBA of the record to be accessed. An RBA is used to access an ESDS. All file control commands that refer to an ESDS base, and specify a RIDFLD, must specify the RBA option. The following list shows the commands:

- READ
- READNEXT
- READPREV
- RESETBR
- STARTBR
- WRITE

Relative record data set (RRDS)

A *relative record data set (RRDS)* has fixed-length slots, predefined to VSAM, in which records can be stored. An RRDS record is always fixed length, equal to the slot size. A record in an RRDS is identified by the *relative record number (RRN)* of the slot that holds it. When a new record is added to an RRDS, VSAM uses the number you supply with the file control request.

You use the *RRN option* on most file control services commands to specify that the RIDFLD contains the relative record number of the record to be retrieved. The first record in the data set is number one. All the file control commands that refer to an RRDS, and specify a RIDFLD, must specify the RRN keyword.

VSAM emulation by SFS and distributed CICS

The correspondence between VSAM data set types and SFS file types is shown in the following table:

Table 15. Comparison of VSAM data set types and SFS files

VSAM data set types	SFS files types
Key-sequenced data set (KSDS)	Clustered files
Entry-sequenced data set (ESDS)	Entry-sequenced files
Relative record data set (RRDS)	Relative files

VSAM emulation of KSDS files by SFS: SFS clustered files are used to emulate KSDS access to VSAM data. All segments of keys should be defined as being in ascending sequence. The primary key should not allow duplicate key values.

SFS allows keys to be segmented, that is, made up from two or more fields in any position in the record. You can use segmented keys in TXSeries CICS where file services are provided by SFS. However, you should not do so if you intend to migrate your CICS applications to regions where files services are provided by DB2, or to a CICS product that does not support segmented keys.

Defining keys affects the order of presentation of records in the following ways:

- If keys are defined using fields that are not byte arrays, then the order of presentation of records is that which SFS uses.
- If some segments of a key are defined as being in descending sequence, the order of presentation of records is affected.
- The GTEQ option associated with CICS commands is interpreted in the context of the ordering of the records used by SFS.

These factors affect CICS behavior when performing EXEC CICS READ and EXEC CICS STARTBR operations. For example, consider a request from an EXEC CICS READ command using a GTEQ option for a record with a key of "ZZZZ". This request could yield a record whose key is actually "AAAA" because of how order of presentation is handled by SFS.

VSAM-based CICS and distributed CICS interpret the EXEC CICS STARTBR command differently due to the SFS-imposed record ordering used by distributed CICS. Under VSAM-based CICS, an EXEC CICS STARTBR command that specifies a key GTEQ low values begins the search with the first or last records in the file. Distributed CICS always interprets an EXEC CICS STARTBR request specifying a key GTEQ low values as a request to start at the first record in a file. Similarly, a request for a key of high values is always interpreted as a request for the last record in the file. The record ordering imposed by SFS has precedence irrespective of where a high values or a low values key might be positioned in the file.

These potential misinterpretations can be avoided by using the *sfs_byteArray* field type.

CICS allows the use of files that have a primary index that is not unique. It also allows the use of files where the alternate indexes have duplicates. When files having these conditions are used, CICS limits the operations which you can perform. This limit occurs because all EXEC CICS READ and EXEC CICS READ UPDATE requests can access only the first of a set of duplicate records. This can be useful in accessing data prepared by a non-CICS application.

VSAM emulation of ESDS files by SFS: TXSeries CICS emulate ESDS using the SFS entry-sequenced file organization. The RBA is mapped onto the SFS implicit primary index for the file. This implicit primary file index is known as an entry sequence number (ESN). This is not necessarily the actual physical position of the record in the file, but it does identify the record as unique. The RBA returned to a CICS application is a four-byte unsigned quantity. This is derived from the eight-byte SFS entry sequence number. No meaning should be attached to the actual numeric value of the RBA. These RBA values should be treated as unique identifiers assigned in an ascending sequence.

CICS permits access to files containing records whose ESNs cannot be packed into a 32 bit RBA. However, records with such ESNs cannot be read, nor can new records be added to such files because, in each case, no RBA can be returned to the application.

VSAM emulation of RRDS files by SFS: SFS relative files are used to emulate RRDS files. SFS requires that the records contain one field to hold the RRN and a number of other fixed-length fields. For VSAM emulation, ensure that all fields other than the RRN field be of type *sfs_byteArray*.

The RRN is mapped onto the SFS relative slot number (RSN). CICS ensures that data seen by CICS applications does not include the RRN field.

Do not specify any secondary (alternate) indexes for these files.

RRDS files with alternate indexes can be accessed with their RRNs but not with their alternate indexes. If such files are used, the DUPREC condition occurs if an insertion or an updates would violate uniqueness constraints.

VSAM alternate index emulation by SFS: Alternate indexes are emulated using SFS secondary (alternate) indexes, which, like that primary key (index) can be segmented. Although SFS allows relative files to have secondary (alternate) indexes, CICS does not support this.

VSAM record emulation by SFS: When defining files to SFS, you must specify the fields that make up the record. For portable CICS family applications, use only *sfs_byteArray* and *sfs_varLenByteArray* type fields. Use of other data types is discussed in "Functionality differences with SFS" on page 112.

The CICS API does not permit records greater than 32 KB in length to be manipulated. Therefore, the total length of all the fields in your records must be less than or equal to 32 KB.

Fixed-length records: Fixed-length records must consist of one or more *sfs_byteArray* fields, the sum of their lengths equal to the required record length. You need a separate field for each index to the file, and a sufficient number of other fields to define the remainder of the record.

For example, Figure 5 on page 111 illustrates a VSAM file having the following characteristics:

- Record length is 20 bytes
- Primary index has a key that is five bytes long and begins at offset four
- Secondary (alternate) index has a key that is seven bytes long and begins at offset eleven.

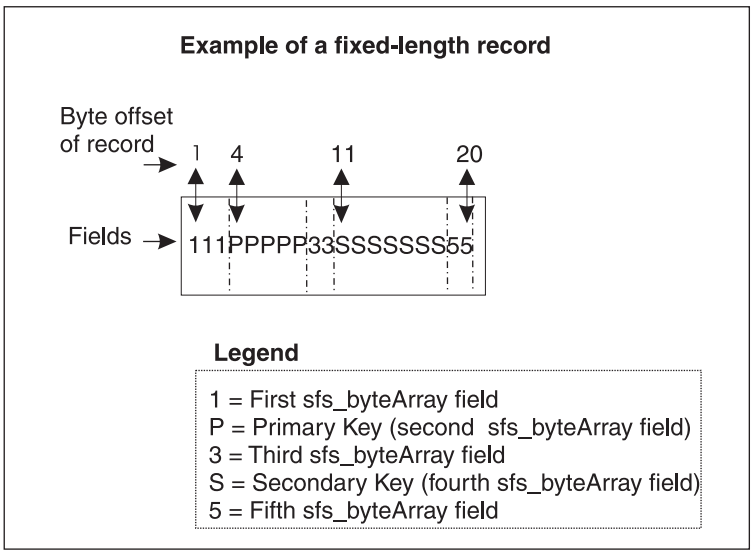


Figure 5. Fixed-length record example

To emulate this VSAM file to SFS, you would define five *sfs_byteArray* fields to SFS of lengths three, five, two, seven, and three respectively.

Variable-length records: To define a VSAM file that holds variable-length records to SFS:

1. Specify a number of *sfs_byteArray* fields for the portion of the record holding any key fields.
2. Specify one *sfs_varLenByteArray* type field. Define the length of this field so that the total record length is the maximum length you need.
3. The variable-length field must be the last field in the record.

Note: The size of the variable-length field is recorded in a four-byte header to the field by SFS. However, the data that is visible to the CICS application **does not** include this four-byte header.

For example, Figure 6 on page 112 shows a VSAM file having a variable-length record of maximum length of 1000 bytes. The keys for this record are in the same positions as the keys for the record discussed in Figure 5:

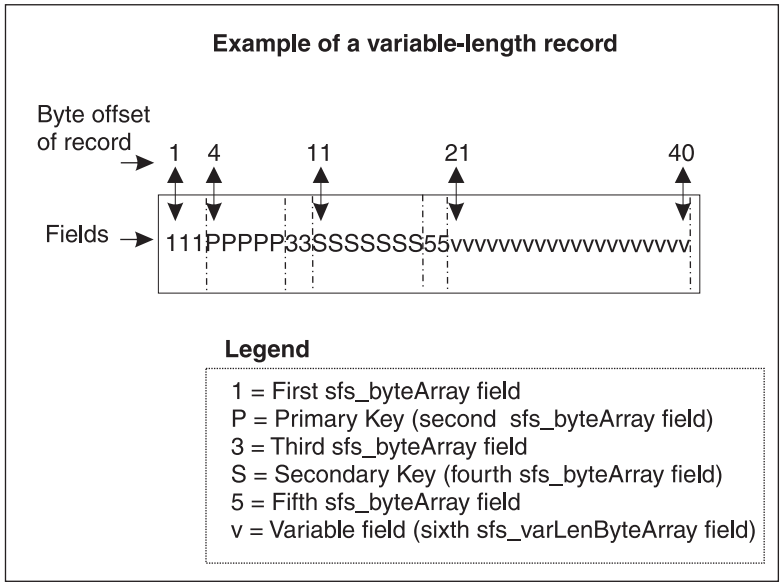


Figure 6. Variable-length record example

To emulate this VSAM file with its variable-length record, you define five *sfs_byteArray* fields to SFS of lengths three, five, two, seven, and three respectively, and one *sfs_varLenByteArray* of maximum length 979; and again define suitable indexes.

Note that although SFS's four-byte header to a variable-length field is invisible to CICS, a CICS application can make use of the field to convey information to the SFS. A CICS application can write a 40-byte record to SFS with a command such as:

```
EXEC CICS WRITE(FILE) FROM(buffer) LENGTH(40)
```

where *buffer* contains:

```
1 1 1 P P P P P 3 3 S S S S S S S  5 5 vvvvvvvvvvvvvvvvvvvv
```

CICS arranged that the actual record written to SFS has the length of the variable field of the record (twenty bytes) placed in the field header.

Functionality differences with SFS: The following behavior differences are important to understand when emulating VSAM files in SFS:

- CICS does not allow simultaneous access to SFS files.

  Security facilities are provided to configure SFS and CICS so that simultaneous access is restricted or prohibited.
- CICS can access SFS records containing data types other than the various byte arrays. To do this, you must be aware of the restrictions imposed by SFS on manipulations of these data types.
  - Numeric data types: SFS does not accept or return partial fields of numeric types such as *sfs_integer*. Ensure that applications that access records containing such fields do not attempt such operations. Examples of how such requests occur are:

    ```
    EXEC CICS READ INTO() LENGTH() ...
    ```

    where the specified length encompasses only part of a numeric field, and:

    ```
    EXEC CICS READ INTO() LENGTH()
              RIDFLD() GENERIC KEYLENGTH()
    ```

where the generic key specified includes only part of an integer field.
  - String data types: SFS requires that strings be null terminated. If data is supplied which is not null terminated, an SFS error can occur, and CICS raises an ILLOGIC condition.
- CICS does not permit access to files whose records contain more than one variable-length field. It refuses to open such files.
- SFS permits records to be deleted from Entry Sequenced files. CICS does not support this function. However, non-CICS processes can delete records from such files; this means that there is no guarantee that the first record in an ESDS file has an RBA of zero.
- In general, the use of native SFS calls is not supported. The following conditions describe the exceptions to this rule:
  - When XA resource managers are not used, SFS native calls are supported. XA resource managers are defined with Product Definitions (XAD) entries.
  - SFS native calls are supported when recoverable use of CICS file services or CICS queues is simple and does not require the use of EXEC CICS SYNCPOINT commands.

See the *CICS Administration Guide* and the *CICS Intercommunication Guide* for related information.

VSAM emulation by DB2 and distributed CICS

CICS can accomplish table creation in DB2 using the following methods:

- The **cicsddt** tool allows the interactive creation of CICS family portable table structures. (This is described in the *CICS Administration Guide* and the *CICS Administration Reference*.)
- The **cicsdb2import** command allows the creation of appropriate DB2 tables from Schema File Definitions (SCD). (This is described in the *CICS Administration Guide* and the *CICS Administration Reference*.)
- DB2 tables can also be created using the DB2 External File Handler.)
- DB2 tables can be created through the native SQL interfaces.

Each of these utilities supports the creation of appropriate DB2 indexes. It is important that all tables and indexes are created with the owner designated as CICS.

Use DB2 data types that are not associated with a coded character set when developing CICS family portable applications. This can be achieved by defining columns as CHAR or VARCHAR ″FOR BIT DATA″, or by using BLOBs.

The following sections give more information on the structure of DB2 tables required for CICS family portable applications.

VSAM emulation of KSDS files by DB2: A series of DB2 data types that are not associated with a coded character set must be used to create the table definitions. The number of columns defined must allow for each and every index against the file, as well as for an additional number of columns sufficient to define the remainder of the record.

Tables that emulate KSDS files must be created with a unique, primary index. For performance reasons, it is recommended that you also create a corresponding index with inverted ordering. For instance, the following command creates a primary index on the ACCT column of the account table:

```
CREATE UNIQUE INDEX CICS.ACCT0 ON CICS.ACCOUNT(ACCT)
```

In addition, a corresponding descending index should be created. This can be done using the following command:

```
CREATE UNIQUE INDEX CICS.ACCT0@ ON CICS.ACCOUNT(ACCT DESC)
```

Similarly, when an alternate index is created, the corresponding index with inverted ordering should also be created.

This process of creating a unique primary index and a corresponding index having an inverted order applies to table definitions in the following situations:

- Fixed-length KSDS with no overlapping alternate index fields
- Variable-length KSDS with no overlapping alternate index fields
- Variable-length KSDS with overlapping alternate index fields.

The following sections compare the procedures for creating table definitions for each of these situations in VSAM and in DB2 emulation.

Fixed-length KSDS with no overlapping alternate index fields: Table 16 and Table 17 compare the procedures for creating fixed-length KSDS records in VSAM and for emulating them in DB2:

Table 16. Fixed-length KSDS with no overlapping alternate index fields in VSAM

KSDS definition
Primary key (offset length) 1 5 Alt key 1 (offset length) 24 4 Alt Key 2 (offset length) 44 20 Record size (avg max) 500 500

If the record is defined as fixed-length, then a DB2 table definition can be created using a series of CHAR columns. One CHAR column is required for each key field. Additionally, each section of the record that lies before, between, and after the keys must have a CHAR column to define it. No section can be longer that 254 characters.

Table 17. Fixed-length KSDS with no overlapping alternate index fields emulated in DB2

DB2 columns
PART1 CHAR(1) NOT NULL FOR BIT DATA PKEY CHAR(5) NOT NULL FOR BIT DATA PART2 CHAR(18) NOT NULL FOR BIT DATA ALTKEY1 CHAR(4) NOT NULL FOR BIT DATA PART3 CHAR(16) NOT NULL FOR BIT DATA ALTKEY2 CHAR(20) NOT NULL FOR BIT DATA PART4 CHAR(254) NOT NULL FOR BIT DATA PART5 CHAR(182) NOT NULL FOR BIT DATA

Variable-length KSDS with no overlapping alternate index fields: Table 18 on page 115 and Table 19 on page 115 compare the procedures for creating variable-length KSDS records in VSAM and for emulating them in DB2:

Table 18. Variable-length KSDS with no overlapping alternate index fields in VSAM

KSDS definition
Primary key (offset length) 1 5 Alt key 1 (offset length) 24 4 Alt Key 2 (offset length) 44 20 Record size (avg max) 200 200

If the record is defined as variable-length, then a DB2 table definition can be created using a series of CHAR columns. One CHAR column is required for each index field. Additionally, each section of the record that lies before, between, and after the indexes must have one CHAR column to define it. The final column is defined as VARCHAR, LONG VARCHAR, or BLOB, depending on the maximum record length. The minimum length assumed is up to and including the last alternate index.

Table 19. Variable-length KSDS with no overlapping alternate index fields emulated in DB2

DB2 columns
PART1 CHAR(1) NOT NULL FOR BIT DATA PKEY CHAR(5) NOT NULL FOR BIT DATA PART2 CHAR(18) NOT NULL FOR BIT DATA ALTKEY1 CHAR(4) NOT NULL FOR BIT DATA PART3 CHAR(16) NOT NULL FOR BIT DATA ALTKEY2 CHAR(20) NOT NULL FOR BIT DATA PART4 VARCHAR(436) NOT NULL FOR BIT DATA

Variable-length KSDS with overlapping alternate index fields: Table 20 and Table 21 compare the procedures for creating variable-length KSDS records in VSAM and for emulating them in DB2. These examples illustrate the handling of overlapping alternate index fields:

Table 20. Variable-length KSDS with overlapping alternate index fields in VSAM

KSDS definition
Primary key (offset length) 1 5 Alt key 1 (offset length) 24 4 Alt Key 2 (offset length) 44 20 Record size (avg max) 200 200

In this example, the first index comprises columns ALTKEY1A and ALTKEY1B, while the second index comprises columns ALTKEY1B and ALTKEY2B.

While DB2 allows keys to be segmented, that is made up from one or more fields in any position in the record, it does not allow you to utilize segmented keys in CICS on Open Systems where file services are provided by DB2.

Table 21. Variable-length KSDS with overlapping alternate index fields emulated in DB2

DB2 columns
PART1 CHAR(1) NOT NULL FOR BIT DATA PKEY CHAR(5) NOT NULL FOR BIT DATA PART2 CHAR(18) NOT NULL FOR BIT DATA ALTKEY1A CHAR(2) NOT NULL FOR BIT DATA ALTKEY1B CHAR(2) NOT NULL FOR BIT DATA ALTKEY2B CHAR(18) NOT NULL FOR BIT DATA PART4 VARCHAR(454) NOT NULL FOR BIT DATA

Data associated with a coded character set: Defining keys affects the order of presentation of records in the following ways:

- If keys are defined using fields that are not defined as FOR BIT DATA, the order of presentation of records is that which DB2 uses. This is dependent on the database code page and the collation sequence specified when the database is created.
- The GTEQ option associated with CICS commands is interpreted in the context of the ordering of the records used by DB2.

This impacts CICS behavior when performing EXEC CICS READ and EXEC CICS STARTBR operations. For example, consider a request from an EXEC CICS READ command using a GTEQ option for a record with a key of "ZZZZ". This request could yield a record whose key is actually "AAAA" because of how order of presentation is handled by DB2.

VSAM-based CICS and distributed CICS interpret the EXEC CICS STARTBR commands differently due to the DB2–imposed record ordering used by CICS. Under VSAM-based CICS, an EXEC CICS STARTBR command that specifies a key GTEQ low values begins the search at the first or last records in the file. Distributed CICS always interprets an EXEC CICS STARTBR request specifying a key GTEQ low values as a request to start at the first record in a file. Similarly, a request for a key of high values is always interpreted as a request for the last record in the file. The record ordering imposed by DB2 has precedence irrespective of where a high values or a low values key might be positioned in the file.

These potential misinterpretations can be avoided by using the FOR BIT DATA column types when defining DB2 tables.

VSAM emulation of ESDS files by DB2: CICS emulates ESDS files using a special 32-bit RBA column. In addition, a sufficient number of other columns is required define the ESDS record. One column must be defined for each and every index against the file. Furthermore, a sufficient number of additional columns must be defined to encompass the remainder of the record. These columns cannot be associated with a coded character set.

CICS processing ensures that data seen by the CICS application does not include the RBA column. When a CICS facility, such as **cicsddt**, is used for DB2 table creation, an RBA column for a file of type ESDS is automatically generated. However, if you are using native SQL interfaces to create a table to emulate an ESDS file, then you must ensure that the first column in the table is named RBA. Furthermore, this column must be defined as CHAR(4) NOT NULL FOR BIT DATA.

Tables that emulate ESDS files should be created with a unique primary index against the RBA column. For performance reasons, it is recommended that you also create a corresponding index with inverted ordering. For instance, if you create an ESDS table named ESDSTAB, a primary index could be defined as follows:

```
CREATE UNIQUE INDEX CICS.ESDSTAB0 ON CICS.ESDSTAB(RBA)
```

A corresponding descending index should be created by using the following command:

```
CREATE UNIQUE INDEX CICS.ESDSTAB0@ ON CICS.ESDSTAB(RBA DESC)
```

Similarly, when an alternate index is created, a corresponding index with inverted ordering should also be created.

No meaning can be attached to the actual numeric value of the RBA. These RBA values are treated as unique identifiers assigned in an ascending sequence.

This process of creating a unique primary index and a corresponding index having an inverted order applies to table definitions in the following situations:

- Fixed-length ESDS with no overlapping alternate index fields
- Variable-length ESDS with no overlapping alternate index fields
- Variable-length ESDS with overlapping alternate index fields.

The following sections compare the procedures for creating table definitions for each of these situations in VSAM and in DB2 emulation. The ESDS table representation closely follows the equivalent KSDS case.

Fixed-length ESDS with no overlapping alternate index fields: Table 22 and Table 23 illustrate the procedures for creating fixed-length ESDS records in VSAM and for emulating them in DB2:

Table 22. Fixed-length ESDS with no overlapping alternate index fields in VSAM

ESDS definition
Alt key 1 (offset length) 24 4 Alt Key 2 (offset length) 44 20 Record size (avg max) 500 500

Table 23. Fixed-length ESDS with no overlapping alternate index fields emulated in DB2

DB2 columns
RBA CHAR(4) NOT NULL FOR BIT DATA PART1 CHAR(24) NOT NULL FOR BIT DATA ALTKEY1 CHAR(4) NOT NULL FOR BIT DATA PART2 CHAR(16) NOT NULL FOR BIT DATA ALTKEY2 CHAR(20) NOT NULL FOR BIT DATA PART3 CHAR(254) NOT NULL FOR BIT DATA PART4 CHAR(182) NOT NULL FOR BIT DATA

Variable-length ESDS with no overlapping alternate index fields: Table 24 and Table 25 illustrate the procedures for creating variable-length ESDS records in VSAM and for emulating them in DB2:

Table 24. Variable-length ESDS with no overlapping alternate index fields in VSAM

ESDS definition
Alt key 1 (offset length) 24 4 Alt Key 2 (offset length) 44 20 Record size (avg max) 200 500

Table 25. Variable-length ESDS with no overlapping alternate index fields emulated in DB2

DB2 columns
RBA CHAR(4) NOT NULL FOR BIT DATA PART1 CHAR(24) NOT NULL FOR BIT DATA ALTKEY1 CHAR(4) NOT NULL FOR BIT DATA PART2 CHAR(16) NOT NULL FOR BIT DATA ALTKEY2 CHAR(20) NOT NULL FOR BIT DATA PART3 VARCHAR(436) NOT NULL FOR BIT DATA

Variable-length ESDS with overlapping alternate index fields: Table 26 and Table 27 illustrate the procedures for creating variable-length ESDS records in VSAM and for emulating them in DB2. These examples illustrate the handling of overlapping alternate index fields:

Table 26. Variable-length ESDS with overlapping alternate index fields in VSAM

ESDS definition
Alt key 1 (offset length) 24 4 Alt Key 2 (offset length) 44 20 Record size (avg max) 200 500

Table 27. Variable-length ESDS with overlapping alternate index fields emulated in DB2

DB2 columns
RBA CHAR(4) NOT NULL FOR BIT DATA PART1 CHAR(24) NOT NULL FOR BIT DATA ALTKEY1A CHAR(2) NOT NULL FOR BIT DATA PART2 CHAR(18) NOT NULL FOR BIT DATA ALTKEY1B CHAR(2) NOT NULL FOR BIT DATA ALTKEY2B CHAR(18) NOT NULL FOR BIT DATA PART4 VARCHAR(454) NOT NULL FOR BIT DATA

VSAM emulation of RRDS files by DB2: CICS emulates RRDS files using a special 32-bit RRN column. In addition, a sufficient number of additional columns must be defined to encompass the remainder of the record. These columns cannot be associated with a coded character set.

CICS processing ensures that data seen by the CICS application does not include the RRN column. When a CICS facility, such as **cicsddt**, is used for DB2 table creation, an RRN column for a file of type RRDS is automatically generated. However, if you are using native SQL interfaces to create a table to emulate an RRDS file, then you must ensure that the first column in the table is named RRN. Furthermore, this column must be defined as CHAR(4) NOT NULL FOR BIT DATA.

Tables that emulate RRDS files should be created with a unique primary index against the RRN column. For performance reasons, it is recommended that you also create a corresponding index with inverted ordering. For instance, if you create an RRDS table named RRDSTAB, a primary index could be defined as follows:

```
CREATE UNIQUE INDEX CICS.RRDSTAB0 ON CICS.RRDSTAB(RRN)
```

A corresponding descending index should be created using the following command:

```
CREATE UNIQUE INDEX CICS.RRDSTAB0@ ON CICS.RRDSTAB(RRN DESC)
```

Do not specify alternate indexes for RRDS files. RRDS files with alternate indexes can be accessed with their RRNs but not with their alternate indexes. If such files are used, then DUPREC conditions can occur if insertions or updates would violate uniqueness constraints.

Table 28 on page 119 and Table 29 on page 119 illustrate the procedures for creating RRDS files in VSAM and for emulating them in DB2:

Table 28. RRDS files in VSAM

RRDS definition
`Record size (avg max)  500  500`

Table 29. RRDS files in emulated in DB2

RRDS definitions

`RBA      CHAR(4)      NOT NULL FOR BIT DATA`
`PART1    CHAR(24)     NOT NULL FOR BIT DATA`
`ALTKEY1A CHAR(2)      NOT NULL FOR BIT DATA`
`PART2    CHAR(18)     NOT NULL FOR BIT DATA`
`ALTKEY1B CHAR(2)      NOT NULL FOR BIT DATA`
`ALTKEY2B CHAR(18)     NOT NULL FOR BIT DATA`
`PART4    VARCHAR(454) NOT NULL FOR BIT DATA`

VSAM alternate index emulation by DB2: Alternate indexes are emulated using DB2 secondary indexes. Although DB2 allows relative files to have secondary indexes, CICS does not support this.

VSAM record emulation by DB2: The CICS API does not permit records greater the 32 KB in length to be manipulated. Therefore, the total length of all the fields in your records must be less than or equal to 32 KB.

The size of a variable-length field in DB2 is recorded in a prepended header to the field. However, the data visible to the CICS application does not include this header.

Functionality differences with DB2: The following behavior differences are important to understand when emulating VSAM files in DB2:

- CICS allows simultaneous access to DB2 files, either by two CICS regions, or by a CICS region and a non-CICS application. Applications need to be designed to account for this possibility.

  Security facilities provided to configure DB2 and CICS can be used to restricted or prohibited simultaneous access.
- The Numeric and Datetime DB2 data types are not supported by CICS and cannot be accessed by CICS applications.
- CICS allows the use of files whose primary index is not unique. It also allows the use of files where the alternate indexes have duplicates. When files with these conditions are used, CICS limits the operations which you can perform. This limit occurs because all EXEC CICS READ and EXEC CICS READ UPDATE requests can access only the first of a set of duplicate records. This can be useful in accessing data prepared by some non-CICS application.
- CICS does not permit access to files whose records contain more than one variable-length field. It refuses to open such files.
- DB2 permits records to be deleted from entry sequenced files. CICS does not support an interface to this. However, non-CICS processes can delete records from such files; this means that there is no guarantee that the first record in an ESDS file has an RBA of zero.

See the *CICS Administration Guide*and the *CICS Intercommunication Guide* for related information.

Differences in file behavior between SFS and DB2

The following list describes some differences in file behavior between SFS files and DB2 tables:

- In DB2, the maximum length for any char field is 254 bytes. See the DB2 restrictions documented in the SQL reference manual.
- All records over 4005 bytes are treated as variable by CICS using DB2 as the file manager. The following behaviors occur in migrated CICS applications that were originally written to access fixed-length records over 4005 bytes long:
  - On a READ, the value returned in the length field is the true length of the data, not the expected fixed length.
  - On a READ, the LENGTH option must be specified, otherwise LENGERR is returned.

  When DB2 is used as the file manager, files over 4005 bytes long must be created with the last column of a variable type (VARCHAR, LONGVAR, BLOB). The total length of the preceding fixed columns cannot exceed 4005 bytes.
- The process of migrating CICS files from SFS to DB2 is subject to the following restrictions:
  - SFS files that have a fixed length of over 4005 bytes must be redefined.
  - SFS files that have a fixed length of over 4005 bytes and have key fields beyond byte 4004 cannot be directly migrated. The file must be redesigned to ensure all key fields are within the first 4005 bytes of the record.

Accessing files from CICS application programs

This section describes the following functions available to application programs for accessing files:

- Reading records
- Updating records
- Deleting records
- Adding records.

Each of these functions is discussed for the KSDS, ESDS, and RRDS file types. In addition, the command options that can be used in combination with file services commands are discussed. Finally, a discussion on avoiding transaction deadlocks is included.

Reading records

There are several methods of reading records, including:

- Direct reading
- Sequential reading (browsing)
- Skip-sequential browsing

If the file contains fixed-length records, and if the application program provides an area into which the record is to be read, that area must be of the defined length. If the file contains variable-length records, the command must also specify the maximum length of the area provided to hold the records.

For fixed-length records and for records retrieved into CICS-provided SET storage, no length argument needs to be specified. However, a specified length argument is helpful when reading fixed-length records without using a SET command. The specified length argument ensures that the record being read is not too long for the available data area. If you provide the length argument, CICS uses the length field to return the actual length of the record retrieved.

Direct reading: The EXEC CICS READ command reads records in a file. The command must identify the record you want, and it must indicate whether the record is to be read into an area of storage provided by your application program (EXEC CICS READ INTO), or into CICS SET storage acquired by file control (EXEC CICS READ SET). If area is to be CICS SET storage, the address of the data in the CICS SET storage is returned to your program.

The length of time that the CICS SET storage remains valid depends on whether the EXEC CICS READ command is issued for UPDATE or read-only access. The SET storage for a non-update EXEC CICS READ command survives until another non-update or UPDATE READ command (either INTO or SET) is encountered for the same CICS file. The SET storage for an EXEC CICS READ UPDATE command survives until the next EXEC CICS REWRITE, EXEC CICS UNLOCK, EXEC CICS DELETE (without RIDFLD) or EXEC CICS SYNCPOINT, whichever is encountered first.

For both UPDATE and non-update commands, you must identify the record to be retrieved by the record identification field specified in the RIDFLD option. Immediately upon completion of an EXEC CICS READ UPDATE command, the RIDFLD data area is available for reuse by the application program.

To support application portability to VSAM environments, you can only specify one UPDATE operation for each file within a transaction at any given time. Your next command to the file should be an EXEC CICS REWRITE, EXEC CICS DELETE without the RIDFLD option, or EXEC CICS UNLOCK.

Direct reading from a KSDS: When reading from a KSDS, you identify the record you want by specifying its full key. However, if the key is not unique, the DUPREC condition occurs.

You can also specify a partial (generic) key. When you specify a partial key, both SFS and DB2 retrieve the first record whose leftmost characters match the partial key. Additionally, you can retrieve the record in the file whose key is greater than or equal to the full key provided with the command. Finally, you can also identify the record you want by providing a generic key together with the greater than or equal option (GTEQ) .

An EXEC CICS READ command raises the NOTFND condition if no record with the key specified is found. Also, the NOTFND condition is raised if no record is found with a key greater than or equal to the specified key when the GTEQ option is used. Either the GTEQ option or the EQUAL option can be specified. The EQUAL option requests a record having an exact match with the specified key. The GTEQ option requests a record with the first key greater than or equal to the specified key.

Direct reading from an ESDS: When reading from an ESDS, you identify the record you want by an RBA. Because the RBA of a ESDS record cannot change (unless the file is reorganized), your application program can keep track of the values of the RBAs corresponding to the records it wants to access. An access to an ESDS specifying an incorrect RBA, or an RBA where there is no record, returns the ILLOGIC condition.

Direct reading from an RRDS: When reading from an RRDS, you identify the record you want by its relative record number. Again, the application program must know the RRN values of the records it wants. For records not present in the file, the NOTFND condition is returned.

Direct reading by way of a path: If a KSDS or an ESDS has an alternate index and an appropriate entry in the File Definition, you can retrieve a record in the file by using the alternate key that you set up in the alternate index. The generic option and the greater than or equal option still work in the same way as for a read from a KSDS using the primary key.

If the alternate key in an EXEC CICS READ command is not unique, the first record in the file with that key is read, and you get the DUPKEY condition. To retrieve other records with the same alternate key, you have to start a browse operation at this point.

Sequential reading (browsing): The EXEC CICS STARTBR command begins the browse function. You must identify a particular record in the same way for a browse function as for a direct read function. However, EXEC CICS STARTBR only identifies the starting position for the browse; it does not retrieve a record.

You can reset a browse at any time using the EXEC CICS RESETBR command. Use EXEC CICS RESETBR to define a new starting position for the browse, or to change the type of search argument used.

The EXEC CICS READNEXT command reads records sequentially from the file; its starting point is set by the EXEC CICS STARTBR command. Each time an EXEC CICS READNEXT command is processed, CICS updates the field specified in the RIDFLD option with the complete key, relative byte address, or relative record number of the record retrieved.

The record can be read into an area of storage supplied by the application program (EXEC CICS READNEXT INTO), or into CICS-provided SET storage (EXEC CICS READNEXT SET). In the latter case, the CICS SET storage remains valid until the next EXEC CICS ENDBR for this REQID, or EXEC CICS SYNCPOINT, whichever is encountered first.

The EXEC CICS READPREV command performs the inverse process to the READNEXT command; it reads the records sequentially backwards from the starting point set by the EXEC CICS STARTBR command.

If your file supports unique keys only and you change from EXEC CICS READNEXT to EXEC CICS READPREV (or the converse), then the same record is retrieved twice. It is first retrieved by the EXEC CICS READNEXT command, and then on the READPREV command.

For browsing (and for direct read), if the file contains fixed-length records, and if the application program provides an area into which the record is to be read, that area must be of the defined length. If the file contains variable-length records, the command must also specify the maximum length of the area provided to hold the records.

Browsing through a KSDS: You can use a generic key on the EXEC CICS STARTBR command when browsing through a KSDS. However, the browse can only continue forward through the file. If you process an EXEC CICS READPREV during a generic key browse, you get the INVREQ condition.

You can use the options EQUAL and GTEQ on the EXEC CICS STARTBR command. The default is GTEQ option. If no record matches the key specified with the EXEC CICS STARTBR command, you get the NOTFND condition.

An EXEC CICS READNEXT or READPREV command works only after the successful execution of an EXEC CICS STARTBR.

You can start a forward browse through a KSDS at the start of the file by specifying a key of hexadecimal zeros, or by specifying options GENERIC, GTEQ, and KEYLENGTH(0) on the EXEC CICS STARTBR or EXEC CICS RESETBR command. (In the latter case, you need the RIDFLD keyword, even though its value is not used.) You can also start from the end of the file by specifying a complete key of X'FF's on the EXEC CICS STARTBR. This points to the last record in the file ready for a backward browse.

If you browse with a key whose underlying fields, as defined to SFS, include a numeric type, then the order in which the browse results are returned for that field are numeric. This is different from mainframe CICS which treats the key as a bit pattern. You can use byte arrays to avoid this problem. You should also note that SFS allows descending keys.

Browsing through an ESDS: You cannot use the GTEQ option on the EXEC CICS STARTBR command when browsing through a ESDS. If no record matches the RBA specified in the EXEC CICS STARTBR command, CICS raises the ILLOGIC condition. Like KSDS, keys of X'00's and X'FF's on the EXEC CICS STARTBR command enable browses to start at the first or last record respectively.

Browsing through an RRDS: You can use the GTEQ option on the EXEC CICS STARTBR command when browsing through an RRDS. On a direct EXEC CICS READ command, this option has no effect. A direct read GTEQ command giving an RRN that does not exist returns NOTFND because only the EQUAL option is taken. However, an EXEC CICS STARTBR GTEQ command using the same RRN completes successfully, and sets a pointer to the relevant position in the file for the start of the browse. The first record in the file is identified using 01 and the last record by X'FF's.

Browsing using a path: Browsing can also use an alternate index path to a KSDS or an ESDS. The browse is just like that for a KSDS, but using the alternate key. The records are retrieved in alternate key order.

When non-unique alternate keys are involved, a browse operation retrieves all records with the same alternate key. The EXEC CICS READNEXT command retrieves those records in an underlying file manager-dependent order. The DUPKEY condition is returned for each retrieval operation except the last. For example, if there are three records with the same alternate key, the DUPKEY condition is raised for retrieval of the first two, but not the third. You can design the application program to revert from browsing to direct reading when the DUPKEY condition no longer occurs.

Ending the browse: Stop a browse with the EXEC CICS ENDBR command. Trying to browse past the last record in a file raises the ENDFILE condition. Always issue the EXEC CICS ENDBR command before a syncpoint, before task termination, or before performing an I/O operation on the same file (READ, UPDATE, DELETE with RIDFLD, or WRITE). If you do not, you get unpredictable results, possibly including deadlock.

Simultaneous browse operations: CICS allows a transaction to perform more than one browse on the same file at the same time. You distinguish between browse operations by including the REQID option on each browse command.

Skip-sequential browsing: For direct access to records quickly, you can browse using skip-sequential processing. This type of browsing reduces index search time.

To use skip-sequential processing, you change the key, RBA, or RRN in the RIDFLD option of the EXEC CICS READNEXT or READPREV command to point to the next record you want. You can even do this on the first EXEC CICS READNEXT command or the first EXEC CICS READPREV command after an EXEC CICS STARTBR or EXEC CICS RESETBR command.

Note: The RIDFLD option on the EXEC CICS READNEXT or READPREV command must be in the same form (key, RBA or RRN) as that used in the EXEC CICS STARTBR command or last EXEC CICS RESETBR command. If you use generic keys on a forward browse, the new RIDFLD must also be a generic key, although the length can be different.

Specifying a different keylength in the KEYLENGTH option of an EXEC CICS READNEXT command has the same effect as an EXEC CICS RESETBR command. To continue browsing from this new point, remove the KEYLENGTH option from subsequent EXEC CICS READNEXT commands.

If a key equal to search is specified on an EXEC CICS STARTBR command, or an EXEC CICS RESETBR command, a EXEC CICS READNEXT command using skip-sequential processing can give a NOTFND condition. It is not possible to obtain the last record on the file by specifying a complete key of X'FF's during skip sequential processing. You must use EXEC CICS RESETBR to specify the key in this form.

Updating records

To update a record, you must first retrieve it using an EXEC CICS READ command with the UPDATE option. The record is identified in exactly the same way as for a direct read. After modification by the application program, the record is written back to the file using the EXEC CICS REWRITE command. In a KSDS or ESDS, the record can (as with a direct read) be accessed through either the primary key or any alternate key.

The EXEC CICS REWRITE command cannot identify the record being rewritten. In any one transaction, CICS allows only a single update to a given file to be in progress at any time. Therefore, the record being rewritten is identified by the previous EXEC CICS READ UPDATE command.

A record retrieved as part of a browse operation can not be updated during the browse. The application program must end the browse; read the desired record with an EXEC CICS READ UPDATE, and perform the update. Failure to do this can cause a deadlock.

The record to be updated can (as in the case of a direct read) be read into an area of storage supplied by the application program or into CICS-provided SET storage. If the record is read into CICS SET storage, it should normally be copied into application program storage and rewritten from that storage. For an EXEC CICS READ UPDATE command, CICS SET storage remains valid until the next EXEC CICS REWRITE, EXEC CICS UNLOCK, EXEC CICS DELETE (without RIDFLD) or EXEC CICS SYNCPOINT, whichever is encountered first.

For a KSDS, the primary key in the record must not be altered when the record is modified. Similarly, if the update is being made by way of an alternate index, the

alternate key used to identify the record must not be altered either, although other alternate keys can be altered. If the file allows variable-length records, the length of the record can be changed.

Specifying record length: When rewriting a fixed-length record, do not include the length with the command. If the length is specified, and does not agree with that defined to SFS or DB2, then the record is not written.

For variable-length records, you must specify the length with both the EXEC CICS READ and the EXEC CICS REWRITE commands. The length specified must be less than, or equal to, the keyspan of the underlying SFS file (including alternate indexes). If the length specified is greater than the maximum defined to SFS or DB2, then a record is truncated.

Note that the length of records read from an ESDS cannot be changed.

Deleting records

CICS does not allow records to be deleted from an ESDS.

You can delete a record in a KSDS or RRDS by first retrieving it for update, and then issuing an EXEC CICS DELETE command. You can also delete a record in a single operation, again using the EXEC CICS DELETE command. In this case, you must identify the record to be deleted as part of the command by specifying the RIDFLD option. For a KSDS or RRDS, instead of rewriting the record, the application program can issue an EXEC CICS DELETE command to erase it from the file. As in the case of the EXEC CICS REWRITE command, the record to be deleted must not be identified within the EXEC CICS DELETE command, but is, by default, the record most recently read for update. When an EXEC CICS DELETE command follows a EXEC CICS READ UPDATE, it must not include the RIDFLD option. If RIDFLD is included, an INVREQ condition is returned to the application program.

If a full key is provided with the EXEC CICS DELETE command, a single record with that key is deleted. So, if the file is being accessed by way of an alternate index path that allows non-unique alternate keys, only the first record with that key is deleted. After the deletion, you know if further records exist with the same alternate key, because you get the DUPKEY condition if they do.

Unlocking: Use the EXEC CICS UNLOCK command to release a lock held by an EXEC CICS READ UPDATE command without rewriting or deleting the record. EXEC CICS UNLOCK releases any CICS storage acquired for the EXEC CICS READ and releases SFS resources held by the EXEC CICS READ.

Deleting groups of records (generic delete): You can use a generic key with the EXEC CICS DELETE command. Then, instead of deleting a single record, all the records in the file whose keys match the generic key are deleted with the single command. The number of records deleted is returned to the application program if the NUMREC option is included with the command. If access is by way of an alternate index path, the records deleted are those whose alternate keys match the generic key.

Adding records

You can add new records to a file with the EXEC CICS WRITE command. They must always be written from an area provided by the application program.

Adding to a KSDS: When a record is being added to a KSDS, the primary key of the record identifies the position in the file where the record is to be inserted. The application program must specify the key using the RIDFLD option on the EXEC CICS WRITE command, even though the key is part of the record.

A record added to a KSDS by way of an alternate index is also inserted into the file in the position determined by the primary key. However, in this case the RIDFLD option must be used to specify the alternate key.

Adding to an ESDS: A record added to an ESDS is always added to the end of the file. You cannot insert a record in an ESDS between existing records. After the operation is completed, the relative byte address (RBA) identifying the record is returned to the application program.

When adding a record to an ESDS by way of an alternate index path, the record is also placed at the end of the file. The command must include the alternate index key in the same way as for a KSDS path. The commands are the same whether you are using an ESDS or a KSDS so the application program is not affected by which type of file is in use.

Adding to an RRDS: To add a record to an RRDS, include the relative record number (RRN) as a record identifier on the EXEC CICS WRITE command. The record is then stored in the file in the position corresponding to the RRN.

Specifying record length: When writing to a fixed-length file, do not include the length with the command. If the length is specified, and does not agree with that defined to SFS or DB2, then the record is not written.

If the file is defined as containing variable-length records, the command must always include the length of the record. The length specified must be less than, or equal to, the keyspan of the underlying SFS or DB2 file (including alternate indexes). If the length specified is greater than the maximum defined to SFS or DB2, then a truncated record is written.

See the *CICS Administration Reference* and the *CICS Application Programming Reference* for related information.

Using options with file services commands

This section discusses the following command options that are used in combination with file services commands:

- The RIDFLD option
- The INTO and SET options
- The FROM option

Using the RIDFLD option: Any function you perform on a record (read, add, delete, or start a browse) requires that you identify the record by the RIDFLD option. The only exception to this requirement is when you have initially read the record for update first. Additionally, you must include the RIDFLD option when performing a browse operation using the EXEC CICS READNEXT or EXEC CICS READPREV commands. The RIDFLD option gives CICS a way to return the identifier of each record retrieved.

With EXEC CICS READNEXT or EXEC CICS READPREV commands, the application program would not usually set the RIDFLD field. Therefore, after each occurrence of these commands, CICS updates the RIDFLD field with the actual

identifier of the record retrieved. (You can alter the RIDFLD value to set a new position from which to continue the browse.)

The RIDFLD option identifies a field containing the record identification appropriate to the access method and the type of file being accessed. By itself, the RIDFLD option usually is not enough to identify a specific record in the file. In addition, one or both of the options GTEQ and GENERIC can be used with your command. You use the GTEQ option to search for the first key that is greater than or equal to the key specified in RIDFLD. The GENERIC option is available if you are specifying an imprecise or partial key. These rules apply in the following circumstances:

- When any record retrieval is done from a KSDS
- When an alternate index path is used from a KSDS or ESDS
- When setting a starting position for a browse in a KSDS or ESDS

Indexing and the RIDFLD option: An index can contain multiple fields. When you create a schema (see the *CICS Administration Reference*), the index is made up of the fields listed in the "Field Names for Primary Index" or "Secondary Index - Field Names" fields. This list is separated by spaces, as shown in these example fields for an address record: *number street state zip*. However, the fields are not separated by spaces when you pass the field names with the RIDFLD option, for example: *numberstreetstatezip*). When passed by the RIDFLD option, these fields are concatenated together in the same order in which the fields are defined in the schema file definitions. This is not necessarily the same as the order in which the fields appear in each file record.

Using our address record field example, you can assume that the file definition consists of 4 fields, all of which are byteArrays of 1 byte each. If an index file were made up of field 2 and field 4, then the RIDFLD for record "*numberstreetstatezip*" would be "*streetzip*". The command would be specified as:

```
EXEC CICS WRITE FILE() FROM("numberstreetstatezip") RIDFLD("streetzip")
```

If, however, a alternate index consisted of field 4 following by field 2, the command would be:

```
EXEC CICS WRITE FILE() FROM("numberstreetstatezip") RIDFLD("streetzip")
```

This means that any field in an index can be sorted in any order. If we use the address record example, the file contains house numbers, street name, states and zip codes. This file can be sorted on states, and a alternate index can sort on zip codes. An index on the street name field allows a search for all entries matching the street name "Maple". Alternate indexes allow search criteria to be combined to yield a more narrow match, for example, a search for all entries matching the street name "Maple" and matching the state "Maine".

Any field in an index can be sorted in ascending or descending order. And any field can be unique or non-unique. This explanation applies only to KSDS files, or a KSDS path over an ESDS file.

Combining the KEYLENGTH and RIDFLD options: In general, file services commands need the RIDFLD and KEYLENGTH options. The KEYLENGTH option can be specified explicitly in the command, or determined implicitly from the SFS file definition.

For remote files (that is, those for which the SYSID option has been specified), the KEYLENGTH option must be specified if RIDFLD holds a key.

When accessing remote files, refer to the documentation for the underlying file manager for any restrictions in specifying CICS file services commands.

See the *CICS Intercommunication Guide* and the *CICS Application Programming Reference* for related information.

Using the INTO and SET options: With the EXEC CICS READ, EXEC CICS READNEXT, or EXEC CICS READPREV commands, the record is retrieved and put in main storage according to the INTO and SET options.

- The INTO option specifies the main storage area into which the record is to be put.

  For fixed-length records, do not include the LENGTH option. If you do, the length specified must exactly match the defined length, or you get the LENGERR condition.

  For variable-length records, always specify (in the LENGTH option) the longest record your application program accepts. (This value is likely to correspond with the value defined to SFS as the maximum record size when the file was created.) If you do not specify the LENGHTH option, you get the LENGERR condition. The LENGERR condition occurs if the record exceeds this maximum length, and the record is then truncated to that length. After the record retrieval, if you include the LENGTH option, the data area specified in it is set to the actual record length (before any truncation occurs).
- The SET option specifies a pointer to the address that holds the record. This address, which is acquired by CICS, points to the buffer in main storage that holds the record. When using the SET option, do not include the LENGTH option. If you do include it, the data area specified is set to the actual record length after the record has been retrieved.

Using the FROM option: When you add records (using the EXEC CICS WRITE command), or update records (using the EXEC CICS REWRITE command), the record that is to be written is specified with the FROM option.

The FROM option specifies the main storage area that contains the record to be written. In general, this area is part of the storage owned by your application program. With the EXEC CICS REWRITE command, the storage area indicated by the FROM option is usually (but not necessarily) the same area as the corresponding area indicated by the INTO option on the EXEC CICS READ UPDATE command. The length of the record can be changed when rewriting to a variable-length KSDS.

Always include the LENGTH option when writing to a variable-length file. If the value specified exceeds the maximum allowed in the SFS definition, The LENGERR condition is raised when the command is executed, and a truncated record is written. The LENGERR condition is also raised if the length option is omitted when accessing a variable-length file.

When writing to a fixed-length file, CICS uses the length specified in the SFS file definition as the length of the record to be written. So, the LENGTH option is not needed. If you do use the LENGTH option, its value is checked against the defined value, and you get the LENGERR condition if the values do not match.

See the *CICS Administration Reference* and the *CICS Application Programming Reference* for related information.

Avoiding transaction deadlocks

A deadlock occurs when a transaction (say, transaction A) requires exclusive use of a resource, such as a record in a file, and another transaction (say, transaction B) currently holds that resource. Transaction A waits for the resource to become available. However, suppose transaction B is not in a position to release the resource because it, in turn, is waiting on some resource held by transaction A. Then, both are deadlocked. The only way of breaking the deadlock is to cancel one of the transactions, thus releasing its resources.

An example of a deadlock: Two transactions running concurrently are modifying records within a single recoverable file, through the same file definition entry, as follows:

```
1   Transaction A  ---> EXEC CICS READ UPDATE   ---> record 1

2   Transaction B  ---> EXEC CICS DELETE        ---> record 2

3   Transaction A  ---> EXEC CICS WRITE         ---> record 2

4   Transaction B  ---> EXEC CICS READ UPDATE   ---> record 1

5   Transaction B  ---> EXEC CICS REWRITE       ---> record 1
```

Transaction A has acquired the record lock for *record 1*. Transaction B has similarly acquired the record lock for *record 2*. The transactions are then deadlocked because each wants to acquire the lock held by the other.

A transaction can have to wait for a resource while executing file control commands for several reasons:

- For SFS files, any record that is being modified is held in exclusive control for the duration of the request.
- If a transaction has modified a record in a recoverable file, SFS locks out access to that record by all transactions except the transaction that made the modification. This lock continues even after the request that performed the change has completed. This allows that transaction to continue to access and modify the same record. Other transactions must wait until the initial transaction releases the lock. The release occurs in either of two ways: by terminating the transaction or by issuing a syncpoint request.

Whether a deadlock actually occurs depends on the relative timing of the acquisition and release of the resources by different concurrent transactions. Application programs can continue to be used for some time before meeting circumstances that cause a deadlock; so it is important to recognize and allow for the possibility of deadlock early in the application program design stages.

Using the INVREQ condition to avoid deadlocks: CICS recognizes specific situations that can lead to a deadlock and stops them by returning the INVREQ condition to your application. For example, CICS does not allow a transaction to issue an EXEC CICS READ UPDATE, EXEC CICS WRITE, or EXEC CICS DELETE with RIDFLD request to a particular file if the transaction issued a previous EXEC CICS READ UPDATE to the same file, and the request has not yet been completed by an EXEC CICS REWRITE, EXEC CICS DELETE, or EXEC CICS UNLOCK command.

Tips to avoid deadlocks: CICS does not detect every situation that might cause a deadlock. You can avoid deadlocks by following these rules:

- All applications that update (modify) multiple resources should do so in the same order. For instance, if a transaction is updating more than one record in a file, it can do so in ascending key order. A transaction that is accessing more than one file should always do so in the same predefined sequence of files.
- An application that issues an EXEC CICS READ UPDATE command should follow it with an EXEC CICS REWRITE, EXEC CICS DELETE without RIDFLD, or EXEC CICS UNLOCK to release the position before doing anything else to the file.

See the *CICS Administration Reference*and the *CICS Application Programming Reference* for related information.

Queue services

Queues are sequential storage facilities that are global resources within either a single CICS region or a system of interconnected CICS regions. That is, queues, like files and databases, are not associated with a particular task. Any task can read, write or delete queues, and the pointers associated with a queue are shared across all tasks.

Two types of queues are provided by CICS: transient data queues and temporary storage queues. Although these names imply impermanence, CICS queues are permanent storage. Except for temporary storage queues kept in main storage, CICS queues persist across executions of CICS, unless explicitly discarded in a cold start. Both of these queue types are discussed and compared in the following sections.

Transient data queue services

CICS transient data queue services provide a generalized queueing facility. Data can be queued (stored) for subsequent internal or external processing. You can apply the following functions to selected data, specified in an application:
- Write data to a transient data queue (EXEC CICS WRITEQ TD)
- Read data from a transient data queue (EXEC CICS READQ TD)
- Delete an intrapartition transient data queue (EXEC CICS DELETEQ TD)

To use a transient data queue, you must specify the TD option. If the TD option is omitted, the command is assumed to be for a temporary storage queue. (See "Temporary storage queue services" on page 133 for details.)

Intrapartition destinations

Intrapartition destinations are data queues that are located on direct-access storage devices. These destinations are available for use with one or more programs running as separate tasks within the CICS region. Data directed to or from these internal destinations is called intrapartition data. Intrapartition transient data queues are efficient, but they must be created and processed entirely within CICS.

Typical uses of intrapartition data include:
- Message switching
- Broadcasting
- Database access
- Routing of output to several terminals (for example, for order distribution)
- Queuing of data (for example, for assignment of order numbers or priority by arrival)
- Data collection

Extrapartition destinations

Extrapartition destinations are queues (files) residing on any file system resource (disk, tape, printer, and so on) that are accessible by programs outside (or inside) the region. In general, sequential extrapartition destinations are used for storing and retrieving data outside the region. For example, one task can read data from a remote terminal, edit the data, and write the results to a file for subsequent processing in another region. Logging data, statistics, and transaction error messages are examples of data that can be written to extrapartition destinations. In general, extrapartition data created by CICS is intended for subsequent input to non-CICS programs. Data can also be routed to an output device such as a line printer.

CICS uses extrapartition transient data queues to handle standard system files. Any system file can be processed sequentially with CICS extrapartition transient data. Extrapartition transient data files are not recoverable.

Data directed to or from an external destination is called extrapartition data and consists of sequential records that are fixed-length or variable-length. The record format for an extrapartition destination must be defined in the TDD. (Refer to the *CICS Administration Reference* for details.)

You cannot delete an extrapartition queue.

CICS provides three different logical organizations for the byte-stream data stored in extrapartition queue data files:

- Fixed-length records
- Variable records
- Terminated records, which subdivide into:
  - Line-oriented records
  - Null-terminated records
  - Byte-terminated records

Fixed-length record files: Fixed-length record files partition the byte stream into adjacent, nonoverlapping blocks of bytes, all of the same length. The size of the block for a given queue can take any value between 1 and the maximum permitted record size (32767), but must remain fixed for all records in the file. Users must supply records of the chosen length in an EXEC CICS WRITEQ TD call and expect to receive a record of that length when performing an EXEC CICS READQ TD call.

Files whose length is not a multiple of the chosen record size are regarded as incorrectly formatted and can lead to IOERR conditions being raised if a task attempts to operate on the corresponding queue.

Variable-length record files: Variable-length record files partition the file byte stream into adjacent, nonoverlapping blocks of bytes of varying length, each of which is preceded by a two-byte length count used to determine the length of the following record. The record can be of any length between 1 and the maximum permitted record size. Users should not supply the length bytes in the record passed to an EXEC CICS WRITEQ TD call as it is written to the file by CICS. The record returned on an EXEC CICS READQ TD call does not contain the length bytes. Refer to the TDD **RecordLen** attribute description in the *CICS Administration Reference* for information on how the permitted record size is established.

The length count is stored in the file *high byte first*. Applications reading a file written by CICS determine the record length by reading the first byte, multiplying it by 256 and then adding the second byte. This should generate a value between 1

and the maximum permitted record size, giving the size of the following record in bytes. The next record can be obtained by reading this number of bytes from the file. If another record is stored in the file (end of file is not reached), the same procedure can be repeated to obtain subsequent records.

Files with negative or zero record lengths and files whose last record falls short of the last record length are regarded as incorrectly formatted and can lead to IOERR conditions being raised if a task attempts to operate on the corresponding queue.

Terminated record files: Terminated record files partition the file byte stream into adjacent, nonoverlapping blocks of bytes of varying length, each of which ends with a common terminating byte. The three subcategories correspond to different choices for this terminating byte:

- Line-oriented record files employ X'10', the ASCII newline character, as a terminator. (This is particularly useful as a format for queues containing readable text because it allows the file to be viewed/written using conventional text editors).
- Null-terminated record files employ X'00', the ASCII null character, as a terminator.
- Byte-terminated record files employ a user-defined byte in the range 0 to 255 as a terminator. (This actually subsumes the preceding categories; they are merely provided as convenience interfaces to support commonly employed formats).

Users should not append the terminator byte to the record passed to an EXEC CICS WRITEQ TD call as it is written to the file by CICS. The record returned on an EXEC CICS READQ TD call does not contain the terminator byte.

Normally, the terminator byte should not appear embedded anywhere in the record supplied in an EXEC CICS WRITEQ TD call. Any program which subsequently reads the file is not able to distinguish such embedded terminator bytes from the byte appended by the EXEC CICS WRITEQ TD call. (This applies in particular when the file is to be reused as the source for an input TD queue.) The result of embedding terminator bytes in the record data is an apparent fragmentation of the record into separate sub-records. In the case where a line-oriented queue is employed to write readable text, this can not be a restriction.

Indirect destinations

Intrapartition and extrapartition destinations can be used as indirect destinations. Indirect destinations provide some flexibility in program maintenance by allowing data to be routed to one of several destinations with changes being required only in the TDD, not in the program.

When the TDD has been changed, application programs continue to route data to the destination using the original symbolic name; however, this name is now an indirect destination that refers to the new symbolic name. Because indirect destinations are established by means of TDD entries, you do not need to be concerned with how this is done.

Triggered transaction initiation

CICS applications can initiate transaction processing in accordance with predefined parameters by using automatic transaction initiation (ATI), a queue management facility within CICS.

CICS provides the ATI facility for intrapartition destinations. The system administrator establishes a basis for the ATI facility by specifying a nonzero trigger

level and a triggered transaction identifier for a particular intrapartition destination in the TDD. When the number of entries (created by EXEC CICS WRITEQ TD commands issued by one or more programs) in the queue (destination) reaches the specified trigger level, the transaction specified in the definition of the destination is automatically initiated. The transaction must issue repetitive EXEC CICS READQ TD commands to deplete the queue.

Once the queue has been emptied, a new ATI cycle begins. That is, a new task is scheduled for initiation when the specified trigger level is again reached, whether or not execution of the earlier task has ended.

If an automatically initiated task does not empty the queue, access to the queue is not inhibited. The task can be normally or abnormally ended before the queue is emptied (that is, before a QZERO condition occurs in response to an EXEC CICS READQ TD command). Regardless of the facility type, the task is not started until the specified trigger level is reached. If the triggered transaction does not read from the TD queue, it is not re-initiated. If the trigger level of a queue is zero, no task is automatically initiated. If the trigger level is already exceeded because the last triggered transaction abended before clearing the queue, a task is scheduled the next time a record is written to the queue. To ensure that completion of an automatically initiated task occurs when the queue is empty, the application program should test for a QZERO condition rather than for some application-dependent factor such as an anticipated number of records. Only the QZERO condition indicates an emptied queue.

See the *CICS Application Programming Reference* for related information.

Temporary storage queue services

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

You can:

- Write data to a temporary storage queue (EXEC CICS WRITEQ TS).
- Update data in a temporary storage queue (EXEC CICS WRITEQ TS with the REWRITE and ITEM options).
- Read data from a temporary storage queue (EXEC CICS READQ TS).
- Delete a temporary storage queue (EXEC CICS DELETEQ TS).

If the TS option is omitted, the command is assumed to refer to temporary storage.

See “Error-handling services” on page 233 for a description of how temporary storage control commands respond to conditions that occur during execution.

Typical uses of temporary storage control

A temporary storage queue having only one record can be treated as a single unit of data that can be accessed using its symbolic name. Using temporary storage control in this way provides a typical *scratch pad* capability. Use the EXEC CICS READQ TS command with the ITEM(data area) option for access to this type of storage; failure to use this access method can cause the ITEMERR condition to be raised.

In general, use temporary storage queues of more than one record only when direct access or repeated access to records is necessary. For sequential files, use transient data queues.

Some uses of temporary storage queues follow:

- ***A suspend file***. Assume a data collection task is in progress at a terminal. The task reads one or more units of input and then allows the terminal operator to interrupt the process by some kind of coded input. If not interrupted, the task repeats the data collection process. If interrupted, the task writes its *incomplete* data to temporary storage and terminates. The terminal is now free to process a different transaction (perhaps a high-priority inquiry). When the terminal is available to continue data collection, the operator initiates the task in a *resume* mode, causing the task to recall its suspended data from temporary storage and continue as though it had not been interrupted.
- ***Preprinted forms.*** An application program can accept data to be written as output on a preprinted form. This data can be stored in temporary storage as it arrives. When all the data has been stored, it can first be validated and then transmitted in the order required by the format of the preprinted form.
- ***Data sharing***. Temporary storage is most suited to the task of data sharing because there is no need to predefine the facility (as is necessary for file usage). This is particularly true for scratchpad data.
- ***Paging through large quantities of data.*** You can read from a file in sections (for example, 10K), and put the sections into a temporary storage queue. This approach allows the display of only as much data as the screen can hold, and permits paging up and down. This approach is quicker than successive file access, especially if the data is being accessed remotely. This approach is not recommended if an update to the file is required.

Naming temporary storage queues

Temporary storage queues are not defined in the region database; instead, they are created the first time you write to that queue. When you write to a queue that does not already exist (and you have not specified the SYSID option), CICS creates a new, local queue that is based on a Temporary Storage Definitions (TSD) entry.

Temporary storage queue templates are defined in the region database and in the TSD. The purpose of the TSD is to determine the attributes given to temporary storage queues for that region. The TSD entry names (the TSD *key*) can be from one to eight characters long.

Temporary storage queues are identified by symbolic names that are assigned by the originating task. These names must be exactly eight characters long. When CICS creates a new queue, it tries to match this eight character name with a TSD entry name, and uses the template name that matches the most characters at the start of the queue name.

If no match is found, the queue automatically becomes a nonrecoverable, local queue with private access.

For example, a queue called FIRSTTSQ is to be created by an application. If there is a TSD entry with a key of FIRSTTSQ for the region, CICS uses this template when creating FIRSTTSQ. If there is no match, CICS searches for FIRSTTS, then FIRSTT, then FIRST, and so on.

Note: The TSD entry names that are exactly eight characters long are a special case. When an EXEC CICS WRITEQ TS command is received by a queue

having a name that exactly matches an eight character TSD name, the TSD entry becomes a temporary storage queue.

Temporary storage queue names are byte strings, not character strings. They can be made up from any bytes including binary zeros, and are not null terminated.

The CECI transaction pads names that are shorter than eight bytes with spaces when reading from or writing to a temporary storage queue. The name is not null terminated (unlike other names in CICS). The name has pattern matching rules associated with it that vary depending on whether the queue is local or remote. This is explained in the *CICS Administration Reference*.

If you write to a temporary storage queue from a transaction using a queue name that is less than eight characters, CICS reads eight bytes from the start of the queue name. This results in unexpected characters at the end of the queue name. Therefore, it is recommended that you always allocate eight bytes for the temporary storage queue name.

Temporary data can be retrieved by the originating task or by any other task using the symbolic name assigned to it. Specific items (logical records) within a queue are referred to by relative position numbers. To avoid conflicts caused by duplicate names, establish a naming convention. For example, the user identifier, terminal identifier, or transaction identifier could be used as a prefix or suffix to each programmer-supplied symbolic name.

The TSD entries can resolve to remote temporary storage queue templates, by entering values for the **RemoteSysId** and the **RemoteName** attributes in the TSD. Enter the sysid (up to four ASCII characters) of the remote region on which the queue is to reside in the **RemoteSysId** and the Communications Definitions (CD), and the name of the temporary storage template on that remote region in the **RemoteName**. The local temporary storage queue template name and the remote TSD entry name must be the same length. If you write to a queue that matches the local template, CICS replaces the template name at the start of the queue with the remote template name.

For example, you could have a local TSD entry with the name LOCALQ, defined with **RemoteName=REMOTQ** and with a **RemoteSysid** specified. If you write to a queue called LOCALQXX locally, the queue that is written to on the remote region is called REMOTQXX.

Deleting temporary storage queues

Temporary storage queues remain intact until they are deleted by the originating task or by any other task. Before deletion, they can be accessed any number of times. Even after the originating task is terminated, temporary data can be accessed by other tasks through references to the symbolic name under which it is stored.

You can use the EXEC CICS API commands on temporary storage queues, but not on TSD entries.

Note: The *CICS Administration Reference* provides complete descriptions of how to make changes to the resource definitions.

In the special case where a template name is exactly eight characters long, and the TSD entry becomes a temporary storage queue, you must use EXEC CICS DELETEQ TS to delete all items from the queue before you delete this TSD entry using **cicsdelete**.

Location of temporary data

Temporary data can be stored either in main storage (memory) or in auxiliary storage (a database file). Generally, use main storage if the data is needed for short periods of time; use auxiliary storage if the data is to be kept for long periods of time.

Data stored in auxiliary recoverable temporary storage queues is retained after CICS termination and can be recovered in a subsequent restart. Data stored in auxiliary nonrecoverable temporary storage queues is retained only across a normal shutdown, but not across an immediate shutdown or system failure unless a database is being used as the file manager.

Data stored in main storage is not retained across any type of shutdown and so cannot be recovered.

Queue aging

Temporary storage has a queue aging facility that automatically deletes queues that have not been accessed for a specified number of days. The number of days are defined with the Region Definitions (RD) **TSQAgeLimit** attribute. The storage occupied by these queues is freed and becomes available to temporary storage once again.

This feature is useful for temporary storage where queues are created dynamically when required. It is not needed for files or transient data queues that must be predefined before use.

Queue attributes

Temporary storage queues are created when the first write command is issued to them. Attributes (such as **RemoteSysId**, **RemoteName**, and **RecoverFlag**) are inherited from the longest matching queue template found in the TSD.

To use main storage for a queue, use the MAIN option on the EXEC CICS WRITEQ TS command that writes the first item to the queue. Temporary storage queues use auxiliary storage by default.

See the *CICS Application Programming Reference*and the *CICS Administration Reference* for related information.

Differences between transient data queues and temporary storage queues

There are two important differences between transient data queues and temporary storage queues:

- Transient data queue names must be defined in the Transient Data Definitions (TDD) before they are used by an application. You cannot define them arbitrarily at the time the data is created. Thus, transient data queues do not have the same dynamic characteristics as temporary storage queues.
- Transient data queues must be read sequentially, and each item can be read only once. That is, after a transaction reads an item, that item is removed from the queue and is not available to any other transaction. In contrast, items in temporary storage queues can be read either sequentially or directly (by item

number). They can be read any number of times and are never removed from the queue until the entire queue is purged.

These differences make transient data queues inappropriate for scratchpad data, but suitable for queued data, such as audit trails and output to be printed. In fact, for data that is read sequentially once, transient data queues are preferable to temporary storage for the following reasons:

- Items in a temporary storage queue can be changed; items in transient data queues cannot.
- Transient data queues are always written to a file. (There is no form of transient data queue that corresponds to main temporary storage.)
- You can define transient data queues so that writing items to the queue causes a specific transaction to be initiated (for example, to process the queue). Temporary storage queues have nothing that corresponds to this trigger mechanism, although you are able to use a START command to perform a similar function.
- Transient data queues have more varied recovery options than temporary storage queues. They can be physically or logically recoverable.
- Because the commands for intrapartition and extrapartition transient data are identical, you can switch easily between the internal CICS facility (intrapartition) and an external data set. To do this, you need only change the TDD, not your application programs. Temporary storage queues have no corresponding function of this kind.

Differences in queue behavior between SFS and DB2

Although all CICS queues behave as fully recoverable when DB2 is used as the file manager, the recovery attributes, as defined in the Temporary Storage Definitions (TSD) and Transient Data Definitions (TDD) entries for the queues, are not changed. This determines which file is used to store data for each queue. This is important when migrating between an SFS and DB2 as it means that queues defined as nonrecoverable become nonrecoverable when moving from DB2 to SFS.

Journal services

CICS provides facilities for creating and managing journals during CICS processing. A *journal* is a set of special-purpose sequential files. Journals can contain any and all data the user needs to facilitate subsequent reconstruction of events or data changes. For example, a journal might act as an audit trail, a change-file of database updates and additions, or a record of transactions passing through the system (often called a *log*). Each journal can be written from any task.

CICS journals provide an alternative to extrapartition transient data queues, although only for output files. Each journal command specifies operation characteristics (for example, synchronous or asynchronous), whereas extrapartition operations are governed entirely by the Transient Data Definitions (TDD). CICS journals have slightly higher overhead than extrapartition transient data queues.

Journal services commands are provided to allow the application programmer to:

- Create a journal record (JOURNAL)
- Synchronize with (wait for completion of) journal output (WAIT JOURNAL).

The responses of a journal control command to conditions that occur during execution are discussed in “Error-handling services” on page 233.

CICS journaling

Journals are fundamental to the recoverability of transactions. In particular, CICS uses the system journal to log transaction commit processing and syncpoint data so that CICS can recover all necessary recoverable resources in the event of a CICS or a transaction failure.

Before considering journaling in detail, we need to review the different facets of CICS logging and recovery in order to clarify the reasons for logging.

Dynamic transaction backout

If CICS abnormally terminates a transaction, then all changes made by the transaction to a recoverable resource, such as a recoverable temporary storage queue, must be backed out to the state existing before the transaction started. This is known as *dynamic transaction backout (DTB)*.

Recovery after a system abnormally terminates

Recovery after a system abnormally terminates ensures that all recoverable resources and all prepared transactions are restored to their pre-failure state, before the system resumes normal operation.

For CICS this is a special case of the more general problem of recovering the state of partially finished transactions. In principle, CICS records any change made to a recoverable resource in the system journal as part of two-phase commit processing so that the change can be committed from that point onwards. It therefore follows that, during normal operation, CICS only writes to the system journal enabling CICS transactions to uphold their guarantees. CICS neither reads nor performs processing on the system journal during normal operation.

During recovery processing (at start-up after your CICS system abnormally terminates), CICS processes the system journal to re-prepare all transactions that were in-flight at the time of the crash. CICS recovery processing reads the system journal to obtain a list of active transactions, and subsequent processing plays back the appropriate records.

To help speed up recovery, CICS writes checkpoint records to the system journal with CICS transaction and region status information at appropriate intervals. Without this information, CICS recovery processing would need to search through the whole system journal to locate the necessary records.

CICS journaling

CICS provides the following journaling facilities:

- A system journal
- User journals numbered 1 through 99

At startup CICS initializes the CICS system journal followed by any user journals. CICS abnormally terminates if any of the following conditions occur:

- It cannot open the system journal
- It cannot write to the system journal
- It cannot open a crucial user journal

During a system shutdown, the output from all journals is synchronized, and they are then closed.

You use the EXEC CICS JOURNAL and EXEC CICS WAIT JOURNAL commands in your application to write to a journal and to synchronize the journals you are using.

See the *CICS Administration Guide*, the *CICS Administration Reference*, and the *CICS Application Programming Reference* for related information.

Journal records

Data can be written to any journal specified in the Journal Definitions (JD), which define the journals available during a particular CICS execution. Each journal is identified by a number known as the journal identifier. This number can range from 1 through 99.

Journal 1 is used as the system log on other CICS products. Therefore, consider starting from journal 2 for applications that are possibly used on other CICS products.

When a journal record is built, the data is moved to the journal buffer area. All buffer space and other work areas needed for journal operations are acquired and managed by CICS. The user task supplies only the data to be written to the journal.

Journal records are written in a special format. Each record has a system prefix, an optional user-built prefix, and a variable record length. Journals are normally opened for output; in this mode, many users can share the journal. CICS serializes the writes and helps ensure data integrity.

CICS does not provide online facilities for reading journals.

Each journal record begins with a standard length field, a user-specified identifier, and a system-supplied prefix. This data is followed in the journal record by any user-supplied prefix data (optional), and finally by the user-specified data. Journal control services are designed so that the application programmer requesting output services need not be concerned further with the detailed layout and precise contents of journal records. The programmer needs to know only which journal to use, what user data to specify, and what user-identifier to supply.

Journal records are written to extrapartition TD queues in variable-length record format. The format of the journal records is specified below in order to allow you to read and process them in user recovery routines.

Table 30. Journal record format

Byte Offset	Length	Format	Description
0	2	integer	Length of record - not including this field
2	2	integer	Journal ID
4	2	integer	Year that record was created
6	2	integer	Month that record was created
8	2	integer	Day of month that record was created
10	2	integer	Hour that record was created
12	2	integer	Minute that record was created
14	2	integer	Second that record was created
16	2	string	Journal type (JTYPEID on JOURNAL command)
18	2	integer	User prefix length (PFXLENG on JOURNAL command)
20	2	integer	User data length

Table 30. Journal record format (continued)

Byte Offset	Length	Format	Description
22	n	bytes	Prefix data (PREFIX on JOURNAL command)
22+n	m	bytes	User data

The definition of the format type displayed in this table is as follows:

integer
is the internal representation of an integer number. For example, the number 2 is stored as the bit pattern 0000 0000 0000 0010.

string is a character string consisting of alphanumerics.

bytes is a string of bytes as provided by the user on the JOURNAL command.

Note: The implementation of journal records in IBM CICS for Windows and CICS on Open Systems as extrapartition TD queues is not the same as in other CICS platforms. This does not affect the API. However, you can see transient data messages, abend codes, and symrecs when using journal records.

See the *CICS Application Programming Reference*, the *CICS Administration Reference*, and the *CICS Administration Guide* for related information.

Journal output synchronization

When a journal record is created by issuing the JOURNAL command with the WAIT option, the requesting task can wait until the output has been completed. By specifying that this should happen, the application programmer ensures that the journal record is written on the external storage device associated with the journal before processing continues; the task is said to be *synchronized* with the output operation.

The application programmer can also request *asynchronous* journal output. This causes a journal record to be created in an operating system file system buffer and, optionally, initiates the data output operation from the buffer to the external device, but allows the requesting task to retain control and thus to continue with other processing. The task can check and wait for output completion (that is, synchronize) at some later time by issuing the WAIT JOURNAL command.

If possible, journal transactions should write asynchronously to minimize task waits. However, if journal records must be physically written before end of task to maintain integrity, you must use a synchronous write. If there are several journal writes, the transaction should use asynchronous writes for all but the last logical record, so that the logical records for the task are written with a minimum number of physical I/O operations and only one wait. Use journals for audit purposes where the **CrucialFlag** attribute in the Journal Definitions (JD) ensures that all writes succeed, or in situations where a significant proportion of writes can be asynchronous (buffered).

The basic process of building journal records in file system buffers continues until one of the following events occurs:

- A request specifying the WAIT option is made (from any task) for output of a journal record.
- Task completion.

- File system buffers are flushed (this takes place periodically and when file system buffer shortages occur).

When any one of these occurs, all journal records present in the buffer, including any deferred output resulting from asynchronous requests, are written to auxiliary storage as one block.

The advantages that can be gained by deferring journal output are:
- Transactions can get better response times by waiting less
- The load of physical I/O requests on the host system can be reduced

However, these advantages are achievable only at the cost of more buffer space and greater programming complexity. It is necessary to plan and program to control synchronizing with journal output. Additional decisions that depend on the data content of the journal record and how it is to be used must be made in the application program. In any case, the full benefit of deferring journal output is obtained only when the load on the journal is high.

The STARTIO option is used in EXEC CICS JOURNAL commands to request that the journal output operation be initiated immediately. This is the default behavior for CICS, as a write operation to filesystem buffers is always performed immediately, regardless of the presence or absence of the STARTIO option. Control is returned to the requesting program immediately after the record is buffered, unless the WAIT option is used, in which case the return is delayed until after the journal record has become permanent. The WAIT option should not be used unnecessarily because, if every journal request uses WAIT, no improvement over synchronous output requests (in terms of reducing the number of physical I/O operations) is possible.

If there is insufficient file system space available to write the journal record at the time of the request, the NOJBUFSP condition occurs. If no HANDLE CONDITION request is active for this condition, the requesting task loses control and is suspended until space becomes available again and the journal record has been written to the journal.

If the requesting task is not willing to lose control (for example, if some housekeeping must be performed before other tasks get control), a HANDLE CONDITION command should be issued. If the NOJBUFSP condition occurs, no journal record is built for the request, and control is returned directly to the requesting program at the location provided in the HANDLE CONDITION request. The requesting program can perform any housekeeping needed before reissuing the journal output request.

When using journal 1, which is the system journal on other CICS systems, it is advisable to specify a journal type identifier (JTYPEID) to distinguish between user journal record types and system journal record types.

See the *CICS Administration Reference* and the *CICS Application Programming Reference* for related information.

Relational database services

CICS allows access to relational databases that provide a programmable interface through embedded SQL commands in COBOL, C, C++, or PL/I programs. For details of how to use EXEC SQL commands, refer to the SQL documentation supplied with your relational database.

Note: TXSeries CICS on Windows allows access to Microsoft SQL server that provides a programmable interface through ODBC API in C, IBM VisualAge COBOL programs.

If the relational database management system (RDBMS) you are using is not compliant with the X/Open XA standard, or you are not accessing that RDBMS with the XA interface, you cannot use two-phase commit between the resources that are coordinated by CICS.

Note: Some databases mentioned in this section are possibly not supported by your CICS product. Refer to the information about supported products in the *Release Notes*.

SQL restrictions and relational database services

Restrictions apply when using SQL to access relational databases with the IBM CICS for Windows or CICS on Open Systems API. These restrictions apply to XA enabled and non-XA enabled databases.

SQL restrictions for XA-enabled relational databases

If your relational database is compliant with the X/Open XA interface and your CICS region is configured to use it, you must use CICS facilities for transaction control rather than the relational database's facilities. This precludes the use of database COMMIT and ROLLBACK statements. Instead, code with the EXEC CICS SYNCPOINT or EXEC CICS SYNCPOINT ROLLBACK commands. The EXEC CICS SYNCPOINT command (or the implicit end of transaction syncpoint) ensures that the relational database system and the region are updated using the two-phase commit protocol.

For DB2, the default value of the collection identifier (schema name) of the SQL package is inherited from the BIND/PREP operation that was used to create the package. For CICS applications, this schema name is CICS. If you are using SQL to access objects in the DB2 database that do not belong to the CICS schema, you need to explicitly qualify the object name with the schema name. For example:

```
EXEC SQL SELECT COL1 from NOTCICS.MYTABLE
```

In this example, MYTABLE is the object that is being accessed by CICS. This object resides outside of the CICS schema in a schema named NOTCICS.

Do not use the EXEC SQL SET CURRENT PACKAGESET NOTCICS command to enable access to this object without the schema name identifier. Using this approach causes all access to the database to default to the new schema name, even access to the TS or TD queues if DB2 was being used as the file control for the CICS region. The PACKAGESET identifier MUST then be reset back to CICS before you access any TS or TD queues.

Therefore, it is recommended that you avoid the use of the SQL SET CURRENT PACKAGESET call. Instead, qualify the object to be accessed with the schema name when the object resides outside the CICS schema.

DB2 allows a CICS transaction to CONNECT to any database that has been defined as an XA resource in the Product Definitions (XAD). If you have only one database defined, then CICS establishes implicit connections to the database with the XA interface. For further information on using CONNECT in an XA environment with DB2, see the appropriate documentation for your version of DB2.

Do not use DB2's CONNECT RESET command in an XA environment. Use of this command in a CICS XA application can cause the loss of the XA connection. The CONNECT RESET command will work in non-XA CICS applications. The behavior of the command differs based on the type of SQL CONNECT command that is used in the application. Consult DB2 or UDB documentation for details on the behavior of CONNECT type 1 and CONNECT type 2 commands.

DB2 programs need to be compiled with the NOSQLINIT option. The following sample application makefile, located in the CICS examples directory, shows how this can be done:

```
xa/uxa1_db2cob.mk
```

With the exception of DB2, do not use any relational database commands to connect to or to disconnect from the database. CICS establishes implicit connections to the database via the XA interface.

Do not use the Microsoft SQL Server CONNECT statement in an XA enabled CICS environment. The set of all possible Microsoft SQL Server SQL connections to be used from CICS applications in a specific region should be defined in the Product Definitions (XAD) stanza for that region. CICS establishes long running connections to the Microsoft SQL Server during application initialization. Each connection should be identified by a unique connection name defined in the XAD XAOpen string. You can switch between connections in your application by using the following call:

```
EXEC SQL SET CONNECTION name
```

All connections are automatically enlisted in all CICS transactions.

Other than these statements, almost any SQL statement is legal in an XA-enabled CICS environment. A few statements can behave differently or even return errors within a CICS transaction. For more information on such statements, it is strongly recommended that you review the appropriate relational database literature.

SQL restrictions for non-XA enabled relational databases

In this environment, you must specifically code the SQL COMMIT WORK command to commit SQL resources that you have updated and explicitly code EXEC CICS SYNCPOINT commands to commit updates to CICS resources (or let this happen implicitly at the end of the transaction). If one of these succeeds while the other does not, you must use manual means to make the two systems consistent.

In addition, your transaction needs to connect explicitly to the RDBMS prior to making any SQL calls and must explicitly close the connection to the RDBMS after all SQL calls have been made. This last activity should be conducted with care as the disconnection process can involve multiple steps on some RDBMS.

For example:

Disconnecting from an Informix V5 database after running a non-XA enabled C transaction involves the use of all 3 of the following calls (shown in C):

```
EXEC SQL COMMIT WORK;
EXEC SQL CLOSE DATABASE;
sqlexit();
```

Disconnecting from an Informix V7 database after running a non-XA enabled COBOL transaction involves the use of all 3 of the following calls (shown in COBOL):

```
EXEC SQL ROLLBACK WORK END-EXEC.
EXEC SQL CLOSE DATABASE END-EXEC.
CALL ECO-SQE USING <eco-sqe-status>.
```

where the variable *<eco-sqe-status>* is defined as:

```
<eco-sqe-status> PIC S(9).
```

See Informix manuals for further information.

The COMMIT WORK statement commits all modifications made to the database since the beginning of a transaction. The CLOSE DATABASE statement closes the database but does not terminate the associated database server process, the latter is achieved through the sqlexit call.

Restrictions for linking non-XA enabled relational databases: CICS transactions that include EXEC SQL statements to non-XA enabled databases and use EXEC CICS LINK or EXEC CICS XCTL calls must be coded carefully. Database connections cannot be conveyed across the EXEC CICS LINK or EXEC CICS XCTL. You must disconnect from the database before these calls and reconnect in the called program.

This restriction also extends to EXEC CICS ABEND handling when working with non-XA enabled relational databases. You cannot use an abnormal termination exit program in this environment, because a database connection could not be conveyed to that exit program. If a C application is abnormally terminated, it is possible that the database connection and associated locks are not released. In this situation, it is necessary to shutdown and warm start the CICS region for database resources to be freed. COBOL applications can make use of the EXEC CICS HANDLE ABEND command with the LABEL option to free database resources and servers when abending a non-XA transaction.

CICS for Windows:

The exit routine for a COBOL non-XA enabled transaction running against DB2 would include:

```
EXEC SQL ROLLBACK WORK
EXEC SQL CONNECT RESET
EXEC SQL ROLLBACK
EXEC SQL DISCONNECT
```

If you are coding in C, ensure that all EXEC CICS ABEND calls are preceded by RDBMS calls to free the database. For example, the following would be included before each EXEC CICS ABEND call for DB2 non-XA programs:

```
EXEC SQL ROLLBACK WORK
EXEC SQL CONNECT RESET
```

CICS on Open Systems:

The exit routine for a COBOL non-XA enabled transaction running against the following databases would include:

For Informix:

```
EXEC SQL ROLLBACK WORK END-EXEC
EXEC SQL CLOSE DATABASE END-EXEC
CALL ECO-SQE
```

For Oracle:

```
EXEC SQL ROLLBACK
EXEC CICS RELEASE
```

For DB2:

```
EXEC SQL ROLLBACK WORK
EXEC SQL CONNECT RESET
EXEC SQL ROLLBACK
EXEC SQL DISCONNECT
```

If you are coding in C, ensure that all EXEC CICS ABEND calls are preceded by RDBMS calls to free the database. For example, the following would be included before each EXEC CICS ABEND call:

For Informix non-XA C programs:

```
EXEC SQL ROLLBACK WORK;
EXEC SQL CLOSE DATABASE;
sqlexit();
```

For Oracle non-XA programs:

```
EXEC SQL ROLLBACK
EXEC CICS RELEASE
```

For DB2 non-XA programs:

```
EXEC SQL ROLLBACK WORK
EXEC SQL CONNECT RESET
```

An example transaction for XA-enabled relational databases

This simple example illustrates some of the principles involved in writing a CICS transaction that accesses an XA-enabled relational database (CICS on Open Systems) and a Microsoft SQL Server database (CICS for Windows).

The example's source, map, makefile, sql, and README file are shipped in the following directory of your CICS development environment:

prodDir/src/examples/xa

Note: Refer to xii for a description of how *prodDir* is used to represent the product pathname.

See the following for more details on how to configure, build, and install the transaction:

prodDir/src/examples/xa/uxa1.README

Example for CICS on Open Systems

The transaction is written in C and runs against a database of cheeses, whose entries are as follows:

Table 31. Example transaction column definitions

Name	Supplier	Supplier Address	Order Quantity
stilton	Northern Dairies	Richmond (Yorks)	23

Table 31. Example transaction column definitions (continued)

Name	Supplier	Supplier Address	Order Quantity
wensleydale	Northern Dairies	Richmond (Yorks)	14
cheddar	Gorge Supplies	Wells (Somerset)	10
brie	La Rochelle Importers	Dover (Kent)	45
camembert	La Rochelle Importers	Dover (Kent)	33
edam	East Anglian Distribution	Felixstowe (Suffolk)	43
cheshire	Manchester Dairies	Manchester	18
gouda	East Anglian Distribution	Felixstowe (Suffolk)	44
red leicester	Midlands Dairies	Leicester (Leics)	34
boursin	La Rochelle Importers	Dover (Kent)	53

This sample transaction comprises 4 panels. These four panels are used to:
- View a cheese order (see panels 1 and 2)
- Update a cheese order (see panel 3)
- Review cheese order messages (see panel 4)

These panels are described in the BMS map source in:
`$CICS/src/examples/xa/uxa1.bms`

The following transaction, UXA1, allows the user to query the number of cheeses of a particular type that have been ordered, and to update this number if required. Messages are displayed on the fourth panel. Refer to the table that follows the example for a description of the code.

The following example code has been written to support Oracle. It is necessary to define the particular database against which the transaction runs (see line 16). Other than line 16, the source is identical.

Refer to the uxa1.README file for the actual name of this example. Refer to Table 32 on page 149 for a description of this program:

```
1 /*
2 * Transaction: UXA1
3 * Program:     UXA1PROG
4 * Mapset:      UXA1
5 */
6
7 #include <stdio.h>
8 #include <cics_packon.h>
9 #include "uxa1.h"
10 #include <cics_packoff.h>
11
12 #define NOCHEESE "There is no such cheese in the table"
13 #define UPDATECHEESE "The cheese table was successfully updated"
14
15 /* define either ORA7 or INF5 here */
16 #define ORA7
17
18 #ifdef ORA7
19 #define SQLNOTFOUND 1403
20 #endif
21
```

```
22 EXEC SQL INCLUDE sqlca;
23
24 int rcode;
25
26 EXEC SQL BEGIN DECLARE SECTION;
27
28 varchar name[15];
29 char supplier[30];
30 char supplier_address[30];
31 int order_quantity;
32
33 EXEC SQL END DECLARE SECTION;
34
35 main()
36 {
37      char errmsg[400];
38      char qmsg[400];
39      short mlen;
40
41      EXEC SQL WHENEVER SQLERROR GOTO :errexit;
42
43 /*   Get addressability for EIB */
44
45      EXEC CICS ADDRESS EIB(dfheiptr);
46
47 /*   Write record to CICS temporary storage queue  */
48
49      sprintf(qmsg, "%s", "Running Transaction UXA1");
50      mlen = strlen(qmsg);
51      EXEC CICS WRITEQ TS QUEUE("TEMPXAQ1") FROM(qmsg) LENGTH(mlen)\
                                              RESP(rcode);
52      if (rcode != DFHRESP(NORMAL))
53         EXEC CICS ABEND ABCODE("X000");
54
55      /* Send the first map */
56
57      EXEC CICS SEND MAP("PANEL1") MAPSET("UXA1") FREEKB ERASE RESP(rcode);
58      if (rcode != DFHRESP(NORMAL))
59         EXEC CICS ABEND ABCODE("X001");
60
61      /* Receive the response */
62
63      EXEC CICS RECEIVE MAP("PANEL1") MAPSET("UXA1") RESP(rcode);
64      if (rcode != DFHRESP(NORMAL))
65         EXEC CICS ABEND ABCODE("X002");
66
67 /* Select a record from the table based on user input */
68
69 #ifdef ORA7
70      sprintf(name.arr, "%s", panel1.panel1i.newnamei);
71      name.len = strlen(name.arr);
72 #endif
73 #ifdef INF5
74      sprintf(name, "%s", panel1.panel1i.newnamei);
75 #endif
76
77
78      EXEC SQL SELECT name, supplier, supplier_address, order_quantity
79      into
80         :name, :supplier, :supplier_address, :order_quantity
81      FROM CHEESE
82      WHERE NAME = :name;
83
84      /* Handle "no rows returned" from SELECT */
85
86      if (sqlca.sqlcode == SQLNOTFOUND)
87      {
```

```
88          sprintf(panel4.panel4o.messageo, "%s", NOCHEESE);
89          EXEC CICS SEND MAP("PANEL4") MAPSET("UXA1") FREEKB ERASE \
                                                       RESP(rcode);
90          if (rcode != DFHRESP(NORMAL))
91              EXEC CICS ABEND ABCODE("X009");
92
93          EXEC CICS SEND CONTROL FREEKB;
94          EXEC CICS RETURN;
95      }
96
97 /* Fill in and send the second map */
98
99 #ifdef ORA7
100      sprintf(panel2.panel2o.nameo, "%s", name.arr);
101 #endif
102 #ifdef INF5
103      sprintf(panel2.panel2o.nameo, "%s", name);
104 #endif
105      sprintf(panel2.panel2o.supplo, "%s",supplier);
106      sprintf(panel2.panel2o.addresso, "%s", supplier_address);
107      sprintf(panel2.panel2o.ordero, "%d", order_quantity);
108
109      EXEC CICS SEND MAP("PANEL2") MAPSET("UXA1") FREEKB ERASE RESP(rcode);
110      if (rcode != DFHRESP(NORMAL))
111         EXEC CICS ABEND ABCODE("X003");
112
113      /* Receive the response */
114
115      EXEC CICS RECEIVE MAP("PANEL2") MAPSET("UXA1") RESP(rcode);
116      if (rcode != DFHRESP(NORMAL))
117         EXEC CICS ABEND ABCODE("X004");
118
119      if (panel2.panel2i.questi == 'y')
120      {
121
122         /* Send the third map   */
123
124        EXEC CICS SEND MAP("PANEL3") MAPSET("UXA1") FREEKB ERASE \
                                                       RESP(rcode);
125        if (rcode != DFHRESP(NORMAL))
126            EXEC CICS ABEND ABCODE("X005");
127
128        /* Receive the response */
129
130        EXEC CICS RECEIVE MAP("PANEL3") MAPSET("UXA1") RESP(rcode);
131        if (rcode != DFHRESP(NORMAL))
132            EXEC CICS ABEND ABCODE("X006");
133
134          /* Update the Database */
135
136          order_quantity = atoi(panel3.panel3i.newordi);
137
138          EXEC SQL UPDATE cheese
139          set order_quantity = :order_quantity
140          where name = :name;
141
142          /* Write a record to the temporary queue */
143
144          sprintf(qmsg, "%s", "The cheese table was updated");
145          mlen = strlen(qmsg);
146          EXEC CICS WRITEQ TS QUEUE("TEMPXAQ1") FROM(qmsg) LENGTH(mlen) \
                                                     RESP(rcode);
147          if (rcode != DFHRESP(NORMAL))
148             EXEC CICS ABEND ABCODE("X010");
149
150      }
151      else
```

```
152      {
153
154      /* The user does not wish to update so free the keyb'd & return  */
155
156         EXEC CICS SEND CONTROL ERASE FREEKB;
157         EXEC CICS RETURN;
158
159      }
160
161      /* Commit the update */
162
163      EXEC CICS SYNCPOINT RESP(rcode);
164      if (rcode != DFHRESP(NORMAL))
165         EXEC CICS ABEND ABCODE("X011");
166
167      /* Send the fourth map confirming successful update  */
168
169      sprintf(panel4.panel4o.messageo, UPDATECHEESE);
170      EXEC CICS SEND MAP("PANEL4") MAPSET("UXA1") FREEKB ERASE RESP(rcode);
171      if (rcode != DFHRESP(NORMAL))
172         EXEC CICS ABEND ABCODE("X007");
173
174      /* free the keyb'd & return  */
175
176      EXEC CICS SEND CONTROL FREEKB;
177      EXEC CICS RETURN;
178
179 errexit:
180
181      /* Handle general errors */
182
183      EXEC SQL WHENEVER SQLERROR CONTINUE;
184
185 #ifdef ORA7
186      sprintf(errmsg, "%.60s\n", sqlca.sqlerrm.sqlerrmc);
187 #endif
188 #ifdef INF5
189      rgetmsg(sqlca.sqlcode, errmsg, sizeof(errmsg));
190 #endif
191      strncpy(panel4.panel4o.messageo, errmsg, 60);
192      sprintf(panel4.panel4o.codeo, "%d", sqlca.sqlcode);
193
194      /* Send the fourth map with appropriate message */
195
196      EXEC CICS SEND MAP("PANEL4") MAPSET("UXA1") FREEKB ERASE \
                                                    RESP(rcode);
197      if (rcode != DFHRESP(NORMAL))
198         EXEC CICS ABEND ABCODE("X008");
199
200      /* Rollback the transaction */
201
202      EXEC CICS SYNCPOINT ROLLBACK;
203      EXEC CICS SEND CONTROL FREEKB;
204      EXEC CICS RETURN;
205 }
```

Table 32. Sample transaction for an XA-enabled database

Lines	Description
1 to 34	As well as including the SQL Communications area (for database error handling) and database host variable declarations, this section includes a file named uxa1.h, which is the symbolic map file generated by running the cicsmap command against the BMS source.
35 to 46	The WHENEVER statement is used to pass control to a generic error handler at line 179.

Table 32. Sample transaction for an XA-enabled database (continued)

Lines	Description
47 to 66	A record is written to a recoverable CICS Temporary Storage queue. The first panel is sent to the user and the name of the cheese that is input is returned in the variable “panel1.panel1i.newnamei”.
67 to 83	A SELECT is done against the database using the name of the cheese input by the user in the WHERE clause.
84 to 96	This section manages the situation where the SELECT fails to find any rows that match the name of the cheese input by the user.
97 to 121	The second panel is filled using data returned by the SELECT and is sent to the user. The user responds by keying “y” if he wishes to update the record displayed.
122 to 133	The third panel is sent to the user querying the new order quantity needed for the particular cheese.
134 to 150	The database is updated with the new quantity of cheeses to be ordered and, if the update is successful, a record is written to the CICS temporary storage queue.
151 to 160	This section handles the case where the user has chosen not to update the database.
161 to 166	A CICS SYNCPOINT is executed to commit the update.
167 to 173	The fourth panel is sent to the user to confirm that the database has been successfully updated.
174 to 178	Return from the application.
179 to 205	If an error is detected, then the sqlca sqlcode and corresponding message are passed to the user with the fourth panel.

Example for CICS for Windows

```
/*
* Transaction: UXA1
* Program:     UXA1PROG
* Mapset:      UXA1
*/
#define DLLIMPORT __declspec(dllimport)
#define DLLEXPORT __declspec(dllexport)
#define CDECL __cdecl
#include <stdio.h>
#include <stdlib.h>
#include "uxa1.h"
#define NOCHEESE "There is no such cheese in the table"
#define UPDATECHEESE "The cheese table was successfully updated"
#define SQLNOTFOUND 100
EXEC SQL INCLUDE sqlca;
int rcode;
EXEC SQL BEGIN DECLARE SECTION;
char    co[30];
char name[15];
char supplier[30];
char supplier_address[30];
int order_quantity;
EXEC SQL END DECLARE SECTION;
DLLEXPORT CDECL main()
{
char errmsg[400];
char qmsg[400];
short mlen;
/* Initialise the maps */
memset (&panel1, 0x00, sizeof(panel1));
```

```
memset (&panel2, 0x00, sizeof(panel2));
memset (&panel3, 0x00, sizeof(panel3));
memset (&panel4, 0x00, sizeof(panel4));
/*-----------
 * Change the name server1 to a connection name you defined
 * in your CICS MS SQL Server XAD stanza XA_OPEN string.
 *-----------*/
sprintf(co, "server1");
EXEC SQL SET CONNECTION
```

Writing a CICS application program by using an ODBC API that accesses a Microsoft SQL Server database (CICS for Windows only)

This section shows some of the principles that you use when you write a CICS transaction under an XA-enabled environment, and the environment accesses an Microsoft SQL Server database through the ODBC API/ODBC Driver Library. You can write CICS application programs by using ODBC API/ODBC Library in C , IBM VisualAge COBOL, or Micro Focus NetExpress COBOL. The following sections explain how to use these methods to write programs. Each section gives an example.

- "Writing Programs using ODBC API/ODBC Driver Library in C"
- "Writing Programs using ODBC API/ODBC Driver Library in IBM VisualAge COBOL" on page 153
- "Writing Programs using E-SQL API/ODBC Driver Library in Micro Focus NetExpress COBOL" on page 156

Writing Programs using ODBC API/ODBC Driver Library in C

This procedure describes how to use the ODBC API under an XA environment to write CICS application programs that can access data that resides in a Microsoft SQL Server database.

Prerequisite tasks and conditions: If you want to use Microsoft SQL Server, CICS for Windows requires MS SQL Workstation or Server to be installed and configured according to your Microsoft SQL documentation. When installing MS SQL, ensure that Distributed Transaction Coordinator (DTC) support is included.

To build CICS/ODBC applications, which integrate CICS and Microsoft SQL Server, you must have the MSDTC and ODBC components of the Microsoft SQL Server installed on your development systems. Also, you must have Microsoft SQL Server ODBC API Library for C and Microsoft Visual C++ to enable you to build ODBC applications for C.

For more information refer to the *CICS Administration Guide for Windows Systems*.

Obtaining ODBC connection handles: In an XA environment, connections that are made to the Microsoft SQL Server database by using ODBC are made through the SwitchLoad file. The SwitchLoad file makes the necessary connections to the database through the SQLConnect ODBC API. The SQLConnect ODBC API returns an ODBC connection handle that is maintained in a CICS internal structure. The connection handle is unique for every XA Definition that is in the XAD.stanza.

CICS applications can access the list of ODBC connection handles through a CICS-supplied API. The syntax of that API is:

```
EXEC CICS ASSIGN ODBCHNDLLIST(data-area) ODBCLISTLEN(data-area)
```

Refer to the *CICS Application Programming Reference* for more information about this API.

Using a sample CICS C application: The source, map, makefile, and README file for the example are shipped in this directory of your CICS development environment:

```
ProdDir/src/examples/xa
```

In the following example, ODBC APIs have been used to write the source files in C. Application programmers can refer to these files to determine how to use ODBC APIs to write application programs and also to determine how to use the EXEC CICS ASSIGN ODBCHNDLLIST API to obtain ODBC connection handles.

- cheese_mssql.ccs(source file)
- cheese_mssql.mk (Makefile to compile the source)
- uxa1.bms (map file)

This portion of the cheese sample source file that is written in C shows how to use EXEC CICS ASSIGN ODBCHNDLLIST API to obtain a list of ODBC connection handles.

```
int GetDataFromDB(SQLHSTMT,char *,struct CheeseTbl **);
int UpdateCheeseDB(SQLHSTMT,int,char *);
SQLHDBC GetODBCHandleFromList(struct ODBC_ConArray [],int);
struct ODBC_ConArray Buffer[MAX_ENTRIES];

main()
{

    SQLHSTMT    hstmt;
    SQLHDBC     ODBCHandle;
    int         CId,ret,retcode;
    char        errmsg[400];
    short       mlen,Length;
    char        name[15];
    struct      CheeseTbl *CheeseData=NULL;
    int        order_quantity;
    char       *UPDATECHEESE,*NOCHEESE,*ODBCCON=NULL;

        /* Initialise the maps */
    memset (&panel1, 0x00, sizeof(panel1));
    memset (&panel2, 0x00, sizeof(panel2));
    memset (&panel3, 0x00, sizeof(panel3));
    memset (&panel4, 0x00, sizeof(panel4));

    CId=1;
    memset(Buffer,NULL,MAX_ENTRIES*sizeof(struct ODBC_ConArray));
    EXEC CICS ASSIGN ODBCHNDLLIST(Buffer) ODBCLISTLEN(Length);
    ODBCHandle = GetODBCHandleFromList(Buffer,CId);
    if(ODBCHandle == 0)
    {
        ODBCCON="Fetching Connection Handle Failed";
        EXEC CICS SEND TEXT FROM(ODBCCON) LENGTH(37);
        EXEC CICS RETURN;
    }


SQLHDBC GetODBCHandleFromList(struct ODBC_ConArray Buffer[],int ConId)
 {
     int i=0;
     if(Buffer[i].ConId != 0)
    {
          for(i=0;Buffer[i].ConId != 0;i++)
         {
```

```
                if(Buffer[i].ConId == ConId)
               {
                          return (SQLHDBC)Buffer[i].DBHandle;
               }
          }
    }
   return 0;
}
```

In this portion of the example code, the EXEC CICS ASSIGN ODBCHNDLLIST API fills the buffer that is passed by the program. It fills the buffer with an array of structures that contains the List of ODBCHandles, Type of the each ODBCHandle, and the ConnectID.

The ConnectID member identifies the database to which the connection handle refers. The user sets this ConnectID in the XAOpen string of XAD.stanza for the XAD product definition. Users use this ConnectID in the application, to obtain the ODBC connection handle for each database that they want to access.

For more information about the ODBC_ConnArray structure, refer to the header file `cics_odbc.h`, which you can see in the directory Drive:\opt\cics\include.

Writing Programs using ODBC API/ODBC Driver Library in IBM VisualAge COBOL

This procedure describes how use ODBC API(IBM VisualAge COBOL) under an XA environment to write CICS application programs that can access data that resides in a Microsoft SQL Server database.

Prerequisite tasks and conditions: If you want to use Microsoft SQL Server, CICS for Windows requires MS SQL Workstation or Server to be installed and configured according to your Microsoft SQL documentation. When installing MS SQL, ensure that Distributed Transaction Coordinator (DTC) support is included.

To build CICS/ODBC applications, which integrate CICS and Microsoft SQL Server, you must have the MSDTC and ODBC components of the Microsoft SQL Server installed on your development systems. Also, you must have Microsoft SQL Server ODBC Library and IBM VisualAge COBOL to enable you to build ODBC applications for VisualAge COBOL.

For more information refer to the *CICS Administration Guide for Windows Systems*.

Obtaining ODBC connection handles: In an XA environment, connections that are made to the Microsoft SQL Server database by using ODBC are made through the SwitchLoad file. The SwitchLoad file makes the necessary connections to the database through the SQLConnect() ODBC function. The SQLConnect ODBC API returns an ODBC connection handle that is maintained in a CICS internal structure. The connection handle is unique for every XA Definition that is in the XAD.stanza.

CICS applications can access the list of ODBC connection handles through a CICS-supplied API. The syntax of that API is:

```
EXEC CICS ASSIGN ODBCHNDLLIST(data-area) ODBCLISTLEN(data-area)
```

Refer to the *CICS Application Programming Reference* for more information about this API.

Using a sample CICS IBM VisualAge COBOL application: The source, map, makefile, and README file for the example are shipped in this directory of your CICS development environment:

```
ProdDir/src/examples/xa
```

In the following example, ODBC APIs have been used to write the source files in IBM VisualAge COBOL. Application programmers can refer to these files to determine how to use ODBC APIs to write application programs and also to determine how to use the EXEC CICS ASSIGN ODBCHNDLLIST API to obtain ODBC connection handles.

- chzvacob_mssql.ccp(source file)
- chzvacob_mssql.cpy (copy file)
- chzvacob_mssql.mk (make file to compile the source)
- uxa1_mscob.bms (map file)

This portion of the cheese sample source file that is written in IBM VisualAge COBOL shows how to use EXEC CICS ASSIGN ODBCHNDLLIST API to obtain a list of ODBC connection handles.

```
*****************************************************************
      * CHZVACOB_MSSQL.CBL                                          *
      *****************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. "CHZVACOB_MSSQL".
       DATA DIVISION.

       WORKING-STORAGE SECTION.
      *  copy ODBC API constant definitions
           COPY "odbc3.cpy" SUPPRESS.
           COPY "uxa1.cpy" SUPPRESS.

      * ODBC Handles
       01  Henv                       POINTER           VALUE NULL.
       01  Hdbc                       POINTER           VALUE NULL.
       01  Hstmt                      POINTER           VALUE NULL.

      * Arguments used for GetDiagRec calls
       01  DiagHandleType             COMP-5  PIC 9(4).
       01  DiagHandle                 POINTER.
       01  DiagRecNumber              COMP-5  PIC 9(4).
       01  DiagRecNumber-Index        COMP-5  PIC 9(4).
                         -
                         -
                         -
                         -
                         -

      * To GET ODBC HANDLES.

       01  BUFLENGTH                        COMP-5 PIC S9(9).
       01  CONNID                           COMP-5 PIC S9(4).
       01  INDEX1                           COMP-5 PIC S9(9) VALUE 1.
       01  ODBCLEN                          COMP-5 PIC s9(9).

       01  ODBCCONLIST.
        02 ODBCLIST OCCURS 20 TIMES.
          03 ConId                         COMP-5 PIC S9(4) VALUE 0.
          03 HNDLTYPE                      COMP-5 PIC S9(4) VALUE 0.
          03 DBHANDLE                      POINTER VALUE NULL.

       LINKAGE SECTION.

      PROCEDURE DIVISION.
```

```
                MOVE LOW-VALUES TO PANEL1O.
                MOVE LOW-VALUES TO PANEL2O.
                MOVE LOW-VALUES TO PANEL3O.
                MOVE LOW-VALUES TO PANEL4O.

                EXEC CICS SEND
                  MAP    ('PANEL1')
                  MAPSET ('UXA1')
                  FREEKB
                  ERASE
                END-EXEC.

                IF EIBRESP IN DFHEIBLK NOT = DFHRESP(NORMAL)
                  EXEC CICS ABEND ABCODE ('X001') END-EXEC
                END-IF.

                IF EIBRESP IN DFHEIBLK NOT = DFHRESP(NORMAL)
                  EXEC CICS ABEND ABCODE ('X000') END-EXEC
                END-IF.

              *
              * Receive the response
              *
                    EXEC CICS RECEIVE
                      MAP    ('PANEL1')
                      MAPSET ('UXA1')
                    END-EXEC.

                    IF EIBRESP IN DFHEIBLK NOT = DFHRESP(NORMAL)
                      EXEC CICS ABEND ABCODE ('X002') END-EXEC
                    END-IF.

                    MOVE SPACES TO CNAME.
                    MOVE NEWNAMEI OF PANEL1I TO CNAME.

              *
              *    SET ODBCHANLDE FROM THE LIST FOR CONNECTIONID 1

                    EXEC CICS ASSIGN ODBCHNDLLIST(ODBCCONLIST)
                              ODBCLISTLEN(ODBCLEN) END-EXEC.
                    MOVE 1 TO CONNID.
                    PERFORM SETDBHANDLEFROMLIST.
                                           -
                                           -
                                           -
                    COPY  chzvacob_mssql.cpy.

            The below portion of the code can be seen in the file chzvacob_mssql.cpy .

            ** SETDBHANDLEFROMLIST SECTION ***********************************

               SETDBHANDLEFROMLIST SECTION.
                   PERFORM VARYING INDEX1 FROM 1 BY 1
                       UNTIL ConId OF ODBCLIST(INDEX1) = 0
                       IF ConId OF ODBCLIST(INDEX1) = CONNID
                       SET Hdbc TO DBHANDLE OF ODBCLIST(INDEX1)
                       END-IF
                   END-PERFORM.
```

In this portion of the example code, the EXEC CICS ASSIGN ODBCHNDLLIST API fills the buffer that is passed by the program. It fills the buffer with an array of structures that contains the List of ODBCHandles, Type of the each ODBCHandle, and the ConnectID.

The ConnectID member identifies the database to which the connection handle refers. The user sets this ConnectID in the XAOpen string of XAD.stanza for the XAD product definition. Users use this ConnectID in the application, to obtain the ODBC connection handle for each database that they want to access.

For more information about the ODBC_ConnArray structure, refer to the header file `cics_odbc.h`, which you can see in the directory Drive:\opt\cics\include.

Writing Programs using E-SQL API/ODBC Driver Library in Micro Focus NetExpress COBOL

This procedure describes how use E-SQL API/ODBC Library (Micro Focus NetExpress COBOL) under an XA environment to write CICS application programs that can access data that resides in a Microsoft SQL Server database.

Prerequisite tasks and conditions: If you want to use Microsoft SQL Server, CICS for Windows requires MS SQL Workstation or Server to be installed and configured according to your Microsoft SQL documentation. When installing MS SQL, ensure that Distributed Transaction Coordinator (DTC) support is included.

To build CICS/ODBC applications, which integrate CICS and Microsoft SQL Server, you must have the MSDTC and ODBC components of the Microsoft SQL Server installed on your development systems. Also, you must have Microsoft SQL Server ODBC Library and Micro Focus NetExpress COBOL to enable you to build ODBC applications for NetExpress COBOL.

For more information refer to the *CICS Administration Guide for Windows Systems*.

Note: To write CICS ODBC applications in Micro Focus NetExpress COBOL, you must use an Embedded SQL API. Micro FocusNetExpress provides an ODBC converter that converts Embedded SQL APIs to ODBC during compilation. Applications can, therefore, access the database through the ODBC driver.

Obtaining ODBC connection handles: Because CICS applications are written with embedded SQL API in Micro Focus NetExpress COBOL, you must use the EXEC SQL SET CONNECTION API to set the connection handles for the application.

Using a sample CICS Micro Focus NetEXpress COBOL application: The source, map, makefile, and README file for the example are shipped in this directory of your CICS development environment:

```
ProdDir/src/examples/xa
```

In the following example, ODBC APIs have been used to write the source files in Micro Focus NetExpress COBOL. Application programmers can refer to these files to determine how to use ODBC APIs to write application programs.

- chzmfcob_mssql.cpp (source file)
- uxa1_mscob.bms (map file)
- chzmfcob_mssql.mk (makefile)

This portion of the cheese sample source file that is written in Micro Focus NetExpress COBOL shows how to use an Embedded SQL API to set the connection.

```
IDENTIFICATION DIVISION.
       PROGRAM-ID.           UXA1.
       AUTHOR.               A SAMPLE.
```

```
 DATE-WRITTEN.          18/10/96.
 DATE-COMPILED.
*****************************************************************
 ENVIRONMENT DIVISION.
*****************************************************************
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY UXA1.
                -
                -
                -
 PROCEDURE DIVISION.
 RESIDENT SECTION 1.
*****************************************************************
 MAIN.
*****************************************************************
*
*   Initialise the maps
*

      MOVE LOW-VALUES TO PANEL1O.
      MOVE LOW-VALUES TO PANEL2O.
      MOVE LOW-VALUES TO PANEL3O.
      MOVE LOW-VALUES TO PANEL4O.

*   Set error handling

      EXEC SQL WHENEVER SQLERROR GOTO :ERR-EXIT END-EXEC.

*----------
* Change the name server1 to a connection name you defined
* in your CICS MS SQL Server XAD stanza XA_OPEN string.
*----------
      EXEC SQL SET CONNECTION server1 END-EXEC.
```

File processing using EXTFH with non-CICS applications

To update user files (files defined in the File Definitions (FD) of a CICS application) from non-CICS application, you possibly need to compile or link your programs with either extra flags or with an extra module. In addition, you possibly need to change identifiers to library links within COBOL system files. The changes that are required differ depending on the language and the file system manager.

Table 33 details the changes required to process files from non-CICS applications.

Table 33. Support required for updating files from non-CICS applications

Compiler	Files on SFS	Files on DB2	Files on Oracle
C and C++	No additional support is required. Refer to the SFS application programming information.	No additional support is required. Refer to the DB2 application programming information.	No additional support is required.
IBM PL/I	Build your application with the appropriate definitions.	Support not available.	Support not available.
IBM COBOL	Use the SFS naming convention as defined in *SMARTdata UTILITIES for AIX* and set the appropriate environment variables.	Support not available.	Support not available.

Table 33. Support required for updating files from non-CICS applications (continued)

Compiler	Files on SFS	Files on DB2	Files on Oracle
Micro Focus Server Express COBOL on UNIX	Indicators to two COBOL libraries must be modified in two COBOL system files. The application or the COBOL runtime unit (rts) must be linked with the Encina EXTFH.	The application must be linked with the DB2 EXTFH library that is provided with CICS.	The application must be linked with the Oracle EXTFH library that is provided with CICS.
Micro Focus Net Express on Windows	Indicators to two COBOL libraries must be modified in two COBOL system files. The application or the COBOL runtime unit (rts) must be linked with the Encina EXTFH.	The application must be linked with the DB2 EXTFH library that is provided with CICS.	The application must be linked with the Oracle EXTFH library that is provided with CICS.

Table 33 on page 157 shows the support requirements that are needed by CICS to enable processing of SFS, DB2, or Oracle files when each of the different language compilers is used. When SFS, DB2, or Oracle files are processed with either the C or the C++ compiler, no extra capability is required. These languages already contain what is needed to access SFS, DB2, or Oracle files. With PL/I compilers, no extra capability is required for SFS access, but DB2 and Oracle file access is not supported. With IBM COBOL compilers, DB2 and Oracle file access is not supported. However, SFS file access is possible when specific naming conventions are employed, and the following environment variables are set:

- **ENCINA_EXTFH_VOL** — specifies the name of the SFS logical volume that was specified when the server was started, <*shortname*>.
- **ENCINA_CDS_ROOT** — set to **/.:/cics**

Note, however, that when the Micro Focus Server Express COBOL compiler (on Open Systems) or Micro Focus Net Express compiler (on Windows systems) is used, extra capability is needed for access to SFS, DB2, or Oracle files.

This extra capability that is needed by Micro Focus Server Express COBOL or Net Express is provided through the *External File Handler* (EXTFH) function. The EXTFH is a package that allows COBOL applications transparently to use SFS, DB2, or Oracle files for record storage. When the UNIX and the Windows environment has been prepared to use the EXTFH function, the routines to access data are the same. The COBOL programmer sees no apparent difference between a standard COBOL I/O and an I/O to an SFS file, DB2 file, or Oracle file that is using EXTFH. The EXTFH is supported in the Micro Focus Server Express COBOL and Net Express software and also by EXTFH code on the DB2, SFS, and Oracle file system managers.

The program is built into an executable by using **cob**, and is run independently.

The EXTFH interface automatically creates output files if they do not exist. If you open an existing file for output, its contents are erased unless you open it in extended mode.

In order to use EXTFH, you must use COBOL and the X/Open TX routines. You cannot use EXTFH and Transactional-C together.

The following sections cover these topics:

- Using DB2 EXTFH with Micro Focus Server Express COBOL and Net Express
- Using SFS EXTFH with Micro Focus Server Express COBOL and Net Express

Using DB2 EXTFH with Micro Focus Server Express COBOL and Net Express

This section discusses the following EXTFH topics:

- Micro Focus Server Express COBOL runtime using DB2 EXTFH on UNIX platforms
- Building a standalone Micro Focus Server Express COBOL program using DB2 EXTFH on UNIX
- Micro Focus Net Express COBOL runtime using DB2 EXTFH on Windows
- Customization of the DB2 EXTFH on UNIX and on Windows
- File and record locking behavior when DB2 EXTFH is used with Micro Focus Server Express COBOL on UNIX and Net Express on Windows

Using Micro Focus Server Express COBOL runtime with DB2 EXTFH on UNIX

The steps required to enable DB2 EXTFH functionality with the Micro Focus Server Express COBOL runtime on the UNIX platform are:

1. Verify that the environment is set up as is shown in the following AIX example:
   - PATH is set to the following:
     ```
     $PATH:/usr/lpp/cics/bin:/usr/lpp/encina/bin:/opt/dcelocal/bin:$DB2DIR/bin
     ```
   - LIBPATH is set to /usr/lpp/cics/lib:/usr/lpp/encina/lib:/opt/dcelocal/lib:$DB2DIR/lib
   - NLSPATH is set to NLSPATH:/usr/lpp/cics/msg/%L/%N/:/usr/lpp/cics/msg/C/%N
   - DB2DBDFT is set to the name of any DB2 database
   - DB2DIR=<*DB2productDirectory*>
   - DB2INSTANCE=<*instanceName*>
2. Export DCELIBS="-ldce —lc_r —ldcepthreads —lpthreads"

Building a standalone MF COBOL program using DB2 EXTFH on UNIX

To build a Micro Focus Server Express COBOL application, follow the instructions for using DB2 EXTFH with an MF COBOL runtime. In addition, you must specify extra parameters on the command-line to the **cob** command, as shown in the following example for clustered files:

```
cob -mixfile=cics_xfh -L/usr/lpp/cics/lib  -lxfhdb2sa  -x
ExtFHClust.cbl -lcicssa
rm *.o *.int
```

This will build a program named *ExtFHClust*.

Using Micro Focus Net Express COBOL runtime with DB2 EXTFH on Windows

The steps required to enable DB2 EXTFH functionality on the Windows platform are:

1. **Ensure COBOL environment file settings.**
   a. Ensure *prodDir*\bin is in the %PATH% environment setting.
   b. Ensure *prodDir*\lib is in the %LIB% environment setting.

2. **Compile the COBOL application with correct compile-time flag.**

   The COBOL application must be compiled with the COBOL compile-time flag **CALLFH"cics_xfh"**.

   For example, build your program using:

   ```
   cobol Testprog1.cbl CALLFH"cics_xfh";
   cbllink -rE Testprog1.obj
   ```

   This redirects all COBOL file operations to the DB2 EXTFH.

3. **Execute the runtime unit.**

   When the COBOL application is run, the application must be able to find and load the library **cics_xfh.dll** which is found in the directory *prodDir***bin**.

Customization of the DB2 EXTFH on UNIX and on Windows

The DB2 EXTFH is customized with the following environment variables:

Table 34. Environment variables used with the DB2 EXTFH on Windows

Environment variable	Description
$CICS_XFH_LOGFILE or %CICS_XFH_LOGFILE%	Defines the path name to an alternative log for the external file handler. Normally this environment variable is not set and the external file handler appends to a file called `xfh.LogFile` in a temporary directory. If this file does not exist, it is created. Multiple concurrent external file handler applications can use the same file. If you do not want to keep a log file, set this environment variable to **NONE**. When this is done, errors are reported to stderr.
$CICS_XFH_DBNAME or %CICS_XFH_DBNAME%	Defines the name of the DB2 database that is used by the external file handler. If the variable is not set, the default DB2 database is used.
$CICS_XFH_USERNAME, $CICS_XFH_USERPASS or %CICS_XFH_USERNAME%, % CICS_XFH_USERPASS%	Tools that are supplied with the database are usually used to regulate who can access the data. However, the external file handler accesses the database as the user who is running the COBOL application. In addition, the access ID can be changed by using these two environment variables to define the user name and password that are to be used to connect to the database.
$CICS_XFH_TRANMODE or %CICS_XFH_TRANMODE%	Defines transactional 'T' or nontransactional 'N' access to the database. The default operation is nontransactional access.

File and record locking behavior when DB2 EXTFH is used with Micro Focus Server Express COBOL on UNIX and Net Express on Windows

Micro Focus Server Express COBOL, Net Express, and the DB2 EXTFH support file and record locking in the following ways:

- Files opened for output or append can take a file lock.
- To take a file lock on files opened for read only, either of the following settings must be specified:
  - the 'WITH LOCK' phrase must be specified on the OPEN statement, or
  - the 'LOCK MODE IS EXCLUSIVE' phrase must be specified on the SELECT statement.

Without these specified settings, files opened for read only can take no locks, and the 'WITH LOCK' clause has no effect on a read statement (standard Micro Focus Server Express COBOL or Net Express operation).

- The DB2 EXTFH only supports automatic record locking. Files opened with manual record locking will default to automatic record locking and the locking schematics used will be that of 'LOCK MODE IS AUTOMATIC WITH LOCK ON MULTIPLE RECORD'.

Non-transactional access mode

With non-transactional access, changes are immediately committed to the database. The DB2 EXTFH performs a COMMIT immediately after any database update operation.

With files opened for input and output in non-transactional access mode, a lock is taken on any record read and then released on the next file operation on that file or when the application ends. If the 'WITH LOCK' phrase is specified on the open statement or 'LOCK MODE IS EXCLUSIVE' is specified on the SELECT statement, then a file lock is taken.

When a ROLLBACK is performed, the file lock is temporarily released before being retaken. In this situation, the application receives an error from the ROLLBACK command and a message is logged.

Transactional access mode

When using transactional access, it is up to the application to:

- Make the changes permanent (COMMIT)
- Disregard the changes (ROLLBACK)

Because some COBOL file operations do not map to single SQL commands, an automatic rollback does not necessarily occur if errors are encountered. For example, during a file create, if the database table is created successfully but the indices are not, the DB2 EXTFH does not perform a rollback of the table creation. This results in a error when the application attempts to open the file. Therefore, you should program the application to handle the error and explicitly perform the rollback.

With files opened for input and output in transactional access mode, a lock is taken on any record read and then released on the next file operation on that file except when that record is updated. When a file is updated, the lock is retained until a COMMIT or ROLLBACK is performed or the application ends. Therefore, you should program the application to handle the error and explicitly perform the rollback.

Using Oracle EXTFH with Micro Focus Server Express COBOL

This section discusses the following EXTFH topics:

- Micro Focus Server Express COBOL runtime using Oracle EXTFH on UNIX platforms
- Building a standalone Micro Focus Server Express COBOL program using Oracle EXTFH on UNIX
- Customization of the Oracle EXTFH on UNIX and on Windows
- File and record locking behavior when Oracle EXTFH is used with Micro Focus Server Express COBOL on UNIX

Using Micro Focus Server Express COBOL runtime with Oracle EXTFH on UNIX

To enable Oracle EXTFH functions with the Micro Focus Server Express COBOL runtime on the UNIX platform:

1. Verify that the environment is set up as is shown in the following AIX example:
   - PATH is set to the following:

     ```
     $PATH:/usr/lpp/cics/bin:/usr/lpp/encina/bin:/opt/dcelocal/bin
       :$ORACLE_HOME/bin
     ```

   - LIBPATH is set to:

     ```
     /usr/lpp/cics/lib:/usr/lpp/encina/lib:/opt/dcelocal/lib:$ORACLE_HOME/lib32
     ```

     (assuming a 64-bit installation for Oracle)
   - NLSPATH is set to:

     ```
     NLSPATH:/usr/lpp/cics/msg/%L/%N/:/usr/lpp/cics/msg/C/%N
     ```

   - ORACLE_SID is set
   - CICS_XFH_USERPASS is set
   - CICS_XFH_USERNAME is set
2. Export DCELIBS=″-ldce —lc_r —ldcepthreads —lpthreads″

Building a standalone MF COBOL program using Oracle EXTFH on UNIX

To build a Micro Focus Server Express COBOL application, follow the instructions for using Oracle EXTFH with an MF COBOL runtime. In addition, you must specify extra parameters on the command-line to the **cob** command, as shown in the following example for clustered files:

```
cob -x ExtFHClust.cbl -mixfile=cics_xfh ${ORA_PRECOMP}/cobsqlintf.o \
           -L/usr/lpp/cics/lib -L ${ORA_LIBP}  \
           -L ${COBDIR}/lib -L ${LIBDIR} \
           -l${EXTFH_LIBNAME} -lcicssa -lclntsh
  rm *.o *.int
```

This l builds a program that is called *ExtFHClust*.

Notes:

1. The variable $COBDIR resolves to the COBOL installation directory. For Solaris and HP-UX, $COBDIR resolves to /opt/cobol/. For AIX, $COBDIR resolves to /usr/lib/cobol/.
2. ORA_PRECOMP resolves to ${ORACLE_HOME}/precomp/lib for Oracle 8.1.7 or Oracle 9i 32-bit installation, or to ${ORACLE_HOME}/precomp/lib32 for Oracle 9i 64-bit installation.
3. ORA_LIBP = ${ORACLE_HOME}/lib for Oracle 8.1.7 or Oracle 9i 32-bit installation, or to ${ORACLE_HOME}/lib32 for Oracle 9i 64-bit installation.
4. EXTFH_LIBNAME=xfhorasa (provided by CICS).
5. CICS provides a script $CICSPATH/bin/cicsmkextfh to aid users to build the standalone MF COBOL executable that is to be used with Oracle Extfh.

Using Micro Focus Net Express COBOL runtime with Oracle EXTFH on Windows

To enable Oracle EXTFH functionality on the Windows platform:

1. **Ensure that the COBOL environment file settings are correct.**
   a. Ensure that *prodDir*\bin is in the %PATH% environment setting.
   b. Ensure that *prodDir*\lib is in the %LIB% environment setting.

2. **Compile the COBOL application with the correct compile-time flag.**

   The COBOL application must be compiled with the COBOL compile-time flag **CALLFH**″**cics_xfh**″.

   For example, build your program by using:

   ```
   cobol Testprog1.cbl CALLFH"cics_xfh";
   cbllink -rE Testprog1.obj
   ```

   All COBOL file operations are then redirected to the Oracle EXTFH.

3. **Execute the runtime unit.**

   When the COBOL application is run, the application must be able to find and load the library **cics_xfh.dll**, which is in the directory *prodDir***bin**.

Customization of the Oracle EXTFH on UNIX and on Windows

The Oracle EXTFH is customized with the following environment variables:

Table 35. Environment variables used with the Oracle EXTFH on Windows

Environment variable	Description
$CICS_XFH_LOGFILE or %CICS_XFH_LOGFILE%	Defines the path name to an alternative log for the external file handler. Normally this environment variable is not set and the external file handler appends to a file called `xfh.LogFile` in a temporary directory. If this file does not exist, it is created. Multiple concurrent external file handler applications can use the same file. If you do not want to keep a log file, set this environment variable to **NONE**. When this is done, errors are reported to stderr.
$CICS_XFH_DBNAME or %CICS_XFH_DBNAME%	Defines the name of the Oracle database that is used by the external file handler. If the variable is not set, the default Oracle database is used.
$CICS_XFH_USERNAME, $CICS_XFH_USERPASS or %CICS_XFH_USERNAME%, % CICS_XFH_USERPASS%	Tools that are supplied with the database are usually used to regulate who can access the data. However, the external file handler accesses the database as the user who is running the COBOL application. In addition, the access ID can be changed by using these two environment variables to define the user name and password that is to be used to connect to the database.
$CICS_XFH_TRANMODE or %CICS_XFH_TRANMODE%	Defines transactional 'T' or nontransactional 'N' access to the database. The default operation is nontransactional access.

File and record locking behavior when Oracle EXTFH is used with Micro Focus Server Express COBOL on UNIX and Net Express on Windows

Micro Focus Server Express COBOL, Net Express, and the Oracle EXTFH support file and record locking in the following ways:

- Files that are opened for output or append can take a file lock.
- To take a file lock on files that are opened for read only, you must specify either of the following settings:
  - Specify the 'WITH LOCK' phrase on the OPEN statement, or
  - Specify the 'LOCK MODE IS EXCLUSIVE' phrase on the SELECT statement.

Without these specified settings, files that are opened for read only can take no locks, and the 'WITH LOCK' clause has no effect on a read statement (standard Micro Focus Server Express COBOL or Net Express operation).

- The Oracle EXTFH supports only automatic record locking. Files that are opened with manual record locking default to automatic record locking. The locking schematics that are used are 'LOCK MODE IS AUTOMATIC WITH LOCK ON MULTIPLE RECORD'.

Non-transactional access mode

With non-transactional access, changes are immediately committed to the database. The Oracle EXTFH performs a COMMIT immediately after any database update operation.

With files are opened for input and output in nontransactional access mode, a lock is taken on any record read, then released on the next file operation on that file or when the application ends. If the 'WITH LOCK' phrase is specified on the open statement, or 'LOCK MODE IS EXCLUSIVE' is specified on the SELECT statement, a file lock is taken.

When a ROLLBACK is performed, the file lock is temporarily released before being retaken. In this condition, the application receives an error from the ROLLBACK command and a message is logged.

Transactional access mode

When using transactional access, the application decides whether:

- To make the changes permanent (COMMIT)
- To disregard the changes (ROLLBACK)

Because some COBOL file operations do not map to single SQL commands, an automatic rollback does not necessarily occur if errors are found. For example, during a file create, if the database table is created successfully but the indices are not, the Oracle EXTFH does not perform a rollback of the table creation. This results in an error when the application attempts to open the file. Therefore, you should program the application to handle the error and explicitly perform the rollback.

With files are opened for input and output in transactional access mode, a lock is taken on any record read, then released on the next file operation on that file, except when that record is updated. When a file is updated, the lock is retained until a COMMIT or ROLLBACK is performed or the application ends. Therefore, program the application to handle the error and explicitly perform the rollback.

Using SFS EXTFH with a Micro Focus Server Express COBOL or Net Express runtime

This section discusses the use of the SFS EXTFH with Micro Focus Server Express COBOL on UNIX and Micro Focus Net Express on Windows systems. The topics discussed are:

- Using SFS EXTFH with Micro Focus Server Express COBOL runtime on UNIX
- Using a standalone SFS EXTFH with Micro Focus Server Express COBOL on UNIX
- Using SFS EXTFH with Micro Focus Net Express COBOL runtime on Windows systems
- Customizing the SFS EXTFH

Using SFS EXTFH with Micro Focus Server Express COBOL runtime on UNIX

To use your Micro Focus Server Express COBOL programs to access SFS files:

1. Verify that the environment is set up as is shown in the following AIX example:
   - PATH is set to $PATH:/usr/lpp/cics/bin:/usr/lpp/encina/bin:/opt/dcelocal/bin
   - LIBPATH is set to /usr/lpp/cics/lib:/usr/lpp/encina/lib:/opt/dcelocal/lib
   - NLSPATH is set to NLSPATH:/usr/lpp/encina/C/%N/:/opt/dcelocal/nls/msg/%L/%N
2. **Prepare the environment for EXTFH use**

   Do this by defining files, setting environment variables, and assigning the appropriate permissions.

   EXTFH uses environment variables to determine which SFS server and volume to use:
   - **ENCINA_SFS_SERVER** — specifies which server to use. This must be set to the fully qualified name of the SFS, for example: **/.:/cics/sfs/***serverName*
   - **ENCINA_EXTFH_SFS** — specifies which server to use. This must be set to the fully qualified name of the SFS, for example: **/.:/cics/sfs/***$ENCINA_SFS_SERVER/serverName*.
   - **ENCINA_EXTFH_VOL** — specifies the name of the SFS logical volume specified when the server was started, <*shortname*>.
   - **ENCINA_CDS_ROOT** — set to **/.:/cics**

   The server is used for all SFS file operations. The volume is used when a new file has to be created. The environment variables are checked when the file is opened. These variables can be set or changed using the following calls:
   - **TR_SET_SFS_SERVER** *serverName nameSize*. The call takes two options: a string variable containing the fully qualified server name and a numeric variable giving the length of the string in the serverName option.
   - **TR_SET_SFS_VOLUME** *volumeName nameSize*. The call takes two options: a string variable containing the name of the volume and a numeric variable giving the length of the string in the volume_name option.
3. Export the SFS EXTFH librarys using the following command:

   ```
   export ENCLIBS="-lEncSfsExtfhWrap -lEncSfsExtfh -lEncSfs -lEncina"
   ```

   Refer to the Encina README for more information.
4. Verify that the following DCE libraries are correct for your operating system. (These libraries are also used for implementing standalone SFS EXTFH with MF COBOL.)

   ```
   export DCELIBS="-ldce -lc_r -ldcepthreads -lpthreads"
   ```

   Note: The Encina **build_rts32** script used to create the MF COBOL runtime links the required libraries.
5. If DCE authentication is used for the SFS, log in to DCE.
6. **Prepare an EXTFH COBOL runtime**

   Follow the instructions for your version of Encina:

On AIX

a. Ensure that the COBDIR environment variable is set to the COBOL directory (for example `/usr/lib/cobol`).

b. Use the following command:

   `/usr/lpp/encina/etc/build_rts32 -o rtssfs -d .`

   The script `build_rts32` is provided with Encina. The **-o** option is used for the name of the runtime being produced and **-d** indicates the directory that you wish to place the runtime module in. If you want to indicate the current directory, use the character ".", or you may fully qualify the directory.

On HP-UX

a. Ensure that the COBDIR environment variable is set to the COBOL directory (for example, `/opt/cobol/cobdir`).

b. Ensure that the LPATH environment variable includes the `/opt/encina/lib` directory. For example,

   `export LPATH=/opt/encina/lib`

c. Use the following command:

   `/opt/encina/etc/build_rts32 -o rtssfs -d .`

   The script `build_rts32` is provided with Encina. The **-o** option is used for the name of the runtime being produced and **-d** indicates the directory that you wish to place the runtime module in. If you want to indicate the current directory, use the character ".", or you may fully qualify the directory.

On Solaris

a. Ensure that the COBDIR environment variable is set to the COBOL directory (for example `/opt/lib/cobol`).

b. Use the following command:

   `/opt/encina/etc/build_rts32 -o rtssfs -d .``cob -u ExtFHClust.cbl`

   The script `build_rts32` is provided with Encina. The **-o** option is used for the name of the runtime being produced and **-d** indicates the directory that you wish to place the runtime module in. If you want to indicate the current directory, use the character ".", or you may fully qualify the directory.

At this point you have a COBOL runtime (named 'rtssfs' located in the directory you specified with the -d option) that includes the Encina-provided routines to access SFS.

7. **Compile your program**

   You may use the following command to compile your COBOL programs. Depending on your program, you may need to indicate compiler directives; refer to the COBOL documentation for details.

```
$ cob -uv testsfs.cbl
cob -u ExtFHClust.cbl
cob -u ExtFHRel.cbl
cob -u ExtfhSeq.cbl
rm *.int *.o
```

where *testsfs.cbl* is the name of the COBOL source file, **v** specifies that messages be sent to the screen as this file is processed, and **u** requests an unlinked version of the output (**.gnt**) for use with the runtime. The filenames *ExtFHClust.cbl, ExtFHRel.cbl,* and *ExtFHSeq.cbl* refer to clustered, relative or sequential files compiled for the application.

8. **Execute your program**

   Use the following command to execute your COBOL programs.

```
dce_login principal password
export ENCINA_SFS_SERVER=/.:/cics/sfs/serverName
export ENCINA_CDS_ROOT=/.:/cics
export ENCINA_EXTFH_VOL=<SFSlogVol shortname>
export ENCINA_EXTFH_SFS=serverName
./rtssfs testsfs
```

   where:
   - *principal* is the DCE principal.
   - *password* is the password for the DCE principal.
   - *SFSlogVol shortName* is the name of the SFS logical volume.
   - *serverName* is the name of the SFS, for example `/.:/cics/sfs/HostA`.

   A DCE login is required to access an SFS file (this is discussed in detail later in this document).

   `testsfs` is the COBOL executable module that was prepared using this procedure. `./rtssfs testsfs` causes the runtime to execute the program.

Using a standalone SFS EXTFH with Micro Focus Server Express COBOL on UNIX

To use a standalone SFS EXTFH with your Micro Focus Server Express COBOL programs to access SFS files:

1. **Prepare the environment for EXTFH use**

   Do this by defining files, setting environment variables, and assigning the appropriate permissions. Set the environment variables as is shown in the instructions for using SFS EXTFH with Micro Focus Server Express COBOL runtime.

2. **Compile your program**

   You may use the following command to compile your COBOL programs. Depending on your program, you may need to indicate compiler directives; refer to the COBOL documentation for details.

   For clustered files, use the following example:

```
cob -x ExtFHClust.cbl
-L $PRODDIR/encina/lib
-L /usr/lib/$ENCLIBS/$DCELIBS
-m ixfile=cobol_Extfh -m ixfilev=cobol_Extfh
-m rlfile=cobol_Extfh -m rlfilev=cobol_Extfh
-m sqfile=cobol_Extfh -m sqfilev=cobol_Extfh
rm *.int *.o
```

   For relative files, use the following example:

```
cob -x ExtFHRel.cbl
-L $PRODDIR/encina/lib
-L /usr/lib/$ENCLIBS/$DCELIBS
```

```
-m ixfile=cobol_Extfh -m ixfilev=cobol_Extfh
-m rlfile=cobol_Extfh -m rlfilev=cobol_Extfh
-m sqfile=cobol_Extfh -m sqfilev=cobol_Extfh
rm *.int *.o
```

For sequential files, use the following example:

```
cob -x ExtFHSeq.cbl
-L $PRODDIR/encina/lib
-L /usr/lib/$ENCLIBS/$DCELIBS
-m ixfile=cobol_Extfh -m ixfilev=cobol_Extfh
-m rlfile=cobol_Extfh -m rlfilev=cobol_Extfh
-m sqfile=cobol_Extfh -m sqfilev=cobol_Extfh
rm *.int *.o
```

3. **Execute your program**

   Use the following command to execute your COBOL programs.

   ```
   dce_login principal password
   export ENCINA_EXTFH_VOL=SFSlogVol
   export ENCINA_EXTFH_SFS=serverName
   ./rtssfs testsfs
   ```

   where:
   - *principal* is the DCE principal.
   - *password* is the password for the DCE principal.
   - *SFSlogVol* is the name of the SFS logical volume.
   - *serverName* is the name of the SFS, for example `/.:/cics/sfs/HostA`.

   A DCE login is required to access an SFS file (this is discussed in detail later in this document).

   `testsfs` is the COBOL executable module that was prepared using this procedure. `./rtssfs testsfs` causes the runtime to execute the program.

Using SFS EXTFH with Micro Focus Net Express COBOL runtime on Windows

To use your Micro Focus Net Express COBOL programs to access SFS files:

1. Verify that the environment is set up to enable command access, library access, and locale access to CICS, Encina, and DCE commands by taking the following steps:
   - Ensure that all applications that use EXTFH are linked to the **libEncExtfh.lib** library by using the /CALLFH compiler option.
   - Ensure that the LIB environment variable includes the name of the directory that contains the **libEncExtfh.lib** library (by default, **C:\opt\encina\lib**).
   - Ensure that the PATH environment variable includes the Encina binary directory (by default, **C:\opt\encina\bin**).
2. **Prepare the environment for EXTFH use**

   Do this by defining files, setting environment variables, and assigning the appropriate permissions.

   EXTFH uses environment variables to determine which SFS server and volume to use:
   - **ENCINA_SFS_SERVER** — specifies which server to use. This must be set to the fully qualified name of the SFS (for example, */.:/cics/sfs/sfs1*.
   - **ENCINA_EXTFH_SFS** — specifies which server to use. This must be set to the fully qualified name of the SFS (for example, */.:/cics/sfs/sfs1*.
   - **ENCINA_EXTFH_VOL** — specifies the name of the SFS logical volume specified when the server was started, *<shortname>* (for example, *sfs_Ssfs1*).
   - **ENCINA_CDS_ROOT** — set to the fully qualified name for the DCE cell (for example, */.:/cics/sfs/sfs1*.

The server is used for all SFS file operations. The volume is used when a new file has to be created. The environment variables are checked when the file is opened. These variables can be set or changed using the following calls:

- **TR_SET_SFS_SERVER** *serverName nameSize*. The call takes two options: a string variable containing the fully qualified server name and a numeric variable giving the length of the string in the serverName option.
- **TR_SET_SFS_VOLUME** *volumeName nameSize*. The call takes two options: a string variable containing the name of the volume and a numeric variable giving the length of the string in the volume_name option.

3. Export the SFS EXTFH librarys using the following command:

```
Set ENCLIBS="-lEncSfsExtfhWrap -lEncSfsExtfh -lEncSfs -lEncina"
```

   Refer to the Encina README for more information.

4. Verify that the following DCE libraries are correct for your operating system. (These libraries are also used for implementing standalone SFS EXTFH with Net Express COBOL.)

```
Set  DCELIBS="-ldce -lc_r -ldcepthreads -lpthreads"
```

   Note: The Encina **build_rts32** script used to create the COBOL runtime links the required libraries.

5. If DCE authentication is used for the SFS, log in to DCE.
6. **Prepare an EXTFH COBOL runtime**

   Follow the instructions for your version of Encina.

Net Express On Windows

a. Ensure that the COBDIR environment variable is set to the COBOL directory (for example `C:\NetExpress\Base\BIN`).
b. Ensure the %PATH% environment setting includes the `\opt\encina\bin` directory in it.
c. Ensure the %LIB% environment setting includes the `\opt\encina\lib` directory in it.
d. Ensure the Encina EXTFH environment variables (ENCINA_EXTFH_SFS and ENCINA_EXTFH_VOL) are set.
e. Use the following command:

```
cobol Testprog1.cbl /CALLFH"libEncExtfh";
cbllink -rE Testprog1.obj
```

   At this point you have a COBOL runtime (named 'rtssfs' located in the directory you specified with the -d option) that includes the Encina-provided routines to access SFS.

7. **Compile your program**

   You may use the following command to compile your COBOL programs. Depending on your program, you may need to indicate compiler directives; refer to the COBOL documentation for details.

```
C: > cob -uv testsfs.cbl
cob -u ExtFHClust.cbl
cob -u ExtFHRel.cbl
cob -u ExtfhSeq.cbl
rm *.int *.o
```

   where *testsfs.cbl* is the name of the COBOL source file on the Windows platform, **v** specifies that messages be sent to the screen as this file is processed,

and **u** requests an unlinked version of the output (**.gnt**) for use with the runtime. The filenames *ExtFHClust.cbl*, *ExtFHRel.cbl*, and *ExtFHSeq.cbl* refer to clustered, relative or sequential files compiled for the application.

8. **Execute your program**

   Use the following command to execute your COBOL programs.

```
dce_login principal password
SET ENCINA_SFS_SERVER=/.:/cics/sfs/serverName
Set ENCINA_CDS_ROOT=/.:/cics
Set ENCINA_EXTFH_VOL=<SFSlogVol shortname>
Set ENCINA_EXTFH_SFS=serverName
./rtssfs testsfs
```

   where:
   - *principal* is the DCE principal.
   - *password* is the password for the DCE principal.
   - *SFSlogVol shortName* is the name of the SFS logical volume.
   - *serverName* is the name of the SFS, for example `/.:/cics/sfs/HostA`.

   A DCE login is required to access an SFS file (this is discussed in detail later in this document).

   `testsfs` is the COBOL executable module that was prepared using this procedure. `./rtssfs testsfs` causes the runtime to execute the program.

Customizing the SFS EXTFH

The advantage of using SFS is that you gain transactional guarantees; if you use the transaction (TX) calls with rollback enabled, changes to records are automatically undone if the transaction aborts.

COBOL supports four file types: line-sequential, record-sequential, indexed, and relative. When EXTFH is in use, three of these are mapped to SFS file types, as shown in Table 36. SFS does not support tape-oriented routines, such as those designed to handle multiple reels of a tape containing an output file.

Table 36. EXTFH File Type Mappings

COBOL File Type	SFS File Type
Line-sequential	Not supported in SFS
Record-sequential	Entry-sequenced
Indexed	Clustered
Relative	Relative

Accessing other SFS features:

The environment variables listed in Table 37 on page 171 allow you to access other SFS features. They are checked when the file is opened. These variables can be set or changed using the COBOL calls described in this same table. The defaults listed apply both when the variable has not been set and when it is set to an invalid value. These function calls return a value of 0 if they are successful and 1 if they fail. These functions only affect subsequent file opens. They do not change the behavior of files that are already open. The possible values for the options to the calls are the same as those for the environment variables.

Table 37. Environment Variables for Accessing SFS Features from EXTFH Applications

Environment Variable	Description	Possible Values	Call
ENCINA_EXTFH_ AUTO_FLUSH	Controls the setting of the operational force flag SFS uses when opening the file.	0 = do not use operational force. Any other value means use operational force. DEFAULT = 0.	TR_EXTFH_SET_ AUTO_FLUSH
ENCINA_EXTFH_ SLOT_CHOICE	Controls where inserts occur in relative files.	The string FIRST = insertions occur in the first open slot. LAST indicates that insertions occur after the last record. DEFAULT=LAST.	TR_EXTFH_SET_ SLOT_CHOICE
ENCINA_EXTFH_ DUP_DETECTION	Controls whether duplicate detection is enabled for a file.	The string NONE disables duplicate detection. ALL enables duplicate detection. DEFAULT = ALL.	TR_EXTFH_SET_ DUP_DETECT
ENCINA_EXTFH_ OP_TIMEOUT	Defines the SFS operation timeout value.	The number of seconds to wait for an SFS operation to complete. DEFAULT = 60 seconds.	TR_EXTFH_SET_ OP_TIMEOUT
ENCINA_EXTFH_ OPEN_TIMEOUT	Defines the timeout value for SFS open file calls.	The number of seconds to wait for an open file call to complete. DEFAULT = 60 seconds.	TR_EXTFH_ SETOPEN _TIMEOUT
ENCINA_EXTFH_ CACHE	Defines the size of the read and insert caches.	The number of pages to be used for the cache. DEFAULT = 0 (caching is not enabled).	TR_EXTFH_SET_ CACHE_SIZE

Chapter 6. Coding for business logic

This chapter describes how to write application programs that use the CICS business logic services.

Introduction to business logic

Transaction processing systems tend to involve a number of application programs that perform separate logical units of work (LUW) or tasks. Dividing transaction processing up in this way has a number of advantages:

- Programs are smaller and easier to maintain.
- You can write pseudoconversational transactions in which each program performs a single task and then returns control to the operating system.
- You can distribute your applications so that you keep the presentation services, data services, and the business logic separate.

For an overview of the terminology used in this chapter, see "Transaction processing terms and concepts" on page 4.

Task initiation

Tasks can be initiated in two ways:

Terminal task initiation (TTI)

This is the most common method of task initiation. When an operator enters a transaction identifier and presses ENTER, the transaction is started. TTI also covers transactions that are initiated by EXEC CICS RETURN when the TRANSID option as the transaction identifier is used. Pseudoconversational transactions that leave a transaction identifier on the screen are TTI by definition because it is the terminal action. If the IMMEDIATE option is specified along with TRANSID, the transaction starts regardless of any other transactions that are enqueued by ATI for this terminal.

See the *CICS Administration Guide* for related information.

Automatic transaction initiation (ATI)

This covers two areas of task initiation:

- *Triggered transaction initiation*: If the systems administrator specifies a nonzero trigger level for a particular transient data intrapartition destination in the Transient Data Definitions (TDD), a task is automatically initiated when the number of entries in the queue reaches the specified level. Control is passed to an application program that then processes the data in the queue.
- *Interval control transaction initiation*: The Interval Control command EXEC CICS START TRANSID specifies the transaction identifier that will be used for a new task, the time the task will be initiated, and, optionally, a terminal identification if the task is associated with a terminal.

See the *CICS Application Programming Reference* for related information.

Program execution services

Transaction processing systems tend to involve a number of application programs that perform separate logical units of work, or tasks. CICS program execution services govern the flow of control between application programs in a CICS system. You can use program execution services commands to:

- Link one of your application programs to another, either locally or remotely, anticipating subsequent return to the requesting program (EXEC CICS LINK). The COMMAREA and the INPUTMSG options of this command allow data to be passed to the requested application program. (You cannot use the INPUTMSG and INPUTMSGLEN options of a LINK command when you are using DPL).
- Return control from one of your application programs to another, or to CICS (EXEC CICS RETURN). The COMMAREA and INPUTMSG options of this command allow data to be passed to a newly-initiated transaction. (You cannot use the INPUTMSG and INPUTMSGLEN options of a RETURN command when you are using DPL).
- Transfer control from one of your application programs to another, with no return to the requesting program (EXEC CICS XCTL). The COMMAREA and the INPUTMSG options of this command allow data to be passed to the requested application program. (You cannot use the INPUTMSG and INPUTMSGLEN options of an XCTL command when you are using DPL).

Note: The name of the application referred to in a program services command must have been defined as a program to CICS.

Application program logical levels

Application programs running under CICS are executed at various logical levels. The first program to receive control within a task is at the highest logical level. When an application program is linked to another, expecting an eventual return of control, the linked-to program is considered to reside at the next lower logical level. When control is simply transferred from one application program to another, without expecting return of control, the two programs are considered to reside at the same logical level, as shown in Figure 7 on page 175:

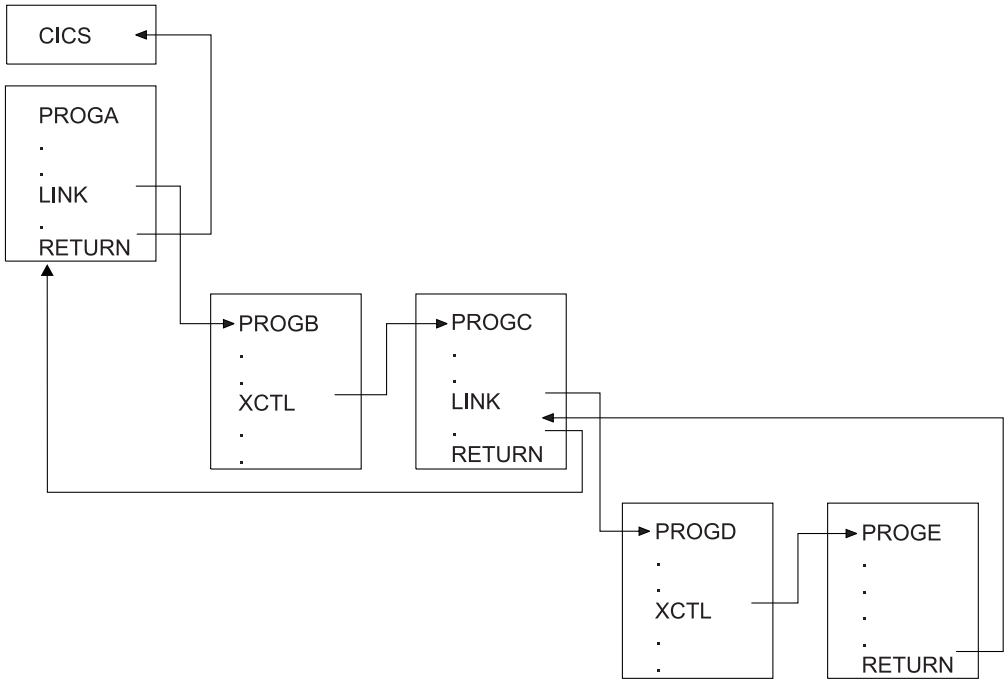


Figure 7. Application programming logical levels

Link to another program anticipating return

Use the EXEC CICS LINK command to pass control from an application program at one logical level to an application program at the next lower logical level. If the linked-to program is not already in main storage, it is loaded. When the EXEC CICS RETURN command is processed in the linked-to program, control is returned to the higher logical level program initiating the linkage at the next sequential process instruction.

The linked-to program operates independently of the program that issues the EXEC CICS LINK command with regard to handling exceptional conditions, attention identifiers, and abends. For example, the effects of HANDLE commands in the linking program are not inherited by the linked-to program, but the original HANDLE commands are restored on return to the linking program. You can use the HANDLE commands to deal with exceptional conditions, attention identifiers, and abends at the new logical level. See the *CICS Application Programming Reference* for more information about these commands.

The linked-to program can reside either locally or remotely. For details of how to access remote programs, see the *CICS Intercommunication Guide* .

Transfer control from one program to another

Use the EXEC CICS XCTL command to transfer control from one program to another at the same logical level. You cannot return to the calling program. Refer to "Application program logical levels" on page 174.

Passing data to other programs

You can pass data to another program when control is passed to that other program using a program execution services command.

Using the COMMAREA option

The COMMAREA option of the EXEC CICS LINK and EXEC CICS XCTL commands specifies the name of a data area (known as a communication area) in which data is passed to the program being invoked.

In a similar manner, the COMMAREA option of the EXEC CICS RETURN command specifies the name of a communication area in which data is passed to the transaction identified in the TRANSID option. (The TRANSID option specifies a transaction that is initiated either when the next input is received from the terminal that is associated with the task, or when the IMMEDIATE option is specified.) For more information about the length of the communication area, see the *CICS Application Programming Reference*.

The invoked program receives the data as a parameter. In COBOL, the program must contain a definition of a data area to allow access to the passed data.

In a receiving COBOL program, you must give the data area the name DFHCOMMAREA. A program passes a COMMAREA as part of an EXEC CICS LINK, EXEC CICS XCTL, or EXEC CICS RETURN command; either the working storage or the linkage section can contain the data area. A program receiving a COMMAREA should specify the data in the linkage section. This applies when the program is one of the following:

- The receiving program during an EXEC CICS LINK or EXEC CICS XCTL command where a COMMAREA is passed
- The initial program where the EXEC CICS RETURN command of a previously called task specified a COMMAREA and TRANSID

A C or C++ program issues an EXEC CICS ADDRESS COMMAREA command to obtain the address of the communication area. The program must pass the address of a pointer as a parameter to the EXEC CICS ADDRESS COMMAREA command. This pointer will then be set to point to the passed data.

The receiving data area need not be of the same length as the original communication area; if access is required only to the first part of the data, the new data area can be shorter. However, it must not be longer than the length of the communication area being passed, because if it is, your transaction may inadvertently read data outside the area that has been passed. This data is outside the area you can safely access, and may cause your transaction to have unpredictable results. It may also overwrite data outside the area, which could cause CICS to abend your transaction.

To avoid this happening, your program can access the EIBCALEN field in the EIB of the task to check that the length of the communication area passed to it is as expected. If no communication area has been passed, the value of EIBCALEN is zero; otherwise, EIBCALEN always contains the value specified in the LENGTH option of the EXEC CICS LINK, EXEC CICS XCTL, or EXEC CICS RETURN command, regardless of the size of the data area in the invoked program. You should check that the value in EIBCALEN matches the value expected by your program, and make sure that your transaction is accessing data within that area.

You might also want to consider adding an identifier to COMMAREA as an additional check on the data that is being passed. This identifier is sent with the sending transaction and checked for by the receiving transaction.

When a communication area is passed using an EXEC CICS LINK command, the invoked program is passed a pointer to the communication area itself. Any changes made to the contents of the data area in the invoked program are available to the invoking program, when control returns to it; to access any such changes, the program names the data area specified in the original COMMAREA option.

When a communication area is passed using an EXEC CICS XCTL command, a copy of that area is made unless the area to be passed has the same address and length as the area that was passed to the program issuing the command. For example, if program PROGA issues an EXEC CICS LINK command to program PROGB that, in turn, issues an EXEC CICS XCTL command to program PROGC, and if PROGB passes to PROGC the same communication area that PROGA passed to PROGB, program PROGC will be passed addressability to the communication area that belongs to A (not a copy of it) and any changes made by PROGC will be available to PROGA when control returns to it.

When a lower-level program, which is a linked-to program, issues the EXEC CICS RETURN command, control passes back to the level one logical level higher than the program returning control. If the task is associated with a terminal, the TRANSID option can be used at the lower level to specify the transaction identifier for the next transaction that is to be associated with that terminal. The transaction identifier comes into use only after the highest logical level has relinquished control to CICS by using the EXEC CICS RETURN command, and input is received from the terminal. Any input that is entered from the terminal is interpreted wholly as data. If the IMMEDIATE option is specified, the input from the terminal is not required to start the transaction. You can use TRANSID without COMMAREA when returning from any link level, but be aware that it might be overridden on a later EXEC CICS RETURN command. Also, you can only specify the COMMAREA option at the highest level, otherwise you will get an INVREQ.

In addition, the COMMAREA option can be used to pass data to the new task that is to be started.

The invoked program can access field EIBFN in the EIB to determine which type of command invoked the program. The field must be tested before CICS commands are issued. If an EXEC CICS LINK or EXEC CICS XCTL invoked the program, the appropriate code is found in the field; if EXEC CICS RETURN is used and no CICS commands are issued in the task, the field will contain zeros.

Using the INPUTMSG option

The INPUTMSG option of the LINK, XCTL, and RETURN commands provides another way of specifying the name of a data area that is to be passed to the program that is being invoked. In this case, the invoked program gets the data by processing a RECEIVE command. This option enables you to invoke (″front-end″) application programs to obtain initial terminal input. These application programs are written to be invoked directly from a terminal, and contain RECEIVE commands.

If an application program is accessed by a LINK command and the application program issues a RECEIVE command to obtain initial input from a terminal, no data exists for that application program to receive if the initial RECEIVE request has already been issued by a higher-level program. In this condition, the application program waits for input from the terminal. You can ensure that the original terminal input continues to be available to a linked program by invoking it with the INPUTMSG option.

When an application program invokes another program, and specifies INPUTMSG on the LINK, XCTL, or RETURN command, the data that is specified on the INPUTMSG continues to be available, even if the linked program itself does not issue an RECEIVE command, but instead invokes yet another application program. Figure 8 shows how INPUTMSG is used in a linked chain.

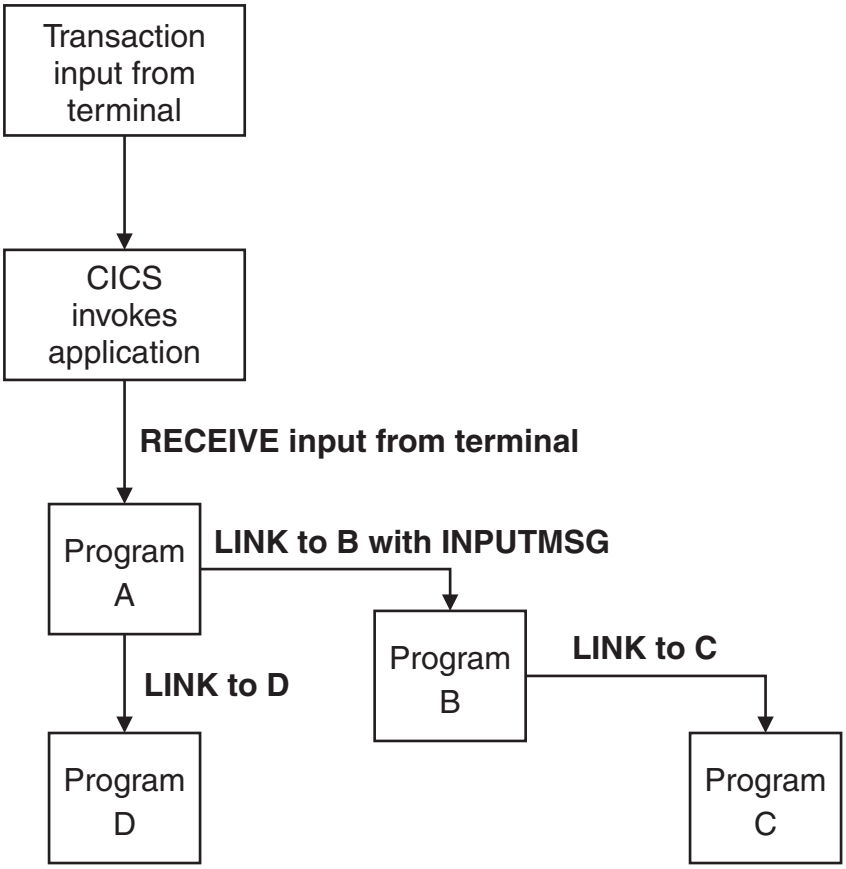


Figure 8. Use of INPUTMSG in a linked chain

In Figure 8, the ″real″ first RECEIVE command is issued by program A. Because program A is linked to program B with the INPUTMSG option, the next program that issues a RECEIVE request can also receive the terminal input. That program can be either program B or program C.

If program A wants only to pass on the unmodified terminal input that it received, it can use the same named data area for the INPUTMSG option that it used for the RECEIVE command. For example:

```
EXEC CICS RECEIVE
INTO(TERMINAL-INPUT) ...
EXEC CICS LINK PROGRAM(PROGRAMB)
INPUTMSG(TERMINAL-INPUT) ...
```

When one program in a LINK chain issues a RECEIVE command, the INPUTMSG data becomes no longer available to any later RECEIVE command. For example, in Figure 8, if program B issues a RECEIVE request before linking to program C, the INPUTMSG data area is not available for program C.

This method of sending data from one program to another can be used for any kind of data. The data does not have to originate from a user terminal. In Figure 8 on page 178, program A could move any data into the named data area, and invoke program B with INPUTMSG referencing the data.

The terminal-data that is passed on INPUTMSG also becomes no longer available when control is eventually returned to the program that issued the link with INPUTMSG. In Figure 8 on page 178, if program C returns to program B, and program B returns to program A, and neither program B nor program C issues a RECEIVE command, program A assumes that the data has been received. If program A then invokes another program (for example, program D), the original INPUTMSG data is no longer available to program D, unless the INPUTMSG option is specified.

The INPUTMSG data becomes no longer available when a SEND or CONVERSE command is issued.

Using the INPUTMSG option on the RETURN command

To pass data to the next transaction specified, you can specify INPUTMSG on a RETURN command with the TRANSID option. To do this, you must issue RETURN at the highest logical level to return control to CICS, and you must also specify the IMMEDIATE option. If you specify INPUTMSG with TRANSID, and do not also specify IMMEDIATE, the next real input from the terminal overrides the INPUTMSG data, which is therefore lost. See the *CICS Application Programming Reference* manual for programming information about the RETURN command.

If you specify INPUTMSG with TRANSID some time after a SEND command, the SEND message is immediately flushed out to the terminal. INPUTMSG on a RETURN command without the TRANSID option is intended for use with a dynamic transaction routing program.

Other ways of passing data

Data can also be passed between application programs and transactions in other ways. For example, the data can be stored in a CICS storage area outside the local environment of the application program, such as the transaction work area (TWA). Another way is to store the data in temporary storage; see the *CICS Administration Guide* for details.

Passing integer data between programs

If you want to pass integer data between COBOL, C, C++, or PL/I programs in a COMMAREA, the data items must be declared in COBOL as COMP-5; otherwise, the byte ordering of the data is incorrect and the values are corrupted.

See the *CICS Application Programming Reference* and the *CICS Intercommunication Guide* for related information.

Timer services

CICS timer services, in conjunction with the time-of-day clock maintained by the operating system, provide commands that can be performed at a specific time. Using these commands you can:

- Ask for the time (EXEC CICS ASKTIME)
- Delay the processing of a task (EXEC CICS DELAY)
- Start a task and store data for the task (EXEC CICS START)

- Retrieve data stored (by an EXEC CICS START command) for a task (EXEC CICS RETRIEVE)
- Cancel the effect of a previous EXEC CICS DELAY or EXEC CICS START command (EXEC CICS CANCEL)
- Suspend the task (EXEC CICS SUSPEND)

Expiration times

The time at which a timer service is to be started is called the *expiration time*. You can specify expiration times absolutely, as a time of day (using the TIME option), or as an interval that is to elapse before the function is to be performed (using the INTERVAL option).

Expiration times cannot be defined using TIME or INTERVAL options in C. When using the EXEC CICS DELAY command, you can use the FOR and UNTIL options, and for the EXEC CICS START command, you can use the AFTER and AT options.

Use INTERVAL to start a transaction after a specified number of hours, minutes, and seconds (expressed as *hhmmss*) have elapsed from the current time. INTERVAL always specifies a time in the future: the current time plus the interval you specify, assuming you specify a nonzero interval.

Use TIME to start a transaction at a specific time; again using hours, minutes, and seconds (expressed as *hhmmss*). An absolute time is measured relative to midnight before the current time and may therefore be earlier than the current time. TIME may be either in the future or the past relative to the time at which the command is executed. For example, to start a transaction at 1530 hours, you would use EXEC CICS START TIME(153000). The following rules apply:

- If you specify a task to start at any time within the previous six hours, it will start immediately, unless the start time is before midnight (past) of the day on which you specify it. For example, the following command issued at 0500 or 0700 hours on Monday expires at 1230 hours on the same day.:

  ```
  EXEC CICS START TIME(123000)
  ```

  The following command issued at 0500 or 0700 hours on Monday expires immediately, because the specified time is within the preceding six hours.

  ```
  EXEC CICS START TIME(020000)
  ```

  The following command issued at 0500 hours on Monday expires immediately, because the specified time is within the preceding six hours. However, if it is issued at 0700 hours on Monday, it expires at 0030 hours on Tuesday, because the specified time is not within the preceding six hours.

  ```
  EXEC CICS START TIME(003000)
  ```

  Note that the TIME given is never taken to be before midnight of the current day.
- If you specify a time with an hours component that is greater than 23, you are specifying a time on a day following the current one. For example, a time of 250000 means 0100 hours on the day following the current one, and 490000 means 0100 hours on the day after that.

  If you do not specify either INTERVAL, TIME, FOR or UNTIL on the EXEC CICS DELAY command, or INTERVAL, TIME, AFTER or AT on the EXEC CICS START command, INTERVAL(0) is assumed, which means immediately.

The started interval control element (ICE) will be recovered only if it has been started with the PROTECT option specified. For example, if an EXEC CICS START command for a transaction to be started at 0700 hours is issued at 0600 hours with

the PROTECT option, and CICS fails at 0630 hours and is warm started at 0800 hours, the started transaction will immediately run.

Because each end of an intersystem link may be in a different time zone, you should use an interval if the transaction to be started is in a remote system. An absolute time is always relative to the local system and converted into an interval before shipping.

The interval to delay the EXEC CICS START request applies from the time of delivery to the remote system. Therefore, if the request is locally queued because the remote system was unavailable, the EXEC CICS START request may be delayed longer than expected.

If the system fails, the EXEC CICS START requests you have made that have not expired are recovered.

Request identifiers

As a means of identifying the request and any data associated with it, a unique request identifier is assigned by CICS to each EXEC CICS DELAY or EXEC CICS START command. You can specify your own request identifier by means of the REQID option; if you do not, CICS assigns (EXEC CICS START only) a unique request identifier and places it in field EIBREQID in the EXEC interface block (EIB). Specify a request identifier if you want the request to be canceled at some later time by an EXEC CICS CANCEL command.

START TRANSID commands

In a transaction that uses the EXEC CICS START TRANSID command to start other transactions, observe the following points to maintain logical data integrity:

- Always use the PROTECT option of the EXEC CICS START TRANSID command. This ensures that, if the start-issuing task is backed out, the new task will not be started.
- Designate the temporary storage queue used for passing data to the started transaction as recoverable. This ensures that data being passed to another task does not inadvertently stay on the temporary storage queue in the event of the start-issuing task being backed out.
- If REQID is used, that REQID is the name of a queue designated as recoverable in the Temporary Storage Definitions (TSD).

You can use the PROTECT option to ensure that if a system failure occurs after the EXEC CICS START that issued the logical unit of work (LUW) has completed its syncpoint, the transaction to be started will do so (as soon as CICS has been initialized and the terminal associated with the task is available).

If you are using CICS Clients, you should refer to the procedure in the *CICS Administration Guide* for further guidance.

See the *CICS Application Programming Reference* for related information.

Synchronization services

The CICS synchronization services allow you to serialize access to critical resources. You do this using the EXEC CICS ENQ and EXEC CICS DEQ commands. Each task that is to use the resource issues an EXEC CICS ENQ command. The first task to do so has use of the resource immediately, but

subsequent EXEC CICS ENQ commands for the resource, issued by other tasks, result in those tasks being suspended until the resource is available. You can override the suspension of a resource by issuing a EXEC CICS HANDLE CONDITION ENQBUSY.

Each task using a resource should issue a EXEC CICS DEQ command when it has finished with the resource. A task must issue the same number of EXEC CICS DEQ commands as the number of successful EXEC CICS ENQ commands, in order to free the resource for other tasks.

The name passed to the EXEC CICS ENQ is purely symbolic: no physical entity is locked. It is up to application designers to agree standards to be followed in using EXEC CICS ENQ commands, and to ensure that all programs follow the chosen rules.

For example, suppose that Program 1 issues the following command, where *phonenum* is a data area containing a character string containing a phone number:

```
EXEC CICS ENQ RESOURCE(phonenum) LENGTH(12)
```

Program 2 then issues the same command. Because each EXEC CICS ENQ command specifies the same character string contents, synchronization occurs and Program 2 waits until Program 1 issues the command:

```
EXEC CICS DEQ RESOURCE(phonenum) LENGTH(12)
```

It is the matching character strings that cause the synchronization. The same effect would be achieved by different RESOURCE option values, provided that the contents of the string at each location match. For example:

```
EXEC CICS ENQ RESOURCE(areaone) LENGTH(12)
EXEC CICS ENQ RESOURCE(areatwo) LENGTH(12)
```

Similarly, you can enqueue on the address of a data area. For example, if two programs both issue the following command:

```
EXEC CICS ENQ RESOURCE(resname)
```

The two programs enqueue on the same location and synchronize, even though the contents are not locked in any way. However, if two tasks are waiting for the same resource, which one gets the enqueue, when available, is dependent on the priorities on the two tasks; the enqueue goes to the task scheduled first.

See the *CICS Application Programming Reference* for related information.

Storage services

CICS storage services provide several commands and storage areas for your applications to use. Broadly, there are two types of storage available to application programs:

- Task-private storage, obtained for the sole use of the currently running transaction.
- Task-shared storage, to be shared in an application-dependent manner between transactions.

In addition, CICS has its own private shared storage.

Task-private storage

Task-private storage is private to the task and cannot be addressed by any other CICS task in the system. If you need working storage in addition to that provided automatically by the COBOL, C, C++, and PL/I languages, you can use the following commands. You can initialize each byte of the acquired storage to any bit configuration, for example to zeros or blanks. You can get and initialize main storage by using the EXEC CICS GETMAIN comand. You can release main storage when you are done using it by using the EXEC CICS FREEMAIN command.

CICS releases all task-private storage associated with a task when the task is ended normally or abnormally. This includes any EXEC CICS GETMAIN storage acquired, and not subsequently freed, by your program. CICS also provides your program with these named task-private storage areas:

- The EXEC Interface Block (EIB) holds information about the last command that CICS executed for your program. For further information, see "EXEC interface block (EIB)" on page 187.
- The COMMAREA is a communication area that you can use to pass information between one program and the next in the same task. For further details, see "Passing data to other programs" on page 175.
- The Transaction Work Area (TWA) is an area that you can use within the transaction. For further information, see "Transaction work area (TWA)" on page 26.

Task-shared storage

Task-shared storage, also known as the *task-shared pool*, is shared between all CICS tasks. As such, all synchronization to these areas is the responsibility of the applications that wish to use them. You can:

- Allocate an area of shared storage (EXEC CICS GETMAIN SHARED)
- Release the allocated shared area (EXEC CICS FREEMAIN)
- Load a map set or data table (EXEC CICS LOAD)
- Release a map set or data table (EXEC CICS RELEASE)

Note: Data tables can be used to load an operating system file into CICS storage for manipulation by an application. The files are loaded (by the EXEC CICS LOAD command) and the address of the storage is passed to the application. This is useful as transactions can be written to take data and place it into either SFS or a database, or can use it directly in a transaction. The **ProgType** attribute (described in the *CICS Administration Reference*) specifies whether the item is a program, a map set, or a data table.

The SHARED option of the EXEC CICS GETMAIN command can be used to acquire storage that is shareable between transactions. When you issue an EXEC CICS GETMAIN SHARED command, you can pass the address of that area of storage to another transaction in three ways:

- Using the common work area (CWA)
- Using a shared CICS area
- Using a shared operating system area

It is also possible to pass the address from one task to the next at the same terminal using the COMMAREA.

It is the responsibility of application programmers to keep track of storage area addresses allocated by EXEC CICS GETMAIN SHARED. Unlike ordinary EXEC CICS GETMAIN storage there is no implicit means of freeing it. Such storage areas

should be released by a call to EXEC CICS FREEMAIN by some application program (in a set of cooperating tasks). Otherwise the shared storage will not be freed until region termination.

The EXEC CICS LOAD command allows you to load a named map set or data table into shared storage, and by using the HOLD option keep it there. This is useful to prevent the same map set or table being repetitively loaded, although CICS itself tries to keep the entity loaded if it can anyway. On a multi-locale system, there could be several map sets with the given name (one per locale) and CICS actually loads all such map sets that it finds.

You use the EXEC CICS RELEASE command to release a named map set or data table that you have loaded. This will not necessarily result in the map set being physically released from shared storage, as CICS will delay this until it is necessary to do so.

You can also use MAIN temporary storage as a scratchpad area to be shared between transactions. For further details, see "Temporary storage queue services" on page 133.

CICS also provides your program with these named task-shared storage areas:

- The *common work area (CWA)* is a storage area allocated at region startup and exists for the duration of the CICS region. For further details, see "Common work area (CWA)" on page 28.
- The *terminal user area (TCTUA)* is a storage area allocated either when you log on (for an autoinstalled terminal) or at system startup (for nonautoinstalled terminals). For further information, see "Terminal user area (TCTUA)" on page 29.
- The *COMMAREA* is a communication area that you can use to pass information between one transaction and the next in a pseudoconversational sequence. For further details, see "Passing data to other programs" on page 175.

See the *CICS Application Programming Reference* for related information.

CICS private shared storage

CICS private shared storage, also known as the *region pool*, is private to CICS, but two commands enable you to obtain information from it and from task-shared and task-private storage. You can:

- Get access to these storage areas using the EXEC CICS ADDRESS command
- Get values from these storage areas using the EXEC CICS ASSIGN command. The values returned to the application by the EXEC CICS ASSIGN command are copied from the region pool. The application program has no access to the region pool other than through CICS commands.

See the *CICS Application Programming Reference* for related information.

Logical unit of work (LUW) services

If an individual task fails, backout is performed automatically by CICS. If the CICS system fails, backout is performed as part of the auto start process.

However, for long-running programs, it may be undesirable to have a large number of changes, accumulated over a period of time, exposed to the possibility of backout in the event of task or system failure. This possibility can be avoided by

using the EXEC CICS SYNCPOINT command to split the program into logically separate sections termed *logical units of work (LUWs)*; the end of an LUW is called a *synchronization point (syncpoint)*.

If failure occurs after a syncpoint but before the task has been completed, only changes made after the syncpoint are backed out.

LUWs should be entirely logically independent, not merely with regard to protected resources, but also with regard to execution flow. Typically, an LUW comprises a complete conversational operation bounded by EXEC CICS SEND and EXEC CICS RECEIVE commands. A BROWSE is another example of an LUW. An EXEC CICS ENDBR command must therefore precede the syncpoint.

Possibility of transaction deadlock and its avoidance

The enqueuing mechanisms that protect resources against double updating can cause a situation known as *transaction deadlock*. Transaction deadlock is sometimes known as *enqueue deadlock* or *enqueue interlock*.

As shown in Figure 9, transaction deadlock means that two (or more) tasks cannot proceed because each task is waiting for the release of a resource that is locked by the other task. (Remember that the locking action protects resources until the next syncpoint is reached.)

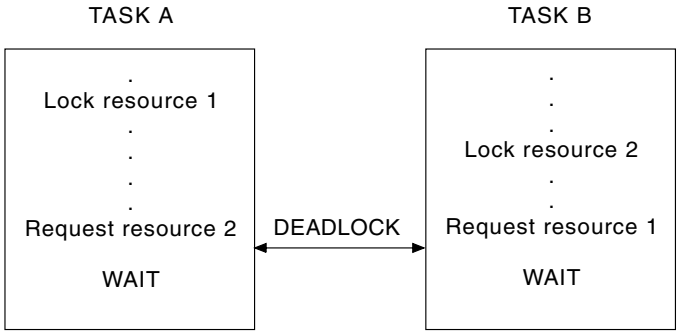


Figure 9. Transaction deadlock (generalized)

If transaction deadlock occurs, one task may be abnormally terminated and the other is allowed to proceed.

If both resources are CICS resources, the task whose deadlock timeout period elapses first is abnormally terminated and its CICS resources are released. (It is possible for both tasks to time out simultaneously.) If neither task has a *DeadLockTimeout* attribute value in the Transaction Definitions (TD), they will both remain suspended indefinitely, unless one of them is abnormally terminated.

CICS then backs out the abnormally terminated task.

Techniques for avoiding transaction deadlock

To avoid transaction deadlock, consider the following techniques:

- Arrange for all transactions to access resources in a predefined sequence. This might be considered a suitable subject for installation standards.

  Extra care is required if you allow updates through multiple paths.
- Explicit enqueueing conventions should also be the subject of your site development standards so that all applications:

  - Enqueue using the same character string
  - Use those strings in the same sequence
- Always access records within a file in the same sequence. For example, where multiple file or database records have to be updated, ensure that they are accessed in ascending sequence. Ways of doing this include:
  - The terminal operator should always enter data in the same sequence as it exists on the data set.

    This method requires special terminal operator action, which may not be practical within the constraints of the application. (For example, orders may be taken by telephone in random product number sequence).
  - The application program first sorts the input transaction contents so that the sequence of data items matches the sequence on the data set.

    This method requires additional application programming, but imposes no external constraints on the terminal operator or the application.
  - The application program issues a user EXEC CICS SYNCPOINT command after processing each data item entered in the transaction.

    This method requires less additional programming than the second method. However, issuing a user syncpoint implies that previously processed data items in the transaction will not be backed out if a system or transaction failure occurs before the processing of the entire transaction is completed. This may not be valid for the application, and raises the question as to which data items in the transaction were processed and which were backed out by CICS. If the entire transaction must be backed out, user synchronization points should not be issued, or only one data item should be entered per transaction.

  Of the three methods, the second method (sorting data items into an ascending sequence by programming) is most widely accepted.

See the *CICS Application Programming Reference* for related information.

Configuration services

CICS provides a number of facilities that enable applications to find out information about the environment in which they are running:

- The EXEC CICS ADDRESS and EXEC CICS ASSIGN commands provide information about fields in CICS control blocks.
- The INQUIRE and SET commands provide information about and control of resources.
- The EXEC Interface Block (EIB) provides information about task-related control blocks.

EXEC CICS ADDRESS and EXEC CICS ASSIGN commands

You can write many application programs using the CICS API commands without any knowledge of or reference to the fields in the CICS control blocks and storage areas. However, you might sometimes need to get CICS information that is valid outside the local environment of your application program. Use the EXEC CICS ADDRESS and EXEC CICS ASSIGN commands to access such information.

Do not access CICS control block fields with the EXEC CICS ADDRESS and EXEC CICS ASSIGN commands explicitly specified as arguments on any EXEC CICS command, because specifying these commands ass arguments leads to a change of CICS fields by the EXEC interface modules.

INQUIRE and SET commands

The INQUIRE and SET commands allow application programs to access information about CICS resources. The application program can retrieve and modify information for CICS resources; such as files, terminals, system entries, mode names, system attributes, programs, and transactions.

EXEC interface block (EIB)

In addition to the usual CICS control blocks, each task has a control block called the EXEC interface block (EIB) associated with it.

An application program can access all of the fields in the EIB by name. The EIB contains information that is useful during the execution of an application program, such as the transaction identifier, the time and date (initially when the task is started, and subsequently, if updated by the application program), and the cursor position on a display device. The EIB also contains information that is helpful when a dump is being used to debug a program.

See the *CICS Application Programming Reference* for related information.

Part 2. Migrating Applications

Table 38. Road map for Migrating applications

If you want to...	Refer to...
Consider the migration process, programming compatibility, and API migration when migrating applications to and from CICS for Open Systems and CICS for Windows.	Chapter 7, "Migrating CICS applications to and from TXSeries CICS," on page 191

Chapter 7. Migrating CICS applications to and from TXSeries CICS

This chapter describes how to migrate application programs to and from TXSeries CICS (TXSeries CICS).

Note: In the migration chapter, the term *existing CICS system* refers to the system you are migrating from.

Preparing to migrate your applications

To migrate CICS applications from a non-TXSeries CICS system, such as CICS/MVS or CICS OS/2, you must consider the differences between the two systems. These differences can include, for example, the CICS API commands supported, or the source languages supported.

If you are planning to migrate from a CICS for Windows system, consult your IBM representative for support.

What is migration?

Migrating your existing CICS system to TXSeries CICS means moving your existing applications in such a way as to achieve the same level of functionality that you were using on your existing CICS system. Migration does not involve adding new functionality through new system features.

The information describing the migration process outlines the functionality of TXSeries CICS, so that you can compare it to your existing system, and identify any inconsistencies. Other CICS systems are described in general terms, without discussing the particular features of any one family member.

The amount of work required to migrate your CICS system depends on what you have done to your system in the past. If you use TXSeries CICS without much customization, and if your programs are written to conform to the supported external interfaces, much of the work of migrating is quite straightforward. You need to check your use of CICS functions against the changes, and then make any necessary adjustments to your programs and operating procedures.

If you have customized TXSeries CICS, or written system programs using unsupported interfaces, internal control blocks, and so on, you will need to reassess the way you use CICS and redesign your code to conform to supported interfaces.

When you consider migrating from an existing CICS system to TXSeries CICS, you should remember that it may not be necessary, or possible, to migrate all of your existing applications. Under these circumstances, you may consider continuing to run your existing CICS system as well as TXSeries CICS, and allowing communication between the two. This is known as coexistence.

Controlling the migration process

The complexity and variety of possible CICS configurations means that there is no simple list of instructions for migration planning and implementation. However, by

combining the general guidance provided as part of migration planning and control with the descriptions of changes elsewhere in the migration topic, you should be able to produce a sound framework on which to build your migration strategy.

Migrating to TXSeries CICS requires careful planning and change management. To achieve this, you will need to:

- Understand the practical issues and the detailed product changes involved in TXSeries CICS migration
- Plan a satisfactory conversion strategy
- Plan in detail the required actions to achieve the strategy
- Estimate with reasonable accuracy the resources and time involved.

You can maintain service levels to users during the migration process by using sound change-management practices. To achieve this, you will need to:

- Minimize the number of changes that take place simultaneously
- Test changes before putting them into production
- Have a backout or fallback strategy for each change in case of failure in production
- Be able to assess the impact of a change before introducing it into the production system
- If necessary, be able to limit the extent or localize the impact of any change to an acceptable level
- Be able to schedule the change or cutover to minimize the impact on end users or the business.

Your ability to meet these change-management needs depends partly on what you have done with your existing CICS systems, and partly on the migration strategy you adopt.

Applying these general rules to TXSeries CICS migration, it is clear that a phased conversion and cutover is necessary. One possible migration strategy consists of the following phases:

- Planning your migration
- Installing, testing, and parallel running
- Phased cutover.

The migration planning phase

The objective of the planning phase is to develop a migration strategy with detailed plans for each step. It should include the following topics:

- Considering corequisite and prerequisite products, such as the specific release of the operating system, DCE, relational databases, and compilers.
- Planning to migrate to the appropriate levels of corequisite and prerequisite products where necessary. You need to consider ordering and delivery lead times when calculating your schedules.
- Planning your education and training requirements.
- Planning to update your operational procedures and the supporting documentation.
- Planning the installation of any features or parameters required in the operating system for TXSeries CICS.

- Estimating and planning the machine resources required for the migration, for example disk space, terminals and machine time for installation, development and testing.
- Determining which resources (tables, programs, and so on) need to be frozen during the migration process.
- Determining where you may require programming changes and planning the necessary coding and testing.
- Planning to update your programming standards and the supporting documentation.
- Planning the installation of TXSeries CICS.
- Considering the actions required by any associated products that have been added to TXSeries CICS.
- Planning conversion steps, such as changes to tables that are not to be frozen during the migration process, and installation of resources.
- Determining your testing criteria.
- Planning the cutover phases.
- Determining the best methods of keeping users informed of any changes to functions or services that might affect them.
- Planning to update your problem determination procedures and the supporting documentation.
- Considering any issues raised by your existing CICS system and TXSeries CICS in parallel.

The end product of the planning phase should be a formally documented plan that you can use for project management and control.

The migration installation phase

This phase has two objectives:

- To install and customize the basic TXSeries CICS system and to make it ready for testing and parallel running.
- To install the programs and procedures needed to support TXSeries CICS processing; such as monitoring and accounting packages, statistics printing, dump printing, journal handling, and so on.

When these two objectives are met, you are ready to migrate your application programs and resource definitions.

Migration phased cutover

The objective of this phase is to migrate the working CICS systems to TXSeries CICS. Note that TXSeries CICS does not support the CICS facility of multiple region operation (MRO). If you want to migrate a CICS system that uses MRO, then some distributed transaction processing (DTP) commands in your application programs may need to be changed.

Regardless of the environment you choose to operate in, you may need to:

- Migrate your application
- Migrate your data
- Reenter all the associated definitions that defined your CICS application and users.

The following discussion assumes that you have already installed the required version of operating system and any necessary supporting software. The discussion

also assumes that you have recoded your programs to comply with the TXSeries CICS API and removed dependency on MRO. Remember that the objective of the migration exercise is to migrate CICS applications and their supporting environment to TXSeries CICS, it should not be an objective of the migration exercise to increase the functionality of the migrated applications. Attempting to combine the two objectives is liable to result in an overly complex project that may be difficult to control and support.

Wherever possible, it is advisable to operate both your existing TXSeries CICS and other CICS systems in parallel until you can prove that your applications are operating correctly; as a minimum you should ensure that you have an adequate fallback position if the migration is not successful.

Single-region systems: A single-region system contains all the programs, definitions and data that you require to operate your application and can be the most difficult systems to migrate because of the possibility of having to make large numbers of changes to your applications and user environment.

Multi-region systems: A multiregion system is, for example, a system that separates the region that contains your application from the region that contains the data with the two regions using communications to provide a seamless interface to the end user. TXSeries CICS has all the same communication facilities as other CICS systems (except MRO) so that you may choose to migrate one or both halves of the application. The ability to migrate the constituent parts of your application only when you deem it necessary or practical may reduce the amount of conversion effort that you have to expend, and consequently the amount of direct support to your users.

Non-migrated regions or applications: You may choose not to migrate some applications that TXSeries CICS does not support, for instance macro-level applications, but to leave them on your existing CICS system.

- If you consider that an application is going to become obsolete in the foreseeable future, it may not be cost effective converting it.
- If the cost to convert an application or its data is high, it may not be cost effective converting them.

In these cases you can leave the application and its associated data on the existing CICS system and access them through CICS Intercommunication facilities. In brief, plan to convert only those of your applications not supported by TXSeries CICS where it is economic to do so.

Coexistence strategies

It can be impossible to migrate all of your existing CICS production regions to TXSeries CICS without substantial effort. For example, you may have some macro-level applications that are not supported in TXSeries CICS. In these instances, you must devise a strategy for migrating some applications to TXSeries CICS, leaving behind those applications that you cannot cost effectively migrate or convert. This information outlines how your existing CICS system can coexist with a TXSeries CICS system using intersystem communication (ISC). ISC is used in all CICS products to provide communication between CICS systems on separate processors across a network.

If you are running a CICS system that has elements that you cannot migrate to TXSeries CICS, you may need to consider adopting an ISC solution that enables

you to transaction route between your existing CICS system and TXSeries CICS. For example, your system may comprise some or all of the following components:

VTAM and BTAM terminals
These are not supported in TXSeries CICS (except as remote terminals for ISC purposes), and you cannot install them on TXSeries CICS systems. To support transaction routing from the CICS system that owns the terminals, you should either ensure that the terminal definitions on the existing CICS system are marked shippable, or on TXSeries CICS, generate Terminal Definitions (WD) entries that specify the terminals as remote terminals.

Note that if the system to which your *VTAM* or *BTAM* terminal is connected has a telnet client, then you may be able to use this to gain access to a region using the **cicsteld** client.

Note: VTAM refers to *Virtual Telecommunications Access Method*, a set of programs that maintain control of the communication between terminals and application programs. BTAM refers to *basic telecommunications access method*, an access method that permits read and write communication with remote devices.

Command-level applications
You can migrate command-level applications provided they conform to the TXSeries CICS program compatibility rules.

Files If the migrated applications access files that remain on your existing CICS system, you need to define the files as remote resources in TXSeries CICS. You can access the files by function shipping the file requests.

If you decide to use ISC, you will have to specify resource definitions that define connections between CICS systems.

When you have migrated the appropriate elements of your old system, there are several points to consider.

- To access migrated applications from your original terminals you must define the transactions as remote transactions on your existing CICS system and define the terminal as remote in the Terminal Definitions (WD) on TXSeries CICS.
- To access applications that you have not migrated to TXSeries CICS, from an existing CICS terminal, define the transactions as remote in the Transaction Definitions (TD) on TXSeries CICS. You need to define the terminal as remote to TXSeries CICS.

  Note: There is an alternative to the creation of a remote definition of a terminal in any remote system to which it wants to direct a transaction routing request. This is to make the terminal's local definition in the terminal owning region shippable with the Terminal Definitions (WD) **IsShippable** attribute. If a terminal definition is shippable, sufficient data is passed with a transaction routing request to enable the remote system to dynamically install the necessary remote terminal definition. For further information about shippable terminals, refer to the *CICS Administration Reference* .

Tests and parallel running

The following sections describe tests and parallel running.

Installation verification procedures

To check that you have installed CICS correctly, TXSeries CICS provides optional Installation Verification Procedures (IVPs) for you to perform. The IVPs comprise a set of programs, transactions, and maps, plus a set of instructions for their use.

Note: The IVPs provided with your existing CICS system cannot be used to test the installation of TXSeries CICS systems.

Testing the running TXSeries CICS

When you have completed installing TXSeries CICS and running the IVPs to ensure that installation has been successful, you can begin to test and tailor the system to meet your specific requirements. The tests you may need to run include:

- Testing all resource definitions
- Testing system initialization options, particularly where you use system default values, which are different from your existing CICS system
- Testing the TXSeries CICS start-up procedure
- Testing off-line utilities
- Testing any associated software products and packages
- Testing any programs that you have rewritten because they originally contained features TXSeries CICS does not support
- Testing any of your programs that use internal or external security checking
- Testing your applications

Running TXSeries CICS in parallel with your existing CICS system

There are some planning and implementation considerations that concern running TXSeries CICS and another CICS product in parallel. You must consider the effects on your operational procedures. Some operational areas are affected by the differences between TXSeries CICS and other CICS systems. These areas include:

- Differences in initialization - messages and system initialization parameters
- Differences in the messages and codes issued, and the operator actions they require
- Differences in monitoring output
- Differences in dump output
- Differences in trace output
- Differences in statistics output
- Differences in problem determination, restart, and recovery requirements
- Differences in security administration (with CICS on Open Systems or CICS for Windows security or with an external security manager)
- Differences in running databases

Migrating data

TXSeries CICS provides the CALF transaction (Convert and Load File) to enable you to migrate data to your region. This transaction allows you to transfer any file for which you can create a File Definitions (FD) entry. In this way, data on remote VSAM files can be migrated. CALF works with function shipping.

CALF accepts the following three filetypes:

- Key sequenced data set (KSDS)
- Entry sequenced data set (ESDS)

- Relative record data set (RRDS)

The transaction copies records from a source file to a destination file. You enter control data in the data entry fields to control the transaction, and use the PF keys to run the transaction.

You start the CALF transaction directly by entering the transaction identifier CALF. You end the transaction by pressing PF3.

Migration and the CICS-supplied transactions

TXSeries CICS provides a subset of the CICS supplied transactions. In some instances, the functionality of these transactions within TXSeries CICS may differ from other CICS systems. TXSeries CICS provides the following supplied transactions:

- Data Conversion (CALF)
- Temporary Storage Browse (CEBR)
- Command Level Interpreter (CECI)
- Syntax Checker (CECS)
- Execution Diagnostic Facility (CEDF)
- Runtime Resource Management (CEMT)
- Signon (CESN)
- Signoff (CESF)
- Routing Transaction (CRTE)
- Application Diagnosis Configuration (CDCN) (CICS on Open Systems only)

A full description of the functionality of each of these transactions is provided in the *CICS Administration Reference*.

Some points to note regarding differences in supplied transactions between TXSeries CICS and other CICS systems are:

- TXSeries CICS does not support the supplied transaction CEDA. You must perform all resource definition using the resource definition online (RDO) facility.
- TXSeries CICS monitoring facilities are controlled by the CEMT transaction, not from the CEDA transaction as in other CICS systems.
- TXSeries CICS does not invoke shutdown from the CEMT transaction as in other CICS systems. Refer to the *CICS Administration Guide* for information about shutdown.
- You cannot run the CEBR transaction under CEDF.
- The CEDF transaction does not support the single-screen mode of operation. You must use two terminals (or two windows), one to run CEDF and one to run the transaction being debugged.
- Because TXSeries CICS uses DCE security for normal user authentication, you can only use the CESN transaction to connect to a region after using the CRTE transaction (transaction routing) or to temporarily change your user identity after connecting to a region.
- TXSeries CICS does not support a replaceable logon shell. The supplied CESN transaction cannot be replaced with a user-supplied module as with other CICS products.

Migration and CICS resource definitions

TXSeries CICS holds the resource definitions for a region in a set of files. These files fulfill the same role as the traditional tables used by other CICS family members, but to emphasize the differences in attribute requirements, the TXSeries CICS table names are called *definitions*. For example, the Transaction Definitions (TD) file replaces the Program Control Table (PCT). The correspondence between TXSeries CICS and other CICS family definitions is shown in the *CICS Administration Reference*.

Suffix support is available in TXSeries CICS, but not in the same way as in other CICS family members.

TXSeries CICS does not provide macro-level resource definition facilities. The facilities to perform the resource definitions are described in the *CICS Administration Reference*. Resource definitions cannot be exported.

TXSeries CICS provides the resource group facility as in other CICS family members. A resource may belong to at most one group. TXSeries CICS does not provide support for the resource list facility that other CICS family members provide.

Migration and programming compatibility

During migration to TXSeries CICS, you should review your CICS application and system programs, and any CICS-related programs that process CICS output, to ensure that they are compatible with TXSeries CICS. You need to consider:

- The source languages and compilers that TXSeries CICS supports (described in "Source language and compiler considerations for migration").
- The application program interface (API) subset that TXSeries CICS supports. For information about the differences in the API across CICS products, refer to *CICS Family: API Structure*.
- Your Basic Mapping Support (BMS) requirements. TXSeries CICS provide a subset of BMS support (see "BMS functions supported in CICS" on page 72). Remember that besides your original map requirements, application programs may also embed BMS commands.
- Other programming considerations such as embedded database commands and macro-level programs.

In keeping with the rest of the migration documentation, the discussion of the topic of program compatibility focuses on the support that TXSeries CICS provides, rather than on the differences between a specific CICS family member and TXSeries CICS.

You must fully test all programs, whether you change them or not, before you introduce them into your production system. Where possible, such testing should be done on your existing CICS system before migrating your programs to TXSeries CICS, as this narrows the scope for errors, and makes resolution easier.

Source language and compiler considerations for migration

TXSeries CICS supports application programs written in COBOL, C, C++, or PL/I (refer to the installation documentation for information about versions of COBOL, C, C++, or PL/I compilers that can be used with your version of CICS and Table 44 on page 219 for a summary of the support on each platform).

TXSeries CICS does not support application programs written in source languages other than COBOL, C, C++, or PL/I. This means that if you do not discard existing application programs written in other languages you must recode them, or you must maintain your existing CICS system and run it in coexistence with the TXSeries CICS region.

Before you recompile any application programs in COBOL, C or C++ that you intend to migrate to TXSeries CICS, you must process the application program with the CICS translator. (Described in Chapter 8, "Translating, compiling, and link-editing CICS application programs," on page 213.) The compilation process highlights, as errors, any language syntax that is not supported.

Micro Focus Server Express COBOL upgrades On CICS on Open Systems only

COBOL programs should be recompiled when you upgrade to a new level of COBOL compiler.

Note: In addition to recompiling COBOL programs when you upgrade the Micro Focus Server Express COBOL compiler, the **cicsmkcobol** command must be run. This command combines the object files containing the CICS COBOL support routines with the COBOL runtime system to create a single loadable and executable file named **cicsprCOBOL**. Refer to the *CICS Administration Reference* for more information.

COBOL base locater linkage (BLL) cells

If your existing application programs use BLL cells, you will need to recode them before porting them to TXSeries CICS as the supported COBOL compilers do not support them.

Service reload statements

If your existing application programs use SERVICE RELOAD statements, you will need to recode them before porting them to TXSeries CICS as the supported COBOL compilers do not support them.

See the *CICS Family: Interproduct Communication*, the *CICS Intercommunication Guide*, and .

The *CICS Administration Reference*.

Other programming considerations for migration

As well as API, source and compiler, and Basic Mapping Support (BMS) considerations, you must also consider other factors when you are migrating application programs to a TXSeries CICS region. These considerations are described in the following sections.

Macro-level applications

TXSeries CICS does not support macro-level application programs. If your existing CICS system contains macro-level applications which you may not discard, then you must do one of the following:

- Rewrite the applications to use the API commands that TXSeries CICS supports.
- Allow the applications to coexist with TXSeries CICS by continuing to run them in your existing CICS system.

Database systems

TXSeries CICS supports the inclusion of Structured Query Language (SQL) statements within application programs but you must separately process the SQL statements through your database manager. It does not support the data language 1 (DL/I) database access language. If your existing CICS system contains applications which use DL/I which you may not discard, then you must do one of the following:

- Rewrite the applications to use the databases that TXSeries CICS supports.
- Allow the applications to coexist with TXSeries CICS by continuing to run them in your existing CICS system.

Monitoring, dump, statistics, and trace post-processors

TXSeries CICS provides the offline utilities that format monitoring, dump, statistics, and trace output. Those utilities are:

- **cicsmfmt**
- **cicsdfmt**
- **cicssfmt**
- **cicstfmt**

As these records are in a format unique to TXSeries CICS, any existing formatting programs will not work unless you modify them.

Journal post-processors

TXSeries CICS does not provide off-line utilities that format user and system journals. As the journal records are in a format unique to TXSeries CICS, any existing formatting programs will not work unless you modify them.

CEMT programmable interface

TXSeries CICS does not support a programmable interface to the supplied transaction CEMT.

Short-on-storage conditions

The underlying main storage architecture for TXSeries CICS differs from that provided by other CICS family members in the handling of short-on-storage conditions. Instead of always suspending a running task (unless you define the task with NOSUSPEND), TXSeries CICS responds in a number of ways depending upon the type of storage request made, and whether storage is requested by an application program or internally by CICS. CICS may:

- Return a CICS condition code to application program.
- Raise a CICS abnormal termination code on the application program, or the CICS runtime system.
- Suspend the running task pending the acquisition of task shared storage for loading Basic Mapping Support (BMS) maps and data tables. TXSeries CICS may subsequently abnormally terminate the application program if storage remains unavailable for some time.

While application programs that you migrate may translate, compile and link correctly, TXSeries CICS may raise a number of additional abnormal termination codes, due to short-on-storage conditions. You may need to change your application programs to handle such conditions.

Hexadecimal character representation

TXSeries CICS represents characters in ASCII format and IBM mainframe-based CICS represents characters in EBCDIC format. If programs move specific hexadecimal values to represent EBCDIC characters (or define variables to have

specific hexadecimal values), you need to change these values to the equivalent ASCII representations. Alternatively, if you are using Micro Focus Net Express version 3.0 or later on a Windows platform, you can compile EBCDIC-enabled COBOL programs that run on a Windows workstation. See "Compiling EBCDIC-enabled COBOL programs" on page 57 and "Using Micro Focus Net Express to compile EBCDIC-enabled COBOL programs" on page 227 for more information.

Static and global data

A TXSeries CICS application program that is loaded into at least two logical levels of the same task should not use static or global data because the operating system shares the programs data segment for this type of data. For example, program A has a static integer called x that has an initial value of **zero**. In the course of processing, program A sets x to **1** and then performs an EXEC CICS LINK PROGRAM(A) command. The second invocation of program A sees x set to **1** and not **zero**.

See "Application program logical levels" on page 174.

Migration and the API

When considering moving an application to TXSeries CICS, you should compare the CICS application programming interface (API) commands that the application uses with those supported by TXSeries CICS, which supports a subset of the full list of CICS API commands. Note that some API commands have different sets of options on TXSeries CICS from the options on other CICS products.

This section describes those aspects of application programming that help you to write portable applications and to migrate applications to and from TXSeries CICS.

Overview of migration and the API

If you need to migrate an application program from your existing CICS system that uses API commands or options that TXSeries CICS does not support, you must either:

- Modify the application to use API commands TXSeries CICS does support, or
- Allow the application to coexist with TXSeries CICS by continuing to run the application in your existing CICS system.

The full list of API commands and options supported by CICS are described in the *CICS Application Programming Reference*. Differences between the TXSeries CICS-supported API and that supported by other CICS family members are given in *CICS Family: API Structure*.

If you do not remove the API commands or options that TXSeries CICS does not support from an application program, you can use the CICS command language translator to highlight the unsupported commands or options as errors. This is a quick way of finding out if an existing program will run under TXSeries CICS without conversion or, conversely, this method enables you to more accurately estimate the amount of effort you will require to migrate an application program to the TXSeries CICS API.

In all CICS products, you can place command options in any sequence in a command, as long as you place the command identifier first. CICS family products interpret the scope of a command identifier differently; some consider just the first

word to be significant (such as HANDLE) while others, of which TXSeries CICS is one, consider one or more words to be significant (such as EXEC CICS HANDLE CONDITION). Thus, HANDLE RESP CONDITION and EXEC CICS SET FILE NOHANDLE OPEN are invalid in TXSeries CICS. You can tell how many words are considered to be significant in a command identifier by looking at its title in the *CICS Application Programming Reference*.

Presentation services API migration

There are two topics that deal with migration of application programs that use the presentation services API:

- Migration and Basic Mapping Support (BMS) services
- Migration and 3270 Information Display System datastreams

Migration and Basic Mapping Support (BMS) services

TXSeries CICS provides support for minimum function BMS support and some standard function BMS support (see "BMS functions supported in CICS" on page 72). This means that you can migrate BMS macro source code at this level to TXSeries CICS.

The BMS processor will highlight any BMS function that TXSeries CICS does not support.

If a map in your existing CICS system (the system you are porting from) uses other than minimum function BMS and you cannot discard the map, then you must either:

- Modify the map to use only minimum function BMS, or
- Allow the applications using the map to coexist with TXSeries CICS by continuing to run them in your existing CICS system.

You can specify the **PS** parameter in BMS maps (see the *CICS Application Programming Reference*) as one of the following:

- PS=8
- PS=X'F8' (EBCDIC)
- PS=X'38' (ASCII)

When an application routes a transaction from an EBCDIC system, TXSeries CICS converts the sent X'F8' value to X'38'. When a TXSeries CICS application routes a transaction, the EBCDIC system converts the sent X'38' value to X'F8'. Application programs that you migrate to TXSeries CICS and that explicitly set PS=X'F8' will continue to work. Applications that you develop for TXSeries CICS that set PS=X'38' work on TXSeries CICS but do not work if you migrate them to an EBCDIC system.

It is advisable that you always code PS=8.

Using the DFHMDF OCCURS option: When the OCCURS option is specified in a BMS map, an array is generated in the symbolic map. For example, on CICS OS/2 this array is present in the output map only while a filler is put in the input map. TXSeries CICS generates the array in both maps.

Refer to the following map:

```
DFHTEST  DFHMSD TIOAPFX=YES,STORAGE=AUTO,MODE=INOUT,LANG=C,                *
               CTRL=(EXEC CICS FREEKB,FRSET)
MAPNAME  DFHMDI SIZE=(24,80)
*
OCCURS   DFHMDF POS=(01,02),LENGTH=15,OCCURS=12,INITIAL='TEST'
         DFHMSD TYPE=FINAL
```

This map generates the header file:

```
/*
 * cicsmap freda.bms -- Tue Oct 27 10:10:06 CUT 1992
 * DFHTEST DFHMSD MODE=INOUT,STORAGE=AUTO
 */
struct mapnameis
{
    char      tioapfx[12];
    struct
    {
         short     occursl;
         char      occursf;
         char      occursi[15];
    }         occursd[12];
};
struct mapnameos
{
    char      tioapfx[12];
    struct
    {
         short     filler01;
         char      occursa;
         char      occurso[15];
    }         occursd[12];
};
union mapnameu
{
    struct mapnameis mapnamei;
    struct mapnameos mapnameo;
};
union mapnameu mapname;
```

Notice that the **occursd** structure is named in both the input map **mapnameis** and the output map **mapnameos**. This enables the structure to be used for both input and output operations.

Migrating maps from CICS OS/2: When generating BMS maps in C or C++, CICS OS/2 always assumes the When generating BMS maps in C, CICS OS/2 always assumes the parameter STORAGE=AUTO while TXSeries CICS does not. To migrate between the two implementations you must either specify STORAGE=AUTO in the CICS OS/2 map definition or amend the source program code to refer to the generated pointer rather than the structure.

This also applies if you are migrating maps from CICS for Windows.

Migrating maps from your CICS system to other family members: TXSeries CICS allows you to include fields with the same name on different maps within a map set. This is not the case in other CICS products. If you wish to port your map set from TXSeries CICS to IBM mainframe-based CICS, you should ensure that all BMS map and field names are unique within a map set.

The **cicsmap** command checks that the TYPE= operand specifies either MAP, DSECT, or &SYSPARM. However, the value is ignored by TXSeries CICS. Map generation is controlled with the **-p** (physical) and **-s** (symbolic) **cicsmap** flags. If neither flag is specified, both physical and symbolic maps are generated.

The **cicsmap** command accepts the TYPE=&SYSPARM operand to aid portability to CICS/MVS.

Migration and 3270 Information Display System datastreams

When migrating from an EBCDIC environment to TXSeries CICS, you need to change application programs that explicitly define 3270 datastreams in hexadecimal format. If the character representation is used, it will be converted to the correct ASCII hexadecimal code.

There are also some exceptions. For example, the *start field extended* special character in ASCII is X'10' but in EBCDIC is X'29'. The *start field extended* special character is not generally supported in ASCII, but has been supported here as a special case.

Migration requirements: These requirements apply equally to application programs, COBOL copybooks, C and C++ include files, and PL/I include files.

Where programs assign COBOL copybook, C or C++ or PL/I include field values to 3270 datastreams, you only have to make changes to the copybook, or include, and recompile the program.

You need to change:

- Any hexadecimal values or bit masks assigned to translated 3270 datastreams
- Any character values assigned to untranslated 3270 datastreams

You do not need to change:

- Any hexadecimal values or bit masks assigned to untranslated 3270 datastreams
- Any character values assigned to translated 3270 datastreams

Setting 3270 datastreams: Application programs can set 3270 datastream values by:

- Copying a COBOL copybook, or C or C++ or PL/I include field value
- Copying a program field value
- Assigning a character value
- Assigning a hexadecimal value
- Building up a bit mask

Untranslated 3270 datastreams: TXSeries CICS does not translate the following 3270 datastream components:

- Write control characters (WCC)
- AttributeTypes
- NumberOfPairs
- Data

Conversion of datastreams: TXSeries CICS communicates with its terminals using ASCII 3270 datastreams. These consist of control sequences and data and may be constructed from a number of different elements:

- INITIAL or GINIT data from a BMS map
- XINIT data from a BMS map
- Data in a symbolic map
- Attribute values in a symbolic map
- Data for terminal control output
- Control sequences for terminal control output
- Write Control Character given by the CTLCHAR option of EXEC CICS SEND

The XINIT data in a BMS map can be transformed from EBCDIC using the **-e** option of **cicsmap** that interprets the data as EBCDIC and automatically converts it to ASCII.

Unless you are using extended attributes, all the other sequences may be portably coded using character values or values from the standard copybook DFHBMSCA. Hexadecimal values or bit masks are non-portable and will need to be modified when moving an application from a non-ASCII system.

Special conversions: TXSeries CICS performs special translations on the following control sequences because they have no standard representation in ASCII datastreams:
- StartFieldExtended
- SetAttribute
- ModifyField

StartFieldExtended and **ModifyField** are represented as follows:

```
Code: NumberOfPairs : AttributeType : AttributeValue : ..
```

and **SetAttribute** is represented as follows:

```
Code: AttributeType : AttributeValue
```

where Code is X'10' for **StartFieldExtended**, X'1A' for **ModifyField**, and X'1F' for **SetAttribute**.

NumberOfPairs is a hexadecimal value, as is the **AttributeType**. These will have the same value on both ASCII and EBCDIC systems.

For all extended attributes, except for **Field Validation** and **Field Outlining**, which TXSeries CICS does not support, the **AttributeValue** is a graphic character and can be coded portably as a character value or as a defined value in the DFHBMSCA copybook.

Data services API migration

There are two topics that deal with migration of application programs that use the data services API:
- File services
- Queue services

File services

If you are migrating application programs that use the API file services, be aware of the following migration concerns:
- EXEC CICS READNEXT and EXEC CICS READPREV with an alternate index
- VSAM emulation
- Exclusive or shared file access permissions
- SFS considerations
- DB2 considerations

EXEC CICS READNEXT and EXEC CICS READPREV with an alternate index: The behavior of the file control commands EXEC CICS READNEXT and EXEC CICS READPREV differs between TXSeries CICS and other CICS family members when dealing with duplicate records in an alternate index. Consider records with the following keys:
1. AAA
2. BBB

3. BBB
4. BBB

If a sequence of EXEC CICS READNEXT commands obtains records 1 to 4, and an application issues an EXEC CICS READPREV command, TXSeries CICS returns record 4, while some other CICS family members return record 2 (this record being the first of the set of duplicates with the specified key.)

If you call EXEC CICS READNEXT and then EXEC CICS READPREV, TXSeries CICS attempts to return the same record, but must read an additional record when it reverses the browse. The extra read call ensures that the DUPKEY condition is correctly raised in indexes that allow records with duplicate keys. In the following example, there are five records and the browse is positioned at record 2:
1. AAA
2. BBB
3. BBB
4. CCC
5. DDD

An EXEC CICS READNEXT, EXEC CICS READPREV sequence is carried out. TXSeries CICS returns record 3 for the EXEC CICS READNEXT. Record 4 is not a duplicate, so it does not set DUPKEY. To carry out the EXEC CICS READPREV call, TXSeries CICS reads record 4 (the NEXT record) and then sets the browse in the new direction. TXSeries CICS reads record 3 (the PREV record) and sets the DUPKEY because record 2 is a duplicate.

Because EXEC CICS READPREV uses two read calls when it reverses the browse, it is possible for another process to simultaneously insert records and cause an unexpected result. For example, a process might insert a record with key BBZ between the two read calls needed for EXEC CICS READPREV. At the end of the EXEC CICS READPREV, the browse would be positioned at the new record, not record 3.

VSAM emulation: TXSeries CICS provides emulation of *virtual sequential access method (VSAM)*. VSAM is an access method for direct or sequential processing of fixed- and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence). or by relative-record number. The VSAM emulation that TXSeries CICS provides is through the use of the SFS, but differs in some areas of detailed implementation:

File access
: If IBM mainframe-based CICS opens a VSAM file, setting the attribute for exclusive access, any CICS transaction can access the file but the file is unavailable outside the CICS region. This limitation is because CICS is responsible for any recovery processing of the file. TXSeries CICS does not need to open SFS files for exclusive access as any recovery processing is performed by SFS.

Record locking
: An application requesting a record for update causes VSAM to lock the complete control interval. The same request using SFS causes only the requested record to be locked.

Updating alternate keys
: Unlike VSAM, SFS allows you to update alternate key values when

rewriting a record accessed by an alternate index. You may still only update the primary key value by deleting the original record and reinserting it.

Unique primary keys
The primary key for SFS key-sequenced files need not be unique.

Deleting file contents
SFS allows you to specify **OpenEmpty** for a file, even if you define alternate indices for the file.

Relative byte addressing
In SFS, the term *relative byte addressing (RBA)* does not indicate the byte offset from the beginning of the file as in VSAM. SFS uses RBA as a value which can be used as a record identifier for any of the three file organizations.

Browse operations
If you perform an EXEC CICS STARTBR command with the key set to its maximum value, then perform an EXEC CICS READPREV command and lastly an EXEC CICS READNEXT command, SFS raises an ENDFILE condition. In CICS/MVS, performing the same commands using VSAM raises an ILLOGIC condition and terminates the application.

SFS considerations: If CICS opens a file, but does not use the file for a configurable period of time, SFS may eventually treat the file as closed, and permit administrative activities to take place on the file. When SFS treats a file as closed and a transaction currently using the file makes a request to SFS, the request fails and CICS abnormally terminates the transaction. The next request may be a request for the same file, a request for a different file, or a syncpoint request.

If a new transaction attempts to access the file that SFS is treating as closed, CICS reopens the file. However, if the structure of the file has changed since CICS last opened the file, CICS generates an error condition.

You may need to amend your application programs, to handle this condition.

For information about SFS or ENCINA for CICS, refer to the appropriate Encina documentation.

1. ***Sharing SFS files between regions***

   It is possible to have an SFS file shared between multiple regions in TXSeries CICS. SFS handles all locking contention between the regions in a manner which is largely transparent to your CICS programs, but you should be aware of some restrictions:

   - Do not function ship a request to a remote region for a resource which it is sharing with your local region in the same logical unit of work (LUW). CICS allows a function-shipped request to read a file for update that has already been read for update locally in the same LUW, whereas, if the accessing is entirely local, CICS raises INVREQ on the second request.
   - Distributed transactions developed on TXSeries CICS that assume the OLTP transaction processing model, which establish two or more conversations to the same remote region and also access the same resources, can deadlock when ported to a CICS/MVS system. Such transactions do not usually deadlock in TXSeries CICS, because they are part of one global transaction. Porting applications the other way, from CICS/MVS to TXSeries CICS, is not affected.

2. ***Effect of ROLLBACK on SFS***

In CICS/MVS, the CICS region manages the LUWs for VSAM operations, and CICS only distributes work when the region either performs some ISC operation with another CICS region, or when a recoverable resource manager, such as a relational database, is used.

TXSeries CICS has a client/server relationship with SFS, with each SFS being a recoverable resource. In consequence all CICS operations that utilize SFS (those concerned with file control, temporary storage queues, transient data queues, and the local queueing of function shipping requests) cooperate with SFS in the management of each LUW. Hence, transactions that are entirely local under CICS/MVS, are effectively distributed under TXSeries CICS.

This difference between the way that CICS/MVS and TXSeries CICS distribute responsibility for an LUW, and its resolution, underlies some differences in behavior between CICS family members.

Consider the following sequence, and the effect of a request by some other transaction to EXEC CICS READ RECORD A, which might occur at any of the 5 indicated points in the sequence. As CICS typically allows locked records to be read, the results at the 5 points are as shown.

```
1  READ RECORD A => NOTFND   WRITE RECORD A
2  READ RECORD A => SUCCESS  SYNCPOINT
3  READ RECORD A => SUCCESS  DELETE RECORD A
4  READ RECORD A => NOTFND   ROLLBACK
5  READ RECORD A => ???????
```

The events at points 1 to 4 are readily understood and consistent between CICS/MVS and TXSeries CICS. At point 5, the distributed nature of the transaction, and the shared responsibility for the LUW result in a difference in behavior.

CICS treats the ROLLBACK command as complete once SFS has been notified of the abnormal termination of the LUW, and has acknowledged the notification. SFS cannot guarantee when it will complete the work associated with rolling back, and further provides no interface to allow the state of the rollback operation to be determined. There is therefore a small interval, of indeterminate length, following the completion of the ROLLBACK request in which an EXEC CICS READ request may return NOTFND. Eventually, when SFS completes the abnormal termination, EXEC CICS READ returns SUCCESS. The true state of affairs is then:

```
1  READ RECORD A => NOTFND   WRITE RECORD A
2  READ RECORD A => SUCCESS  SYNCPOINT
3  READ RECORD A => SUCCESS  DELETE RECORD A
4  READ RECORD A => NOTFND   ROLLBACK
5  READ RECORD A => NOTFND   (SFS starts ROLLBACK)
6  READ RECORD A => NOTFND   (SFS completes ROLLBACK)
7  READ RECORD A => SUCCESS
```

If the read being attempted involves taking a lock, then the reads at points 2, 4, 5 and 6 wait for the transaction resolution, or timeout with an AKCS abnormal termination. Do not view these scenarios as two separate transactions, but a single transaction that performs the following sequence:

```
WRITE RECORD A
SYNCPOINT
DELETE RECORD A
ROLLBACK
READ RECORD A
```

Queue services

If you are migrating application programs that use the API queue services, you need to be aware of the following migration concerns:

Transient data queue trigger levels: When a transient data queue reaches its trigger level for automatic transaction initiation (ATI), and all servers are utilized, TXSeries CICS postpones the ATI until a further transient data queue record is written. Thus, TXSeries CICS does not necessarily initiate a task when the trigger level is reached, but when the trigger level is exceeded. In TXSeries CICS, the ATI takes place as soon as a server becomes available.

Extrapartition transient data queue considerations: Unlike some CICS systems, TXSeries CICS opens extrapartition transient data queues whenever CICS runs a transaction that requires access to an extrapartition transient data queue. Conversely the application server closes the extrapartition transient data queue when CICS finishes running the transaction.

Part 3. Compiling Applications

Table 39. Road map for Compiling applications

If you want to...	Refer to...
Know about source directories, and link libraries when translating, compiling, and link-editing in one or separate steps, and the special requirements of compiling a CICS applications program	Chapter 8, "Translating, compiling, and link-editing CICS application programs," on page 213

Chapter 8. Translating, compiling, and link-editing CICS application programs

After you have written a CICS application, you must first compile and link-edit the application. If the program is written in COBOL, C, or C++, you must first convert the EXEC CICS statements that are included in the programs to COBOL, C or C++ statements. (PL/I programs do not need to be translated prior to compilation.) The two commands you use to do this are:

cicstran
: This is the control language translator that converts source code to an equivalent source program in which each EXEC CICS command has been converted into COBOL, C or C++

cicstcl
: This command performs the translation, compiles the translated program, and links the resulting object.

The remainder of this chapter discusses translating and compiling application systems to run in your CICS region. If you are compiling them to run under the control of a debugging tool, you have to specify additional compiler arguments. See "Using CDCN and the IBM Application Debugging Program (xldb) with CICS for AIX only" on page 258 and "Using a compiler's integrated debugging tool to debug CICS applications" on page 263 for more information.

Note: Whenever you upgrade to a new level of Micro Focus Server Express (on Open Systems) or Micro Focus Net Express (on Windows), always recompile your COBOL programs.

Note: If you are building executable files that use both **.so** libraries and DCE libraries, and if you are using the **-brtl** flag, you must include the **-bnortllib** flag. For example,

```
xlc_r4 -o executable_name object_ files -l dce_library_files  -brtl -bnortllib
```

The combination of the **-brtl** flag and the **-bnortllib** flag is used only to build executable files; it is not used to build libraries.

The PL/I compiler

The IBM PL/I compiler has an integrated preprocessor that should be used instead of **cicstran**. **cicstcl** invokes the PL/I compiler.

Examples (CICS for Windows)

Examples of these procedures are provided by the Installation Verification Programs (IVP) which can be found in opt\cics\samples\IVP. The IVP programs are documented in *Planning and Installation Guide*.

Source directories and link libraries

CICS uses two directory structures, one for regions and one for your installed CICS. As an application programmer, you are interested in the installed CICS system directory structure. More specifically:

Table 40. Source directories and link libraries

$CICS/bin (see note) *prodDir*\bin	Contains those executables that are used by application development programmers. These include, for example, the **cicstran** and **cicstcl** programs. Include this directory in your path list, as well as the path list for all CICS users.
$CICS/lib *prodDir*\lib	Contains those shared libraries that CICS needs to prepare applications written in C, C++, IBM COBOL, and IBM PL/I. This path is included in the compile command that **cicstcl** uses to compile your application programs.
$CICS/include *prodDir*\include	Contains the CICS header files used to prepare applications. These are the COBOL copybooks, C and C++ header files, and PL/I include files. This path is included in the compile command that **cicstcl** uses to compile your application programs.

Translating, compiling, and link-editing in one step

Using **cicstcl** to translate, compile and link-edit CICS programs is the recommended way of generating your executable CICS programs. Consider performing these steps manually only if you have good reason (for example if your CICS program also has to be linked to the runtime of a relational database). See "Translating, compiling and link-editing in separate steps" on page 216 for details on performing these steps separately. The following sections provide the preferred procedure that tells you how to translate, compile, and link-edit your application program using **cicstcl**. The prerequisite tasks must be done before you begin.

Prerequisite Tasks

- Invoke the **cicsmap** command for any map sets that your application program uses. (See "cicsmap - generate BMS map files" on page 276.)
- If you have COBOL copybooks that contain EXEC CICS statements, use **cicstran** to translate them (see "Pre-translating COBOL copybooks" on page 218).

If necessary, set compile and link options. These are not normally required, since **cicstcl** attempts to supply sensible defaults but, if needed, they are supplied as shown in Table 41 and Table 42 on page 215. The values listed are the environment variables from which the **cicstcl** command obtains the flags and directories it uses when compiling and linking programs. For more information on these flags, see the sections in this chapter on the language you are using.

Table 41. Setting compile and link options for cicstcl (CICS on Open Systems).

Compiler	Compile flags	Linker flags	Libraries to link	Include directories
C/C++	CCFLAGS	LDFLAGS	USERLIB	
IBM COBOL	CICS_IBMCOB_ FLAGS	LDFLAGS	USERLIB	SYSLIB

Table 41. Setting compile and link options for cicstcl (CICS on Open Systems) (continued).

Compiler	Compile flags	Linker flags	Libraries to link	Include directories
MF COBOL	COBOPTS			COBCPY
IBM PL/I	CICS_IBMPLI_ FLAGS	LDFLAGS	USERLIB	

Table 42. Setting compile and link options for cicstcl (CICS for Windows).

Compiler	Compile flags	Linker flags	Libraries to link	Include directories
IBM C/C++	CICS_IBMC_ FLAGS	ILINK	USERLIB	
IBM COBOL	CICS_IBMCOB_ FLAGS	USERLIB	USERLIB	
Micro Focus Net Express	COBOPTS			COBCPY
Microsoft® C/C++	CICS_MSC_FLAGS	LINK	USERLIB	
IBM PL/I	CICS_IBMPLI_ FLAGS			

Note: If you have both IBM VisualAge C++ and IBM VisualAge® COBOL installed, the COBOL directories must be in the PATH system environment variable before the C++ directories.

Procedure

Invoke the **cicstcl** command to translate, compile, and link edit your application program. If the translation is successful, **cicstcl** compiles the program, and if the compilation is successful, link-edits it. (See "cicstcl - translate, compile, and link" on page 281.)

CICS writes errors, warnings and informational messages to the operating system file stream, stderr.

Note: When compiling with **cicstcl**, you can use the **-a** option to retain the intermediate file. On CICS for Windows, this is required if you are using the program debugging tools.

Caching transaction programs and NEWCOPY (CICS for Windows)

Caching a program improves performance because reloading costs are saved when the program is used. The following options affect program caching:

- The Region Definitions (RD) **ProgramCacheSize** attribute.

  Programs in C, C++, IBM COBOL, or PL/I are cached only if at the time a program is loaded, the Program Definitions (PD) **Resident** attribute is set to **yes** and the number of cached programs in use has not reached the maximum number.

  Because individual programs in use are not removed from the cache, it is recommended that the cache size allow programs at every logical level to be cached when EXEC CICS LINK is used.

- The environment variable COBSW is set to l*nn*

  where *nn* is the cache size in bytes. This is a Micro Focus Server Express and Net Express environment variable that sets the length of the Server Express and Net Express program cache. This variable is defined in the region's environment file.

Java™ programs can be cached by the Java runtime controlled through the Java Virtual Machine (JVM).

Programs are loaded into the cache in the order they are accessed.

Note:

- Programs are cached for each application server, so a new copy of the program has no effect on an application server if the application server has not yet run the program.
- The SET PROGRAM NEWCOPY or SET PROGRAM COPY(NEWCOPY) commands for a Micro Focus Server Express or Net Express program remove every program previously loaded by the application server, so a fresh copy of every such program is used after one of these commands is run, not just the program for which the SET PROGRAM NEWCOPY or SET PROGRAM COPY(NEWCOPY) command is issued. These commands have no effect on Java programs.
- The SET PROGRAM NEWCOPY and SET PROGRAM COPY(NEWCOPY) commands complete successfully but have no effect on C, C++, IBM COBOL, or PL/I cached programs because, on Windows systems, a file on disk cannot be removed or overwritten while a copy of it exists in the cache. Therefore, new copies of these programs can be used only if they are not cached.

Translating, compiling and link-editing in separate steps

Using **cicstcl** to translate, compile and link-edit CICS programs is the recommended way of generating your executable CICS programs. Consider performing these steps manually only if you have good reason (for example if your CICS program also has to be linked to the runtime of a relational database). See "Translating, compiling, and link-editing in one step" on page 214 for details on using **cicstcl**.

The CICS translator, **cicstran**, is used to translate EXEC CICS commands into COBOL, C or C++ statements (there is no translate step for PL/I programs). Table 43 shows the file names required by the **cicstran** command and the names of the output files produced by that command.

Table 43. File names used by the cicstran command

Language	Input File Name	Output File Name
COBOL	*file*.ccp	*file*.cbl
C	*file*.ccs	*file*.c
C++	*file*.ccs	*file*.C

If your program includes map sets, then you will also use the **cicsmap** command. This command takes source files containing Basic Mapping Support (BMS) macros and generates symbolic and physical maps. They must be generated before translation.

How translation works

For a COBOL application program, each command is replaced by one or more MOVE and SET statements, followed by a COBOL CALL statement, followed by an IF statement.

The purpose of the MOVE and SET statements is to encode the information in the command into the CICS internal format.

For example, an IBM COBOL command such as:

```
EXEC CICS RECEIVE MAP('A') END-EXEC.
```

can be translated to:

```
MOVE 'A' TO CICS-STRING-VALUE(6)
SET CICS-DATA-AREA(9) TO ADDRESS OF AI
MOVE H'00000120' TO CICS-ARG-MASK
MOVE 0 TO CICS-ARG-COUNT
MOVE 68 TO CICS-FN-CODE
MOVE -1 TO CICS-DEBUG-LINE
CALL "CICSAPI" USING CICS-ARGS
IF EIBLABEL NOT=0
GO TO CICS-API-ERROR
END-IF
```

A similar Micro Focus Server Express COBOL command gives the same translation:

```
EXEC CICS RECEIVE MAP('A') END-EXEC.
```

can be translated to:

```
MOVE 'A' TO CICS-STRING-VALUE(6)
SET CICS-DATA-AREA(9) TO ADDRESS OF AI
MOVE H'00000120' TO CICS-ARG-MASK
MOVE 0 TO CICS-ARG-COUNT
MOVE 68 TO CICS-FN-CODE
MOVE -1 TO CICS-DEBUG-LINE
CALL "CICSAPI" USING CICS-ARGS
IF EIBLABEL NOT=0
GO TO CICS-API-ERROR
END-IF
```

Note: CALL "cics_api_exec_cobol" USING CICS-ARGS is kept for compatibility.

The translator modifies the linkage section by inserting the EXEC interface block (EIB) structure as the first parameter, and inserts declarations of the temporary variables that it requires into the working-storage section. It also inserts a DFHCOMMAREA if there is not one in the linkage section.

For a C application program, each command is replaced by one or more assignment statements and function calls. For example, a command such as:

```
EXEC CICS RECEIVE MAP("A");
```

can be translated to:

```
{
 cics_api_strncpy(CicsArgs.ArgData[5].StringValue, "A", (short)7);
 CicsArgs.ArgData[8]DataArea = &a.ai;
 CicsArgs.ArgMask = 0x20000120;
 CicsArgs.FnCode = 68;
 CicsArgs.DebugLine = -1;
 cics_api_exec_c(&CicsArgs);
}
```

Pre-translating COBOL copybooks

If your source contains COBOL copybooks with EXEC CICS statements in them, then, before you translate the main program, you must translate:

- All copybooks that contain EXEC CICS statements
- All copybooks that include copybooks with EXEC CICS statements in them

You can translate the copybooks in any order as long as the main program is translated after all of the copybooks have been translated.

Note: When **cicstcl** is used, all associated **.cbl** files are erased. However, copybooks obtained as a result of using **cicstran** are not erased. This allows you to save pre-translated copybooks for future compiles.

It is advisable to use the **.cbl** extension with the COPYBOOK file name. This allows additional messages associated with translation.

For example, to include a copybook called REPORT:

```
COPY 'REPORT.cbl'.
```

The CICS translator only recognizes copybooks included in the main program by the COBOL copy statement.

Note: Micro Focus Server Express and Net Express default to uppercase so the single quotes are essential.

The translation procedure

This procedure tells you how to translate your application program using **cicstran**. The prerequisite task must be done before you begin.

Note: For an alternate translation procedure, see "Translating, compiling, and link-editing in one step" on page 214.

Prerequisite task

Invoke the **cicsmap** command for any map sets that your application program uses. (See "cicsmap - generate BMS map files" on page 276.)

Procedure

1. If you have COBOL copybooks that contain EXEC CICS statements, use **cicstran** to translate them (see "Pre-translating COBOL copybooks").
2. Use **cicstran** to translate your COBOL, C, or C++ program. Your program is now ready to be compiled and link edited. Refer to "Requirements for compiling CICS application programs."

CICS writes errors, warnings and informational messages to stderr.

Note: The advantage of using **cicstcl** to translate your program is that it uses the proper compile parameters required by CICS.

Requirements for compiling CICS application programs

This section describes the minimum requirements for compiling CICS application programs.

The following table indicates the languages and compilers supported by TXSeries CICS:

Table 44. Programming Language support

On CICS for:	Native C	Native C++	Micro Focus	IBM COBOL	IBM PL/I
AIX	Yes	Yes	Yes Server Express	Yes	Yes
HP-UX	Yes	No	Server Express	No	No
Solaris	Yes	Yes	Yes Server Express	No	No
Windows	Yes	Yes	Yes Net Express	Yes	Yes

Compiling and linking a C application program (CICS on Open Systems)

Following are the minimum requirements for compiling a C application program.

Your program might require other include files, in other include directories, and other libraries in other library directories. These other files and libraries might not be CICS.

For example, to compile and link fred.c, enter:

On CICS for AIX

```
xlc_r4 -I $CICS/include -bI:$CICS/lib/cicsprC.exp -e main
       -o fred fred.c
```

On CICS for HP-UX

```
cc -Aa +z -c -I $CICS/include fred.c
ld -b +e main -L $CICS/lib -o fred fred.o
```

The **+z** option is required to generate position-independent code. The **-Aa** option is required to put the compiler in ANSI mode.

On CICS for Solaris

```
cc -misalign -I $CICS/include -KPIC -Xa -c fred.c
cc -misalign -L $CICS/lib -G -e main -o fred fred.o
```

Compiling and linking a C application program (CICS for Windows)

Following are the minimum requirements for compiling a C application program.

Your program might require other include files, in other include directories, and other libraries in other library directories. These other files and libraries might not be CICS.

Requirement	IBM C	MICROSOFT C
Compiler	icc	cl

Requirement	IBM C	MICROSOFT C
Link with CICS library	*prodDir*\lib\cicsprC.lib	*prodDir*\lib\cicsprC.lib
Include path	*prodDir*\include	*prodDir*\include

For example, to translate, compile and link myprog, enter:

With IBM VisualAge C

```
cicstran -lIBMC myprog.ccs  -------> myprog.c
icc -Mt -IprodDir\include -Ge- -Gd+ -Gm+ myprog.c ibmc.def
         prodDir\lib\cicsprC.lib ---> myprog.dll
```

Note: It is necessary to export **_main** for CICS transaction programs built with the IBM compiler. This can be achieved with a definition file with the following two lines:

```
EXPORTS
_main
```

With Microsoft Visual C

```
cicstran -lC myprog.ccs -------> myprog.c
cl -IprodDir\include -Gz -LD myprog.c mfc.def prodDir\lib\cicsprC.lib
                        -------> myprog.dll
```

Note: The entry point **main** must be exported and this can be done by declaring it as:

```
_declspec(dllexport) void main ....
```

or linking with a definition file with the following two lines

```
EXPORTS
  main
```

Compiling and linking a C++ program (CICS on Open Systems)

A CICS C++ application program must have an extension of **.ccs**, and can be processed using **cicstcl** or **cicstran**, just as a C program but there the similarity ends. The intermediate source file of a C++ program has an extension of **.C** and the executable extension of **.ibmcpp** (on AIX) or **.cpp** (on Solaris and HP-UX).

Using cicstcl for a C++ program

On AIX, specify the C++ language by using the language parameter **IBMCPP**:

On CICS for AIX

```
cicstcl -l IBMCPP fred.ccs
```

This command produces the executable **fred.ibmcpp**.

On Solaris and HP-UX, specify the C++ language by using the language parameter **CPP**:

On CICS for Solaris and HP-UX

```
cicstcl -l CPP fred.ccs
```

This command produces the executable **fred.cpp**.

The **cicstcl** options that are valid for C applications are also valid for C++ applications.

Using cicstran for a C++ program

Just as for **cicstcl**, specify the C++ language parameter **IBMCPP** (on AIX) or **CPP** (on Solaris and HP-UX), and then invoke the compiler:

On CICS for AIX

```
cicstran -l IBMCPP fred.ccs  (to produce fred.C)
xlC_r4 -I $CICS/include -c -o fred.o fred.C
makeC++SharedLib -o fred.ibmcpp -p0 -nmain
  -bI:$CICS/lib/cicsprCpp.exp -lC_r fred.o
```

On CICS for Solaris

```
cicstran -l CPP fred.ccs  (to produce fred.C)
CC -G -misalign -KPIC -I$CICS/include fred.C -o fred.cpp -L $CICS/lib
```

On CICS for HP-UX

```
cicstran -l CPP  fred.ccs  (to produce fred.C)
aCC -c +DAportable -b +z -z -I$CICS/include fred.C -o fred.cpp
```

Use **makeC++SharedLib** on AIX to link your applications to ensure the correct initialization of objects, and link with the CICS library **$(CICS)/lib/cicsprCpp.exp**.

Note: When using the **makeC++SharedLib** command ensure that /usr/lpp/xlC/bin is included in the PATH.

Building a class library

To build a library **libfred.a** from **fred1.ccs** and **fred2.ccs** the process is:

On CICS for AIX:

1. Translate fred1 and fred2

   ```
   cicstran -l IBMCPP fred1
   cicstran -l IBMCPP fred2
   ```

2. Compile the files produced

   ```
   xlC_r4 -c -I $CICS/include -o fred1 fred1.C
   xlC_r4 -c -I $CICS/include -o fred2 fred2.C
   ```

3. Link and archive

   ```
   makeC++SharedLib -o fredshr.o -p 0
   -bI:$CICS/lib/cicsprCpp.exp fred1.0 fred2.0
   ar rv libfred.a fredshr.o
   ```

On CICS for Solaris:

1. Translate fred1 and fred2

```
cicstran -l CPP fred1
cicstran -l CPP fred2
```

2. Compile the files produced

```
CC -c -misalign -KPIC -I$CICS/include fred1.C -oi fred1.o
CC -c -misalign -KPIC -I$CICS/include fred2.C -oi fred2.o
```

3. Link the object files into the shared library

```
CC -G -o libfred.so fred1.o fred2.o
```

To use the library, set the LDFLAGS environment variable to point to the library. For example:

```
LDFLAGS=-L/libpath -l fred
```

On Solaris, you must also set the LD_LIBRARY_PATH environment variable to include this library. Then translate, compile, and link your program as normal.

On Solaris, when using templates, ensure the compiler -ptr option is set to point to the template database. For example, if you are compiling a file that contains a template in a directory called **/path/apps**, the compiler automatically creates a directory called **Templates.DB** in the **/path/app** directory. If you are using the **cicstcl** command to prepare applications that use those templates, set the compiler option using the CCFLAGS environment variable.

On CICS for HP-UX:

1. Translate fred1 and fred2

```
cicstran -l CPP fred1
cicstran -l CPP fred2
```

2. Compile the files produced

```
aCC -c +DAportable +z -I$CICS/include fred1.C -o fred1.o
aCC -c +DAportable +z -I$CICS/include fred2.C -o fred2.o
```

3. Link the object files into the shared library

```
aCC -b -Wl, +s +DAportable +z -z -o libfred.sl fred1.o fred2.o
```

On HP-UX, you must set the LDFLAGS environment variable to include this library. For example, using *libpath* to represent the path to the library, set the LDFLAGS environment variables as follows:

```
LDFLAGS="-Llibpath -lfred"
```

On HP-UX, you must also set the SHLIB_PATH environment variable to include this library. Then translate, compile, and link your program as normal.

Compiling and linking a C++ program (CICS for Windows)

Using cicstran for a C++ program

Following are the minimum requirements for compiling a C++ application program.

Your program might require other include files, in other include directories, and other libraries in other library directories. These other files and libraries might not be CICS.

Requirement	IBM C++	MICROSOFT C++
Compiler	icc	cl
Link with CICS library	*prodDir*\lib\cicsprCpp.lib	*prodDir*\lib\cicsprCpp.lib
Include path	*prodDir*\include	*prodDir*\include

For example, to translate ,compile, and link myprog, enter:

With IBM VisualAge C++

```
cicstran -lIBMCPP myprog.ccs ----> myprog.C
icc -Mt -IprodDir\include -Ge- -Fe myprog.ibmcpp -Gd+ -Gm+ -Tp myprog.C
             prodDir\cicsprCpp.lib ibmc.def ----> myprog.ibmcpp
```

Note: It is necessary to export **_main** for CICS transaction programs built with the IBM compiler. This can be achieved with a definition file with the following two lines:

```
EXPORTS
_main
```

With Microsoft Visual C++

```
cicstran -lCPP myprog.ccs -------> myprog.C
cl -IprodDir\include -Gz -LD -GX -Femyprog.cpp -Tp myprog.C mfc.def
    prodDir\lib\cicsprCpp.lib ----> myprog.cpp
```

Note: The entry point **main** must be exported and this can be done by declaring it as:

```
_declspec(dllexport) void main ....
```

or linking with a definition file with the following two lines

```
EXPORTS
  main
```

Building a class library

This example comprises 2 classes ts and td queue in separate source files and the class is called queue.dll. The process is:

For IBM VisualAge C++

1. translate

```
cicstran -lIBMCPP tsqueue.ccs ----------> tsqueue.C
cicstran -lIBMCPP tdqueue.ccs ----------> tdqueue.C
```

2. compile the files produced

```
icc -c -Mt -IprodDir\include -Ge- -Gd+ -Gm+ -Tp tsqueue.C
            ---> tsqueue.obj
icc -c -Mt -IprodDir\include -Ge- -Gd+ -Gm+ -Tp tdqueue.C
            ---> tdqueue.obj
```

3. link and archive

```
ilib /gi:queue tsqueue.obj tdqueue.obj
            ---> queue.lib, queue.exp
ilink /DLL /O:queue.dll prodDir\lib\cicsprCpp.lib
            tsqueue.obj tdqueue.obj queue.exp  ----> queue.dll
```

It is necessary to export _main for CICS transaction programs built with the IBM compiler, which can be done in a definition file containing the following two lines:

```
EXPORTS
_main
```

Do not use the LIBRARY keyword in the definition, because this causes problems when the program is linked.

If it does not already exist, the definition file is created automatically by **cicstcl**.

These compiler options are described briefly below. Refer to the compiler documentation for further information. Some options, for example, **-Tp**, are positionally dependent and must come immediately before the source file.

-Mt Set default linkage to __stdcall.

-Ge- Use runtime that assumes a DLL is being built.

-Gd+ Use DLL runtime.

-Gm+ Use multithread runtime.

-Fe Specify program name.

-Tp Compile source as C++ file.

gi ilib option to create import library/export object pair.

It can also be necessary to use the **-Ft-** option if you are using class templates.

Note: If you wish to change the default linkage to **_cdecl**, use the **-Mc** option instead of **-Mt**.

For Microsoft Visual C++

1. translate

```
cicstran -lCPP tsqueue.ccs ----------> tsqueue.C
cicstran -lCPP tdqueue.ccs ----------> tdqueue.C
```

2. compile the files produced

```
cl -c -IprodDir\include -GX -Gz -Tp tsqueue.C  -----> tsqueue.obj
cl -c -IprodDir\include -GX -Gz -Tp tdqueue.C  -----> tdqueue.obj
```

3. link and archive

```
link /dll /out:queue.dll $prodDir\lib\cicsprCpp.lib tsqueue.obj
           tdqueue.obj -----> queue.dll
```

The compiler options used were:

-Gz Set default linkage to **__stdcall**.

-LD Build DLL.

-GX Enable C++ exception handling.

-Tp Compile source as C++ file.

Note: If you wish to change the default linkage to **_cdecl**, use the **-Gd** option instead of **-Gz**.

Compiling a Micro Focus Server Express COBOL application program (CICS on Open Systems)

Following are the minimum requirements for compiling a Micro Focus Server Express COBOL application program. Your program possibly requires other copybooks and copybook directories, which are possibly not CICS.

On CICS for AIX

Use the /bin/cob compiler.
The PATH environment variable must include /bin.
Then invoke the COBOL compiler with the **cob** command.

On CICS for HP-UX

Use the /opt/cobol/bin/cob compiler. The PATH environment variable must include /opt/cobol/bin. Then invoke the COBOL compiler with the **cob** command.

On CICS for Solaris

Use the /usr/bin/cob compiler. The PATH environment variable must include /usr/bin. Then invoke the COBOL compiler with the **cob** command.

For Micro Focus Server Express COBOL, use the compiler COBCPY environment variable for the copybook path. The copybook path is:

```
$CICS/include
```

Copybook files are CICS-API and CICS-EIB

Compile with the **-C ANIM** and **-C INITCALL=CBL_DEBUGBREAK** options to use Animator, or the **-u** option to compile without Animator. If you use Animator, the command produces the following two files:

```
filename.int
filename.idy
```

If you compile without Animator, the command produces the following file:

```
filename.gnt
```

For example, to compile and link gred.cbl using Animator, enter:

```
COBCPY="$CICS/include"
cob -C ANIM -C INITCALL=CBL_DEBUGBREAK fred.cbl
```

To compile mprog.cbl without Animator, enter:

```
COBCPY="$CICS/include"
cob -u fred.cbl
```

Note: Do not use the file name extensions for the **PathName**. For example, if the program fred.int is installed at:

```
/usr/mydirectory/fred.int
```

The PD **PathName** is:

```
/usr/mydirectory/fred
```

If you have compiled the file with Animator, you must install the **.idy** file and the **.int** file as described in "Using the debugging tool integrated with Micro Focus Server Express COBOL on CICS on Open Systems (Animator)" on page 269.

Install the **.gnt** file according to the **PathName** attribute in the Program Definitions (PD) for the required transaction.

Note: To avoid losing leading digits, use NOTRUNC if values greater than 9999 are assigned to COMP items.

Compiling a Micro Focus Net Express COBOL application program (CICS for Windows)

CICS for Windows supports Micro Focus Net Express COBOL. For Micro Focus Net Express, CICS supports files with the extensions **.cbmfnt**, **.gnt**, and **.int**. Files with the extension **.int** are interpreted files and do not need to be link edited.

Note: It is necessary to install Microsoft Visual C++ for use with Micro Focus Net Express COBOL because Net Express requires access to the **link.exe** file that accompanies Microsoft Visual C++. You must copy the **link.exe** file from Microsoft Visual C++ and place it in the **\NetExpress\Base\BIN** directory.

Warning messages issued on the CODE and DATA sections when using the Microsoft Visual C ++ version 6.0 can be safely ignored. If Run-Time System Websync Update version 1.0.006 (Micro Focus patch FixPack) or later is installed, these warning messages no longer occur.

To translate, compile, and link a COBOL program, you can either use the **cicstcl** command or perform the processes as three separate steps.

- The following example shows how to use the **cicstcl** command to compile the file **myprog.ccp**. (This command can be used to produce only **.cbmfnt** files.)

  ```
  cicstcl myprog.ccp
  ```

  This example produces the file **myprog.cbmfnt**.
- The following example shows how to translate, compile, and link the **myprog.ccp** file in three separate steps:

  ```
  cicstran myprog.ccp
  cobol myprog.cbl;
  cbllink -D -Mmyprog -Omyprog.cbmfnt myprog.obj
                C:\opt\cics\lib\cicsprCBMFNT.lib
  ```

  In this example, the **cicstran** command uses the **.ccp** file to produce the **.cbl** file, which is used by the **cobol** command. The **cobol** command produces an **.obj** file, which is used by the **cbllink** command. The **cbllink** command produces the **.cbmfnt** file. The command set is best run from within a Micro Focus Net Express command window because the environment is already set up. The semicolon (;) in the compile step is significant because it makes execution faster by stopping the compiler from prompting you.

  If you are using Micro Focus Net Express and want to produce an **.int** file, you do not need to run the **cbllink** command. However, you must specify the */NOGNT* option to the **cobol** command.

  You must either specify `myprog` in the PROGRAM ID paragraph or use the **-M** option to ensure that the program has an entry point matching the source name.

  The CICS COPYBOOK files are CICS-API and CICS-EIB.

Using Micro Focus Net Express to compile EBCDIC-enabled COBOL programs

EBCDIC-to-ASCII data conversion can be necessary for communications between a CICS for Windows system and a remote mainframe or CICS/400® system. For example, if you use function shipping to access file records from a mainframe, the data must be converted from EBCDIC to ASCII in order to be usable by the ASCII program on the CICS for Windows workstation. Often, resource definition templates must be defined to identify the type of conversion to be applied to the data.

To avoid the need to set up these conversion tables or to ensure the collating sequence compatibility of mainframe applications, you can use the **cicscobinsert** utility to compile EBCDIC-enabled programs to run on a CICS for Windows workstation. Such EBCDIC-enabled programs are supported by Micro Focus Net Express version 3.0 or later.

Note: Use in EBCDIC-enabled programs of the Front-End Programming Interface (FEPI), Basic Mapping Support (BMS) macros, and pretranslated copybooks is not supported. Support of EXEC SQL calls is provided through Micro Focus Net Express.

To create an EBCDIC-enabled program, first write the source code on the workstation in ASCII or download the source code from the mainframe, using EBCDIC to ASCII conversion in the usual way. Then set the Micro Focus Net Express option CHARSET to EBCDIC, as shown in the following example:

```
set COBOPTS=/CHARSET(EBCDIC)
```

Then translate the program on the workstation. You can specify the **cicscobinsert** utility as a value for the **-X** parameter of the **cicstcl** command or use it as a separate command step after running the **cicstran** command, but before performing the actual compilation.

- The following example uses the **cicstcl** command to compile a program called **myprog.ccp** as EBCDIC-enabled:

  ```
  cicstcl -X cicscobinsert myprog.ccp
  ```

- The following example uses a separate command step after the **cicstran** command to compile a program called **myprog.ccp** as EBCDIC-enabled:

  ```
  cicstran -qAPOST myprog.ccp
  cicscobinsert myprog.cbl
  cobol /DATA-CONTEXT /CALL-RECOVERY /CHARSET(EBCDIC) myprog.cbl;
  cbllink -D -Mmyprog -Omyprog.cbmfnt myprog.obj
                C:\opt\cics\lib\cicsprCBMFNT.lib
  ```

  In this example, the **cicstran** command uses the **.ccp** file to produce the **.cbl** file, which is used by the **cicscobinsert** and **cobol** commands. The **cobol** command produces the **.obj** file, which is used by the **cbllink** command. The **cbllink** command produces the **.cbmfnt** file. The command set is best run from within a Micro Focus Net Express command window because the environment is already set up. The semicolon (;) in the compile step is significant because it makes execution faster by stopping the compiler from prompting you.

  You must either specify `myprog` in the PROGRAM ID paragraph or use the **-M** option to ensure that the program has an entry point matching the source name.

See "Compiling EBCDIC-enabled COBOL programs" on page 57 and "cicstcl - translate, compile, and link" on page 281 for related information.

Compiling an IBM COBOL application program (CICS for Windows)

To translate, compile, and link myprog, enter:

For IBM COBOL

1. translate

   ```
   cicstran -lIBMCOB myprog.ccp ----------> myprog.cbl
   ```

2. compile the files produced

   ```
   cob2 -c -qlib -qthread -I$prodDir\include myprog.cbl
            -------> myprog.obj
   ```

3. link and archive

   ```
   ilib /nol /gendef /gi:myprog myprog.obj %COBOLMAIN%\lib
            \iwzrwin4.obj
            -------> myprog.def, myprog.exp, myprog.lib

        edit the myprog.def file to insert at the top of the file:
            LIBRARY myprog.ibmcob

        place @1 after the exported symbol _MYPROG@8
            ------> MYPROG@8 @1

   ilib /nol /def:myprog.def /genimplib
            -------> myprog.ibmcob.exp, myprog.ibmcob.lib

   ilink /free /nol /dll /o:myprog.ibmcob myprog.ibmcob.exp
         myprog.obj iwzrwin3.obj iwzrwin4.obj
         prodDir\lib\cicsprIBMCOB.lib -------> myprog.ibmcob
   ```

Note: %COBOLMAIN% is an environment variable that is set by the IBM COBOL compiler during installation.

If you are compiling a program that calls a subprogram, you must compiler the subprogram, then compile the main program. The following example shows the compilation of **mysubprog**:

```
cob2 mysubprog -dll:mysubprog
```

If the main program is going to dynamically link to the subprogram, you can also specify the *-qthread* option. This specifies that the subprogram gets a separate, initialized working storage area for each program that invokes it. The working storage exists for the duration of the main program.

If the subprogram is to be statically linked to the main program, the main program should be translated, compiled, and linked in three separate steps. The steps are similar to those shown above, but the names of both the main program and the subprogram should be used in the **ilib** and **ilink** statements.

If the subprogram is to be dynamically linked to the main program, you can use the **cicstcl** program. Alternatively, you can translate, compile, and link in separate steps, but specify that the linking is dynamic using the *-qDYNAM* option to the compiler.

For more information on statically and dynamically linking subprograms, see the IBM Visual Age COBOL programming guide.

Compiling an IBM COBOL application program (CICS on Open Systems)

Following are the minimum requirements for compiling an IBM COBOL application program. Your program possibly requires other copybooks and copybook directories, which possibly are not CICS.

On CICS for AIX

```
cob2_r -qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
       -e _iwz_cobol_main -L/usr/lib/dce            \
       -ldcelibc_r -ldcepthreads -qAPOST            \
       -I/usr/lpp/cics/include -ofred.ibmcob fred.cbl

The cob2_r command generates a file fred.ibmcob
```

Use the compiler SYSLIB environment variable or the -l flag for the copybook path. The copybook path is:

```
$CICS/include
```

Copybook files are CICS-API and CICS-EIB

Compiling a PL/I application program

Following are the minimum requirements for compiling a PL/I application program. Your program possibly requires other copybooks and copybook directories, which possibly are not CICS.

On CICS for AIX

```
Compile by
pli -c -I/usr/lpp/cics/include -ofred.ibmpli fred.pli
and link by
ld -bI:/usr/lpp/cics/lib/cicsprIBMPLI.exp   -e plicics
   -l plishr_r
   -l c_r
   -ofred.ibmpli
    fred.o
```

INCLUDE files are **cics_api.inc** and **cics_eib.inc**.

Note: The entry point for the program is placed as shown above. However, if **fred** is a dynamically called subprogram resulting from PL/I fetch, the entry point is the same name as the subprogram, that is **fred** in the example.

Part 4. Debugging Applications

Table 45. Road map for Debugging applications

If you want to...	Refer to...
Code for error handling, debugging, and performance monitoring services.	Chapter 9, "Coding for problem determination," on page 233
Look up how to test your application and how to use Animator.	Chapter 10, "Testing and debugging your application," on page 253

Chapter 9. Coding for problem determination

This chapter describes how to write application programs that use the CICS problem determination services.

Error-handling services

Every time you process a CICS command in one of your applications, CICS automatically raises a condition to tell you how the command worked. This condition (which is usually NORMAL) is passed back by CICS to your application. If something out of the ordinary happens, you will get an exception condition, which simply means a condition other than NORMAL. By testing this condition, you can tell what has happened, and possibly why. Not all conditions mean that there is an error, even if they are not NORMAL. (For example, if you get an ENDFILE condition on an EXEC CICS READNEXT during a file browse, it might be exactly what you expect.) For further information about all possible conditions and the commands for which they can occur, see the *CICS Application Programming Reference*.

Handling error conditions

When a condition is raised you have three different ways of doing something about it, and a fourth way that simply mixes these to give you more flexibility. You can:

Method one

Let the program continue, with control coming straight back from CICS to your program. You can then find out what happened by testing (for example) the RESP value that CICS returns after executing a command. The result of this test enables you to decide what to do next. For details, see "Letting the program continue" on page 234.

Method two

Pass control to a specified label if a named condition arises. You do this by using an EXEC CICS HANDLE CONDITION command to name both the condition and the label of a routine in your code to deal with it. For details, see "Passing control to a specified label" on page 237. You can only do this in COBOL programs.

Method three

Do nothing and rely on the CICS system default action. This is a perfectly sensible option in some cases, and means you do nothing by way of testing or handling conditions. The CICS default action is normally (but not always) to abend the task.

Note: For the conditions ENQBUSY, NOJBUFSP, NOSPACE, NOSTG, QBUSY, and SYSBUSY, the default is for CICS to force the task to wait until the required resource (for example, storage) becomes available, and then resume processing the command. (There is an exception to this; see the descriptions of the EXEC CICS WRITEQ TS and EXEC CICS WRITEQ TD commands in the *CICS Application Programming Reference* for the different default actions for NOSPACE.) For the conditions SIGNAL and EXPIRED, the default

action is to ignore the condition and carry on processing. For details, see "Relying on the system default action" on page 239.

Method four

Mix the methods in any way you choose. For details, see "Mixing methods" on page 242.

There are just over 80 conditions, each with a name (such as LENGERR, for length error) and a matching number. Apart from these 80 or so conditions, there's a general condition named ERROR that you can use as a catchall for any condition. The default action is to terminate the task abnormally. There is also the NOTAUTH condition, which is a general condition that is raised when a resource security check on a command has failed.

In short, you have a potentially large number of CICS conditions to contend with, and three distinct ways of dealing with them. Unless your installation has very definite standards to the contrary, it is recommended to use the first method to handle any errors.

C and C++ restrictions for error handling

If you are using C or C++ you cannot use:

- EXEC CICS HANDLE CONDITION
- EXEC CICS HANDLE AID
- EXEC CICS IGNORE CONDITION
- EXEC CICS PUSH HANDLE
- EXEC CICS POP HANDLE

You can use the EXEC CICS HANDLE ABEND PROGRAM command. In a C++ or C application, every EXEC CICS command is treated as if it had the NOHANDLE option specified. This means that the default system-action transaction abends that result from a condition occurring, but not being handled, is not possible in a C++ or C application. Control always flows to the next instruction, and it is the responsibility of the application to test for a normal response.

Letting the program continue

Letting the program continue means allowing control to return from CICS to the next instruction in your program following the EXEC CICS command that has just executed. You have three ways of doing this. You can:

- Put the RESP option on the command
- Put the NOHANDLE option on the command
- Use an EXEC CICS IGNORE CONDITION command

Just letting the program continue is of little use. However, at the same time, CICS sets return codes in the EXEC interface block (EIB). You can test for particular conditions right after each CICS command, executing the command and then immediately checking whether it did what you wanted by testing the RESP value.

CICS makes it very easy to test the RESP value by using a built-in function called DFHRESP. With this, your code can examine RESP values symbolically. This is a lot easier than looking at hexadecimal values that are both less meaningful to someone reading the code, and awkward to manage in COBOL, C, C++ and PL/I.

How to use the RESP option

Simply code the RESP option on your CICS command and follow it up immediately with tests on the returned RESP value.

To execute and test an EXEC CICS RECEIVE MAP command, you code:

```
EXEC CICS RECEIVE MAP (your_mapname)
   ..MAPSET(your_mapset_name)
   ..RESP (your_response_variable_name)
```

For example, consider the following section of COBOL:

```
*    GET INPUT AND CHECK REQUEST TYPE FURTHER.
     EXEC CICS RECEIVE MAP('ACCTMNU')
            MAPSET('ACCTSET') RESP(RESPONSE) END-EXEC.
     IF RESPONSE = DFHRESP(MAPFAIL) GO TO NO-MAP.
     IF RESPONSE NOT = DFHRESP(NORMAL) GO TO OTHER-ERRORS.
     IF REQML > 0 MOVE REQMI TO REQC.
     .
     .
     .
```

In this example, the first thing to do after the EXEC CICS RECEIVE MAP command is to test the value CICS puts into RESPONSE to check whether it worked. You start by looking explicitly for condition MAPFAIL because it can occur without there being any serious error (if, for example, the user presses CLEAR at this point in the application) and you have to be able to recover:

```
IF RESPONSE = DFHRESP(MAPFAIL) GO TO NO-MAP
```

You now branch to the paragraph at label NO-MAP.

MAPFAIL is by no means the only condition that can arise on an EXEC CICS RECEIVE MAP command.

As with all commands, you have a choice. You can either test explicitly for all possible conditions after each command, or test for some subset of those conditions and somehow deal with all other possibilities elsewhere in your program. (Usually, this means taking the system default action if all else fails. However, in this case, because you are using the RESP option, you must make sure that you allow for all possible conditions somewhere in your code. This is because there is not a system default available in this case, because CICS returns straight to the application program. See “How CICS keeps track of what to do” on page 242.)

Here, the decision is that any value other than normal is to be dealt with in the paragraph at label OTHER-ERRORS:

```
IF RESPONSE NOT = DFHRESP(NORMAL) GO TO OTHER-ERRORS.
```

The code at paragraph OTHER-ERRORS is to catch all other conditions, in one way or another, as shown in the following example:

```
*    PROCESSING FOR UNEXPECTED ERRORS.
     OTHER-ERRORS.
          MOVE EIBFN TO ERR-FN, MOVE EIBRCODE TO ERR-RCODE.
          MOVE EIBFN TO ERR-COMMAND, MOVE EIBRESP TO ERR-RESP.
          EXEC CICS HANDLE CONDITION ERROR END-EXEC.
          EXEC CICS LINK PROGRAM('ACCT04')
              COMMAREA(COMMAREA-FOR-ACCT04) LENGTH(14) END-EXEC.
          GOBACK.
```

ACCT01 picks up what information it can about what has happened, and then links to the error-handling program ACCT04, which issues a user abend and displays a final error message to the user.

Finally, with all condition testing out of the way, you can resume normal processing:

```
IF REQML > 0 MOVE REQMI TO REQC.
```

RESP and RESP2 options: You can use the RESP option with any command, and RESP2 with INQUIRE and SET, to test whether an exception condition was raised during its processing.

RESP(xxx)

where *xxx* is a user-defined 32-bit binary data area. On return from the command, it contains a value corresponding to the condition that may have been raised, or to a normal return, that is, xxx=DFHRESP(NORMAL). You can test this value using DFHRESP, as follows:

```
EXEC CICS WRITEQ TS FROM(abc)
                    QUEUE(qname)
                    RESP(xxx) END-EXEC.
   .
   .
IF xxx=DFHRESP(NOSPACE) THEN
```

The above form of DFHRESP applies to COBOL.

For C or C++ the test is:

```
EXEC CICS WRITEQ TS FROM(abc)
                    QUEUE(qname)
                    RESP(xxx);
   .
   .
if (xxx == DFHRESP(NOSPACE))
```

For PL/I, the test is:

```
EXEC CICS WRITEQ TS FROM(abc)
                    QUEUE(qname)
                    RESP(xxx);
   .
   .
if xxx = DFHRESP(NOSPACE)
```

RESP2(yyy)

where *yyy* is a 32-bit binary value that further qualifies the response to INQUIRE and SET commands. RESP2 codes are noted in the command descriptions. For further information, see the *CICS Application Programming Reference*.

How to use NOHANDLE

You can code a NOHANDLE option on any command to ensure that no action is taken for any condition resulting from the execution of that command.

NOHANDLE suspends the error handling that was specified in previous EXEC CICS HANDLE CONDITION commands (or with the CICS defaults) but only for the command on which you put the NOHANDLE. It has no effect on later commands, or on the error handling set by other HANDLE commands.

To use NOHANDLE on the EXEC CICS RECEIVE MAP command, all you write in COBOL is:

```
EXEC CICS RECEIVE MAP('ACCTMNU')
         MAPSET('ACCTSET') NOHANDLE END-EXEC.
```

Note: Using the C or C++ language or the RESP option implies NOHANDLE, so be careful when using C or the RESP option with the EXEC CICS RECEIVE command, because NOHANDLE overrides the EXEC CICS HANDLE AID

command in addition to the EXEC CICS HANDLE CONDITION command. (This means that PF key responses are ignored, and is the reason for testing them earlier in the ACCT code.)

How to use IGNORE condition (COBOL and PL/I only)

Just as you can arrange for control to pass to a particular label for a specific condition with an EXEC CICS HANDLE CONDITION command, so you can have the program continue when a specific condition occurs. You do this by setting up an EXEC CICS IGNORE CONDITION command to ignore one or more of the conditions that can potentially arise on a command.

EXEC CICS IGNORE CONDITION means that no action is to be taken if a condition occurs, so control returns to the instruction following the command and return codes are set in the EIB. The following example ignores the MAPFAIL condition:

```
EXEC CICS IGNORE CONDITION MAPFAIL END-EXEC.
```

While a single EXEC CICS command is being processed, it can raise one of several conditions.

Note: For example, you may have a file control command that is not only invalid but that also applies to a file that is not defined in the file definitions.

CICS checks for these conditions and passes back to your application program the first one (and only the first one) encountered.

An EXEC CICS IGNORE CONDITION command for a given condition applies only to the program you put it in, and it remains active while the program is running, or until a later EXEC CICS HANDLE CONDITION command naming the same condition is met, in which case the EXEC CICS IGNORE CONDITION command is overridden. Also, an EXEC CICS PUSH HANDLE command will disable an EXEC CICS IGNORE CONDITION until an EXEC CICS POP HANDLE command is executed.

You can choose an EXEC CICS IGNORE CONDITION command if you have a program reading records that are sometimes longer than the space you provided, but you do not consider this an error and do not want anything done about it. You might, therefore, code EXEC CICS IGNORE CONDITION LENGERR before issuing your EXEC CICS READ commands.

You can also use an EXEC CICS IGNORE CONDITION ERROR command to catch any condition considered as an error that has never been handled or ignored.

You cannot code more than 16 conditions in the same command; the conditions must be separated by at least one space. You must specify additional conditions in further EXEC CICS IGNORE CONDITION commands.

Passing control to a specified label

This applies to COBOL and PL/I programs only.

You have two ways of passing control to a specified label:

- EXEC CICS HANDLE CONDITION ERROR (*label*) *command*
- EXEC CICS HANDLE CONDITION *condition* (*label*) *command*, where *condition* is the name of an exceptional condition

How to use the HANDLE CONDITION command

EXEC CICS HANDLE CONDITION causes CICS to save information about the *condition*s and labels, and use this information to pass control to appropriate sections of your application if those conditions arise. With an active EXEC CICS HANDLE CONDITION command, control goes to whichever label you specified for that particular condition.

The same condition might arise with many different commands, and for a variety of reasons. For example, you can get IOERR during file control operations and interval control operations. First, determine which command has raised a particular condition and then investigate why it has happened. This is a reason to use the RESP option in a new CICS application. Although you only need one EXEC CICS HANDLE CONDITION to set your error-handling for several conditions, it might be awkward to pinpoint which of several EXEC CICS HANDLE CONDITION commands is currently active when a CICS command fails somewhere in your code.

If a condition arises that you have not named, you get the default action for the condition, unless the default action is to abend the task. If the default action is to abend the task, you get the action, if any, specified for the ERROR condition. If you name the condition but leave out its label, any EXEC CICS HANDLE CONDITION command for that condition is deactivated, and you revert to the default action for the condition, if and when it occurs.

Using EXEC CICS HANDLE CONDITION can be a common source of errors, because you need to deal with all possible conditions. If you use an unfamiliar command, read the *CICS Application Programming Reference* to find out what exceptional conditions are possible. Issue HANDLE commands for all of these, and ensure that you finish all the error-handling code adequately. Otherwise, the outcome might be an error handling routine that, by issuing an EXEC CICS RETURN, allows incomplete or incorrect data changes to be committed.

It is recommended that you use EXEC CICS HANDLE CONDITION, but let the system default action take over if you cannot see an obvious way around a particular problem.

If you use EXEC CICS HANDLE CONDITION commands, or are maintaining an application that uses them, take care not to cause a loop. A loop happens if you include any commands in your error routine that can cause the same condition that gave you the original branch to the routine. Notice, also, that one EXEC CICS HANDLE CONDITION command can name up to sixteen conditions, and that one of these could be the ERROR condition to deal with all remaining conditions.

Take special care not to cause a loop on the ERROR condition itself. (You can avoid a loop by restoring the system default for any ERROR condition temporarily. Do this by coding EXEC CICS HANDLE CONDITION ERROR with no label specified.) After your error processing, you can reinstate the original error action at the end of your error routine by including a second EXEC CICS HANDLE CONDITION ERROR command. If you know the previous HANDLE CONDITION state, you can do this explicitly. In a general subroutine, that might be called from several different points in your code, the EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE commands may be useful. See “Relying on the system default action” on page 239.

How to use HANDLE CONDITION ERROR

The following example shows an EXEC CICS HANDLE CONDITION command used in the COBOL example program:

```
 PROCEDURE DIVISION.
*
*    INITIALIZE.
*    TRAP ANY UNEXPECTED ERRORS.
     EXEC CICS HANDLE CONDITION ERROR(OTHER-ERRORS) END-EXEC.
*
```

In this example, control is passed to the paragraph at label OTHER-ERRORS if any condition arises that is not explicitly handled before any other CICS command.

EXEC CICS HANDLE CONDITION ERROR is the first command executed in the PROCEDURE DIVISION in this example. This is because an EXEC CICS HANDLE CONDITION command must be processed before any CICS command is processed that can raise the condition being handled. Otherwise, the EXEC CICS HANDLE CONDITION does not take effect. Note, however, that your program does not see the effects when it processes the HANDLE. The program only sees the effects later, if and when it issues a CICS command that actually raises one of the named conditions.

These examples generally use the RESP option. All the commands with RESP on them have been written with a catchall test (IF RESPONSE NOT = DFHRESP(NORMAL) GO TO OTHER-ERRORS) after any explicit tests for specific conditions. So any exceptions, other than those you expect, take control to the paragraph at OTHER-ERRORS in each program. Those relatively few commands that do not have RESP on them take control to exactly the same place if they result in any condition other than NORMAL because of this EXEC CICS HANDLE CONDITION ERROR.

Relying on the system default action

You can:

- Use the EXEC CICS PUSH HANDLE command (COBOL and PL/I only)
- Use the EXEC CICS HANDLE CONDITION command without a label (COBOL and PL/I only)
- Do nothing about the condition

You can choose to ignore the condition because you want the default action to happen. However, this can be a potential source of program maintenance problems, especially if the CICS system defaults change.

If you ignore the condition by mistake, for example, by forgetting to handle an obscure condition on an unfamiliar command, and you do not have an EXEC CICS HANDLE CONDITION to cope with that condition, you get the standard system (default) action for the condition, which in most cases abends the task.

The PUSH HANDLE and POP HANDLE commands

(COBOL and PL/I only)

The EXEC CICS PUSH HANDLE command allows you to nest your condition-handling code. For example, when calling a subroutine you may want a completely different set of EXEC CICS HANDLE CONDITION commands while in the subroutine. (EXEC CICS PUSH HANDLE enables you to suspend all current EXEC CICS HANDLE CONDITION, IGNORE CONDITION, EXEC CICS HANDLE AID, and EXEC CICS HANDLE ABEND commands. This can be useful, for example, during a branch to a subroutine embedded in a main program.)

Normally, when a CICS program calls a subroutine, the program that receives control inherits the current HANDLE commands. These commands may not be appropriate within the called program. The called program can use PUSH HANDLE to suspend existing HANDLE commands.

Use PUSH HANDLE therefore, to save your present set of HANDLE commands unaltered while you use a new set in the routine. On exit, you can reinstate the original set of HANDLE commands by using a corresponding POP HANDLE.

You can nest PUSH HANDLE and POP HANDLE command sequences within a task. Each PUSH HANDLE command stacks a set of specifications; the POP HANDLE that follows it restores them. The following example illustrates PUSH HANDLE and POP HANDLE in action:

```
*    PROCESSING FOR UNEXPECTED ERRORS.
 OTHER-ERRORS.
*       FIRST, STACK THE CURRENT CONDITION HANDLING
     EXEC CICS PUSH HANDLE END-EXEC.
     MOVE EIBFN TO ERR-FN, MOVE EIBRCODE TO ERR-RCODE.
     MOVE EIBFN TO ERR-COMMAND, MOVE EIBRESP TO ERR-RESP.
     MOVE LOW-VALUES TO ACCTERRO.
     MOVE EIBTRNID TO TRANEO.
     MOVE ERR-PGRMID TO PGMEO.
     PERFORM REASON-LOOKUP THROUGH REASON-END
         VARYING I FROM 1 BY 1 UNTIL I NOT << IXR.
     MOVE ERR-MSG (IXR) TO RSNEO.
     IF IXR << 12 MOVE EIBDS TO DSN,
         MOVE DSN-MSG TO FILEEO.
     PERFORM COMMAND-LOOKUP THROUGH COMMAND-END
         VARYING I FROM 1 BY 1 UNTIL I NOT << IXC.
     MOVE COMMAND-NAME (IXC) TO CMDEO.
     IF ERR-RESP << 94 MOVE RESPVAL (ERR-RESP) TO RESPEO
         ELSE MOVE RESPVAL (94) TO RESPEO.
     EXEC CICS SEND MAP('ACCTERR') MAPSET('ACCTSET') ERASE EXEC CICS FREEKB
         END-EXEC.
     EXEC CICS WRITEQ TS QUEUE('ACERLOG') FROM(ACCTERRO)
         LENGTH(ERR-LNG) END-EXEC.
*       IF CONDITION NOSPACE OCCURS, WAIT FOR TS TO BECOME AVAILABLE
*       NOW RESET THE PREVIOUS CONDITION HANDLING
     EXEC CICS POP HANDLE END-EXEC.
```

If you choose to include the error lookup and error message display in one of the other programs, you can link to a separate program and arrive at this code from all different points in the code. There are several points to note:

- EXEC CICS PUSH HANDLE commands can be nested. EXEC CICS PUSH HANDLE suspends the current set of HANDLEs (saving them for later use), and EXEC CICS POP HANDLE restores the most recent set suspended.
- When you link to another program, an EXEC CICS PUSH HANDLE is implied.

  That is, an EXEC CICS PUSH HANDLE occurs between the EXEC CICS LINK command and the first instruction of the linked-to program that begins with the system defaults. It can possibly do some of its own HANDLE, EXEC CICS PUSH HANDLE, or EXEC CICS POP HANDLE commands, but afterwards, the stack is popped back to the point where the EXEC CICS LINK occurred to restore the HANDLE status of the linking program when control is returned there. That is, CICS pushes at each EXEC CICS LINK and pops at each EXEC CICS RETURN.

  Note: You cannot use EXEC CICS POP HANDLE on the first command of a linked-to program to re-instate the linked-from program's EXEC CICS HANDLE CONDITIONs.

- When you EXEC CICS XCTL to another program, the current table of conditions, EXEC CICS HANDLE AID commands, and EXEC CICS ABEND commands (except for abend handling programs) are cleared, even though there is no implicit EXEC CICS PUSH HANDLE and you are staying at the same logical program level. (See "How CICS keeps track of what to do" on page 242.)

  One flaw in this example, of course, is that you do not know where to go at the end of the code and you do not need such sophistication if you are not going back.

  (You could have all your HANDLEs sent to different labels, each of which would consist of PERFORM OTHER-ERRORS, GO TO *back*, which would be wherever this particular error occurred. This shows some of the limitations of HANDLEs, even with EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE.)

How to use an EXEC CICS HANDLE CONDITION command

The easiest way to restore the system default action for a given condition is to code EXEC CICS HANDLE CONDITION without a label on a named condition. You cannot do this with C.

You might do this when, having tried the more likely conditions, you decide that anything else is either so unlikely, or so disastrous, that the only feasible option is the abend that the system default action generally gives you.

How CICS decides whether to take the system default action

CICS decides whether to take the system default action for a given condition according to the sequence of tests in the flowchart shown in the following figure:

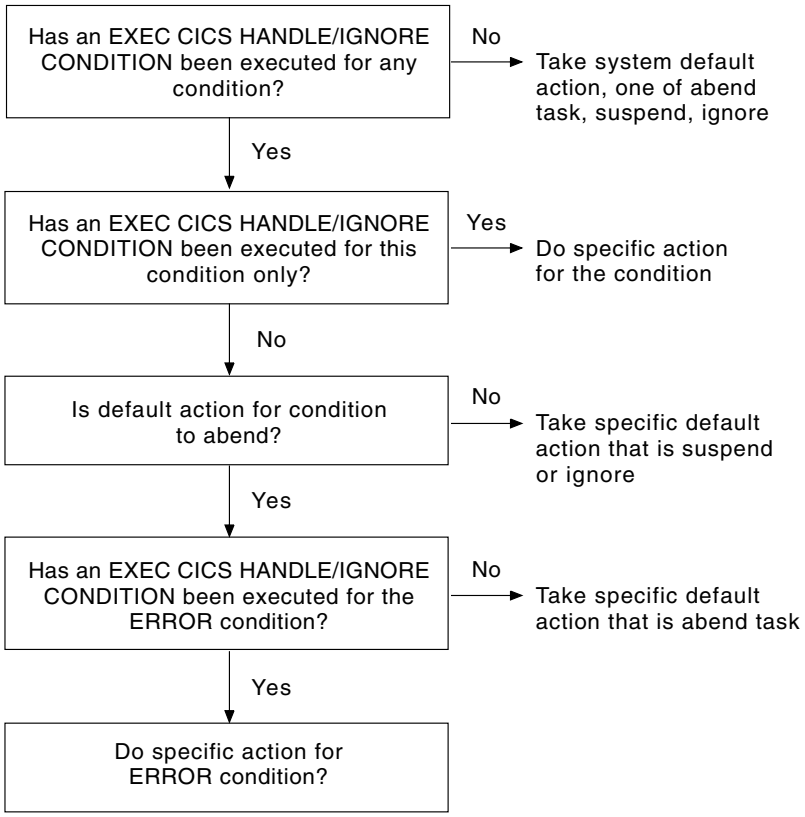


Figure 10. Deciding whether to take the system default

Mixing methods

You can temporarily deactivate the effect of any EXEC CICS HANDLE CONDITION by using the RESP, RESP2, or NOHANDLE option on a command. If you do this, you loose the ability to use any system default action for that command. In other words, you have to do your own catchall error processing.

You can also switch from ignoring a condition to handling it, or to using the system default action. For example, you can code:

```
EXEC CICS IGNORE CONDITION LENGERR END-EXEC.
   .
   .
   .
EXEC CICS HANDLE CONDITION DUPREC(DUPRTN)
     LENGERR
     ERROR(ERRHANDL)
     END-EXEC.
```

This code initially ignores condition LENGERR. So if the program raises a LENGERR condition, nothing happens. The application simply continues its processing. (Of course, if the fact that LENGERR has arisen means the application cannot sensibly continue, you have a problem.)

Later in the code, you can explicitly set condition LENGERR to the system default action by naming it in an EXEC CICS HANDLE CONDITION command without a label. Once this command has been executed, the program no longer ignores condition LENGERR, and if it subsequently occurs, it now causes the system default action. The point about mixing methods is that each condition is treated separately.

How CICS keeps track of what to do

CICS has a table of the conditions referred to by EXEC CICS HANDLE CONDITION and EXEC CICS IGNORE CONDITION commands in your application. Each execution of one of these commands updates an entry in this table. This table is created the first time any condition is mentioned in an EXEC CICS HANDLE CONDITION or EXEC CICS IGNORE CONDITION command, for the current level of condition handling. Each entry tells CICS what to do by indicating one of the four exception-handling states your application can be in, namely:

- Take no action, where control returns to the next instruction following the command that has failed.
- Go to a label, where control goes to a label that has been specified for this condition.
- Abend the task, if this condition is raised.
- Condition has never been handled or ignored.

CICS keeps a table of these conditions for each EXEC CICS LINK level and each EXEC CICS PUSH HANDLE command that has not been popped. Essentially, therefore, each program level has its own HANDLE state table governing its own condition handling.

When each condition occurs, CICS performs the following sequence of tests:

1. If the command has NOHANDLE or RESP:
   a. If the default action for this condition is to abend, control returns to the next instruction in your application program.

b. If the default action for this condition is to wait, further action is governed by item number 3 in this list.

Otherwise CICS scans the condition table to see what to do.

2. If the condition has been handled or ignored, this determines the action.
3. If the condition has never been handled or ignored, and the default action for this condition is to suspend execution:
   a. If the command has the NOSUSPEND or NOQUEUE option, control returns to the next instruction.
   b. Otherwise, the task is suspended.
4. If the condition has never been handled or ignored, and the default action for this condition is to abend, a second search is made, this time for the ERROR condition, then:
   a. If the ERROR condition has been handled or ignored, then this determines the action.
   b. Otherwise, the task is abended. You can, if you want, handle abends.

   The commands EXEC CICS ALLOCATE, EXEC CICS ENQ, EXEC CICS READQ TD, and EXEC CICS WRITEQ TS can all raise conditions for which the default action is to suspend your application program until the specified resource becomes available. So, on these commands, you have the NOSUSPEND option to inhibit this waiting and return immediately to the next instruction in your application program.

Some conditions can occur during the execution of a number of unrelated commands. If you want the same action for all occurrences, code a single EXEC CICS HANDLE CONDITION command at the start of your program.

Handling attention identifiers (EXEC CICS HANDLE AID)

The RESP, RESP2, and NOHANDLE options on EXEC CICS commands suspend the use of EXEC CICS HANDLE AID. In the absence of an EXEC CICS HANDLE AID command, control returns to the application program at the point immediately following the input command. You can suspend the EXEC CICS HANDLE AID command using the EXEC CICS PUSH HANDLE command and restore them using the EXEC CICS POP HANDLE command.

An EXEC CICS HANDLE AID command takes precedence over an EXEC CICS HANDLE CONDITION command. If an attention identifier (AID) is received during an input operation for which an EXEC CICS HANDLE AID command is active, control passes to the label specified in that command regardless of any conditions that may have occurred (but which did not stop receipt of the AID).

The EXEC CICS HANDLE AID command for a given AID applies only to the program in which it is specified, remaining active until the program is ended, or until another EXEC CICS HANDLE AID command for the same AID is met, in which case the new command overrides the previous one.

When control returns to a program from a program at a lower logical level, the EXEC CICS HANDLE AID commands that were active in the higher-level program before control was transferred from it are reactivated, and those in the lower-level program are deactivated.

If no EXEC CICS HANDLE AID command is active for any PA key, PF key, or the CLEAR key, but one is active for ANYKEY, control is passed to the label specified

for ANYKEY. An EXEC CICS HANDLE AID command for an AID overrides the EXEC CICS HANDLE AID ANYKEY command for that AID.

If a task is initiated from a terminal by use of an AID, the first EXEC CICS RECEIVE command in the task does not read from the terminal but only copies the input buffer (even if the length of the data is zero) so that control may be passed by means of an EXEC CICS HANDLE AID command for that AID.

An EXEC CICS RECEIVE MAP command with the FROM option does not cause a HANDLE AID command to be invoked because no terminal input is involved.

Note: If you use NOHANDLE (or the RESP option, which invokes NOHANDLE), it suspends the EXEC CICS HANDLE AID function. If you want to change the program processing, depending on the attention key pressed, compare the contents of EIBAID with the fields in the standard attention identifier list (DFHAID), and then transfer control to the routine needed to perform the function you want.

Abend handling

A program-level abend exit facility is provided in CICS so that you can write exits of your own that can be given control during abnormal termination of a task. An example of a function performed by such an abend exit is the cleanup of a program that has started but not completed normally.

Abnormal termination can occur because of:

- A user request by, for example:
  ```
  EXEC CICS ABEND ABCODE(...)
  ```
- A CICS request as a result of an invalid user request, for example, an invalid EXEC CICS FREEMAIN request, which gives transaction abend code A47B.
- An exception raised while running user code. The task is abended with code ASRA.
- An exception raised while executing a system call in user code; the task is abended with code ASRB.
- A looping task, in which case the task is abended with code AICA.Refer to the *CICS Problem Determination Guide* for related information.

For information on the transaction abend codes for abnormal terminations that are initiated by CICS, their meanings, and your responses, see *CICS Messages and Codes*.

The EXEC CICS HANDLE ABEND command activates or reactivates a program-level abend exit within your application program; you can also use this command to cancel a previously activated exit.

When activating an exit, you must use either the PROGRAM option to specify the name of a program to receive control, or (for COBOL programs only) the LABEL option to specify a routine label to which control will branch, when an abnormal termination condition occurs. The code called in the PROGRAM option is referred to as the abend exit program; that called in the LABEL option is referred to as the abend exit routine.

An EXEC CICS HANDLE ABEND command overrides any preceding such command in any application program at the same logical level. (See “Application

program logical levels" on page 174.) Each application program of a transaction can have its own abend exit, but only one abend exit at each logical level can be active.

Note that the EXEC CICS HANDLE ABEND LABEL command is not able to handle an abend user code, such as an ASRA caused by a segmentation violation.

When a task is abnormally terminated, CICS searches for an active abend exit, starting at the logical level of the application program in which the abend occurred, and proceeding, if necessary, to successively higher levels. The first active abend exit found, if any, is given control. This procedure is shown in "Creating a program-level abend exit," which also shows how subsequent abend processing is determined by the user-written abend exit.

If no abend exit is found, CICS abnormally terminates the task.

To prevent recursive abends in an abend exit, CICS deactivates the exit upon entry to the exit routine or program. If a retry of the operation is desired, the application programmer can branch to a point in his program that was in control at the time of the abend and issue an EXEC CICS HANDLE ABEND RESET command to reactivate the abend exit. This command can also be used to reactivate an abend exit (at the logical level of the issuing program) that was canceled previously by an EXEC CICS HANDLE ABEND CANCEL command. For COBOL programs only, you can suspend the EXEC CICS HANDLE ABEND command by means of the EXEC CICS PUSH HANDLE and EXEC CICS POP HANDLE commands as described in "The PUSH HANDLE and POP HANDLE commands" on page 239.

Note that when an abend is handled, dynamic backout will not occur unless the abend handler issues an explicit EXEC CICS SYNCPOINT ROLLBACK or abends from the top logical level.

Some abends cannot be handled with EXEC CICS HANDLE ABEND, for example A141 and A142.

Note: Abend codes can be 1 to 4 characters with no imbedded blanks. Additionally, they should not begin with the capital letter "A" as this is reserved for CICS usage.

Creating a program-level abend exit

Abend exit programs can be coded in any supported language, but abend exit routines must be coded in the same language as their program.

Upon entry to a CICS application abend exit program, no addressability can be assumed other than that normally assumed for any application program coded in that language.

Upon entry to a COBOL abend exit routine, the working storage is in the same state as it was before the abend occurred.

There are three methods for terminating processing in an abend exit routine or program, as listed below. Note however that any abend processing should always terminate with an abend, except for abends generated as a result of application program logic.

- The EXEC CICS RETURN command indicates that the task is to continue running with control passed to the program on the next higher logical level. If no such program exists, the task is terminated normally.

- The EXEC CICS ABEND command indicates that the task is to be abnormally terminated with control passed to an abend exit specified for a program on a higher logical level. EXEC CICS ABEND at the top level terminates the task abnormally.
- A branch to retry an operation. When you are using this method of retrying an operation, and you want to reenter the original abend exit routine or program if a second failure occurs, the abend exit routine or program should issue the EXEC CICS HANDLE ABEND RESET command before branching. This is because CICS disables the exit routine or program to prevent it reentering the abend exit.

Restrictions on retrying operations

If an abend occurs during the invocation of a CICS service, you should be aware that issuing a further request for the same service may cause unpredictable results because the reinitialization of pointers and work areas, and the freeing of storage areas in the exit routine may not have been completed.

You should not try to recover from ATNI abends by attempting further I/O operations. This abend will result in a TERMERR condition, requiring the session to be terminated in all cases.

If intersystem communication is being used, an abend in the remote system may cause a branch to the specified program or label, but subsequent requests to use resources in the remote system may fail.

Coding considerations for recovery

This information offers guidance on aspects of coding that facilitate recovery and restart in application programs.

Some points for consideration about the EXEC CICS HANDLE ABEND termination command and program level abnormal termination exits are:

- If an abnormal termination occurs during the invocation of a CICS service, you should be aware that issuing a further request for the same service may cause unpredictable results, because the re-initialization of pointers and work areas, and the freeing of storage areas in the exit routine, may not have been completed.
- A command level program can obtain the current CICS abnormal termination code by using the EXEC CICS ASSIGN ABCODE command.
- In program-level abnormal termination exit code, you may wish to perform actions such as those listed below. (However, it is recommended that you keep abnormal termination exit code to a minimum.)
  - Record application-dependent information relating to that task in case it terminates abnormally.
  - If you require a dump to be initiated, you should do this in the exit code at the same program level as the abnormal termination. If the dump is initiated at a program level higher than where the abnormal termination occurred, valuable diagnostic information may be lost.
  - Attempt any local recovery that may be desired.
  - Send a message to the terminal operator if, for example, you can deduce that the abnormal termination is due to unusable input data. If you can't send to a terminal operator, write to a TDQ (for error messages). This can, in turn, be redirected to CSMT.

– For transactions that CICS is to dynamically back out if an abnormal termination occurs, you should beware of writing an abend handler that ends with an EXEC CICS RETURN command. This would indicate to CICS that the transaction had ended normally and would therefore prevent dynamic transaction backout.

You must code abend handlers in the same language as the program of which they are a part.

For information on the transaction abnormal termination codes initiated by CICS, their meanings, and the audience actions, see *CICS Messages and Codes*.

You must terminate a transaction for which an IOERR condition has been raised, by issuing an EXEC CICS ABEND command. Any program that attempts to process an IOERR condition for a recoverable resource must not issue an EXEC CICS RETURN or EXEC CICS SYNCPOINT command.

An EXEC CICS RETURN or EXEC CICS SYNCPOINT command would cause the changes to recoverable resources to be committed.

Debugging services

The CICS API provides services that:

- Allow you to trace the flow of control in your application program and in the region
- Allow you to dump storage areas

These facilities are described in the following pages.

Using the API for trace services

The CICS trace facility provides functions that allow you to determine the flow of control through an application program or through CICS itself. These trace functions allow you to control the type of trace that is produced and its destination (where this trace is written to). CICS also provides a formatter (**cicstfmt**) to convert the data written by the trace facility into a readable form. This information describes the types of trace and their related trace destinations.

There are two distinct types of tracing available within CICS:

- User trace
- System trace

Note: Sensitive information (such as passwords) may be contained in a trace output. For information on how to secure the use of trace facilities, see the *CICS Administration Guide*.

User trace

User trace allows you to trace the actions of a single task. The output is written to an operating system file. When this file is formatted by the CICS trace formatter **cicstfmt**, you can see the EXEC CICS calls made by the task and any trace entries that the application makes using EXEC CICS ENTER.

The directory used to write the user trace file is specified by the **UserTraceDirectory** attribute in the Region Definitions (RD). The file name used is specified by the **TraceFile** attribute in the User Definitions (UD) for the user running the traced task. If there is no UD for this user, the file name comes from

the **PublicUserTraceFile** attribute in the RD. So that CICS does not destroy a trace file that already exists, a sequence number is always added to the file name. CICS uses the first sequence number available. For example, if the **TraceFile** attribute value is **mytracefile** and this is the first trace file produced, CICS will write user trace to **mytracefile.001**. However, even with this protection, you should choose the file names used for this type of tracing carefully, so that users can easily identify their trace files.

User trace is enabled by the EXEC CICS TRACE command issued by the traced application, provided that the master trace flag and the user trace flag have been correctly set.

System trace

System trace allows you to trace the activities of a region, CICS off-line utilities, and CICS terminal clients. This includes entry and exit tracing of function calls, exception and non-exception events, and EXEC CICS interface calls. Because the volume of trace produced by a region can be very large, you can limit the type of trace produced and the functional area that is traced. However, you cannot suppress the generation of the exception event trace that is produced whenever a symrec is raised.

System trace can be written to up to three destinations depending on whether the trace is from a region, a CICS terminal client, or a CICS off-line utility:

- Main storage buffer
- Auxiliary trace
- External trace (on CICS for AIX only)

Main storage buffer (region only): The processes that make up a region always write any system trace they produce to the same cyclical main memory buffer, which is also called the main storage trace. This buffer appears in a system dump.

Auxiliary trace: A system trace can be written to operating system files. For CICS on Open Systems clients and off-line utilities, each operating system process that executes will write to its own file. The path and name of these files is based on the setting of the environment variable CICSTRACE. For example, if CICSTRACE is set to **mytracefile**, then the trace file names used would be **mytracefile.001**, **mytracefile.002**, and so on.

For a region, two files can be specified by the values of the **TraceFileA** and the **TraceFileB** attributes in the RD. These two files are used alternately. You can switch between them using the command CEMT SET AUXTRACE SWITCH. When you cold start a region, auxiliary trace is initially set to use the file name in **TraceFileA**. When a region is restarted, auxiliary trace is set to use the file that was not in use at the time of the previous shutdown.

CICS for Windows

The auxiliary trace files are written in the directory returned by the command:

```
cicsname -r regionname datadir
```

CICS on Open Systems

The auxiliary trace files are written in the following directory:

`/var/cics_regions/region/data`

External trace (On CICS for AIX only): If you have an event performance trace facility (EPTF) on your operating system, you can set the **ExternalTrace** attribute in the RD to **yes** to cause any system trace that a region produces to be written to the EPTF. CICS on Open Systems clients and off-line utilities write to the EPTF if the environment variable CICSEXTERNALTRACE is set to a value other than null.

Trace entry points

This information describes the points at which CICS produces trace entries during its execution for both system trace and user trace.

System trace entry points: System trace can contain the following types of trace entries:

- CICS function entry/exit trace

  This is a record of entry and exit of each CICS function. CICS functions are classified as either internal or external. External functions are those functions that are used as the interface between the various CICS modules. Internal functions are those functions that are internal to a module and are therefore not called from another module.

- Non-exception event trace

  This is a record of significant events within the CICS system. An example of a significant event is the allocation of an area of main memory.

- Exception event trace (CICS on Open Systems)

  This records unexpected errors. It is generated whenever a symrec is written to the symrec file:

  `/var/cics_regions/region/symrecs`

  where *region* is the region name.

- Exception event trace (CICS for Windows)

  This records unexpected errors. It is generated whenever a symrec is written to the symrec file called **symrecs**. The following command gives the path where the **symrec** file can be found.

  `cicsname -r regionname regiondata`

  where *regionname* is the region name.

- System message trace

  This records the message number of system messages as they are written, so that you can determine what other system activity the message is associated with.

- EXEC Interface Trace

  This records the entry and return of EXEC CICS commands. It is also found in User trace.

User trace entry points: User trace can contain the following types of trace entries:

- Application trace

  This records the TRACEID, FROM and RESOURCE values specified on an EXEC CICS ENTER command.

- Exec Interface Trace

This records the entry and return of EXEC CICS commands. It is also found in System trace.

Dump

You can use DUMP to write specified areas of memory to a file, to assist in debugging an application program or to identify why an abnormal termination or storage violation occurred. You or the CICS system can initiate and own the transaction dump. A transaction dump can be generated by:

- EXEC CICS DUMP calls
- EXEC CICS ABEND calls
- Transaction abnormal terminations (including abnormal terminations generating ASRA and ASRB abnormal termination codes when an exception occurs within an application program belonging to that transaction).

CICS system dumps can be generated by:

- CICS system abnormal terminations
- CICS system shutdowns
- CEMT PERFORM SNAP commands
- The transaction abnormal termination code ASRA when an exception occurs in an application program
- The transaction abnormal termination code ASRB when a system call is made incorrectly in an application program.
- EXEC CICS PERFORM SNAP Application Programming Interface.

The dump facility uses the dump sequence number to give each dump file a unique name. CICS increments the dump sequence number each time a valid request to dump occurs. CICS records the number of dumps taken and the number of dump write errors as they occur. You can request and output dump statistics whenever you require. You can disable some types of dump. For example, you can disable all dumps except for those produced as a result of an EXEC CICS DUMP command.

Also, the dump facility uses a number of directories to write the dump data. These directories are sub-directories of the dump directory. The dump sub-directories can be symbolically linked to directories on different physical devices, which gives you more space to save dumps in.

You can ask the CICS operator to request a dump when the region is shut down. Once complete, CICS saves the dump sequence number for the next auto start of the region.

The CICS Runtime Resource Management (CEMT) transaction allows you to use CEMT INQUIRE/SET DUMP and CEMT INQUIRE/SET DUMP OPTIONS to control the dump facility interactively during runtime.

Maximizing Dump Information

There are several things you can do while coding and preparing your applications in order to maximize the information you can use to debug the application from a transaction dump. This is expecially useful when the application is running on a production region, when on-line trace and debuggers are not available.

- Use the cicstran or cicstcl ″-d″ option. This will place the line number of the currently executing EXEC CICS statement in the dump so it can be easily

identified if that EXEC CICS statement generates an abend. The line number relates to the CICS source file (*filename*.ccs or *filename*.ccp).

(On CICS for AIX only)

- Use the cicstcl option ″-s″ for C, C++, or IBM Cobol to obtain assembly listings and intermediate (compiler source) files. These can be used to identify the failing line of source code if the application generates an exception, (which results in a CICS ASRA or ARSB abend) as the offset of the failing instruction may be shown in the dump.

  If the application is compiled with a separate compiler command instead of cicstcl, or if it is a non-CICS program called from a CICS program, use the compiler option ″-qlist″ to generate these listings.

  If an exception occurs in an application program (including a non-CICS subroutine), the contents of the general purpose registers are also included in the transaction dump; this information can be used in conjunction with the assembly listings for debugging the application.
- Use the cicstcl option ″-s″ for IBM Cobol applications also to generate a data map listing. This listing gives the size of all data items and the offsets of data items within data structures in the working storage area. The working storage area is included in the transaction dump; to aid location of data structures in this dumped area it is useful to include unique eyecatchers at the start of such structures. All data items in these structures can then be located in the dumped storage (for this reason, it can also be useful to include all data items in structures where possible).

Performance monitoring services

The CICS API provides services that allow you to obtain statistical information at a task level (monitoring) and at a resource level (statistics). Those services are described below.

The monitoring service

The monitoring service provides information, at the task level, about the performance of your region and your application programs. To use the monitoring service:

1. Set the **MonitorStatus** attribute of the Monitoring Definitions (MD) to **on**. You can set this attribute to **yes** or **no** to indicate whether or not CICS is to perform monitoring. If you initially set this to **no**, you can still use CEMT to set it to **yes** at a later stage. The default value is **no**.

   Note: You cannot set monitoring attributes through your application programs.
2. Use the user-definable performance monitoring program that produces performance monitoring data at a user Event Monitoring Point (EMP) which takes the form of an EXEC CICS ENTER MONITOR or an EXEC CICS ENTER PERFORM command. The options you provide with this command are passed to the user exit along with a current copy of the task's monitoring record. Your user exit can start and stop clocks and counters, and can cause the monitoring record to be written to a file. See the *CICS Application Programming Reference* for related information.

CICS provides a sample user exit program. See the *CICS Administration Reference* for related information. CICS also provides a utility (see the *CICS Administration Reference*) to help you format and display the monitoring information.

Statistics services

You can use the CICS statistics commands (EXEC CICS COLLECT STATISTICS, EXEC CICS INQUIRE STATISTICS, EXEC CICS PERFORM STATISTICS, and EXEC CICS SET STATISTICS) to produce region-wide information on resource utilization and activity. This information may be used in several ways, for example:

Region configuration
: You can find out whether or not your attribute values in the Region Definitions (RD) are correct for your system's usage pattern, by examining items such as storage usage.

Intersystem activity
: You can see how much interaction there is with other systems. This could be used to show that some functions may be better processed locally rather than remotely, for example.

Problem determination
: You can use the statistics to help track down problems as well as to provide input to performance tuning.

CICS provides a utility (see the *CICS Administration Reference*) to help you format and display the statistics information.

Chapter 10. Testing and debugging your application

This chapter discusses methods used to test and debug CICS application programs. The chapter is organized in the following manner:

- "Preparing your application for testing" discusses helpful tools for identifying problems and tips for preparing a testing environment.
- "Using standard CICS facilities to test your application" on page 254 covers using trace, dump, and journaling to aid in testing and debugging.
- "Using CICS-supplied transactions to test your application" on page 255 details using Temporary Storage Browse (CEBR), Command Level Interpreter (CECI) , Syntax Checker (CECS), and Execution Diagnostic Facility (CEDF) for debugging tasks. In addition, details about how to configure CICS to debug Micro Focus Server Express COBOL applications with Animator (CADB), Application Diagnosis Configuration (CDCN), and the IBM Application Debugging Program (**xldb**) are included.
- "Using a compiler's integrated debugging tool to debug CICS applications" on page 263 contains information for debugging with the tools that are integrated with compilers. For CICS for Windows, the debugging tools for the following compilers are discussed: IBM VisualAge for C, C++, and COBOL, Microsoft Visual C++, and Micro Focus Net Express COBOL. For CICS for Open Systems, the debugging tool with the Micro Focus Server Express COBOL compiler is discussed.

Preparing your application for testing

This section discusses tools that are helpful in identifying problems in your applications and tips for preparing your test environment.

Useful tools for identifying problems

CICS and the operating system provide many tools, attributes, and commands to help you detect errors. Tips for your use include the following suggestions:

- Make use of CICS-supplied transactions and the compiler debugging tools that can help you isolate problems. These aids are discussed in "Using CICS-supplied transactions to test your application" on page 255 and "Using a compiler's integrated debugging tool to debug CICS applications" on page 263.
- Set the **IntrospectInterval** attribute in the Region Definitions (RD) to enable CICS to check the internal consistency of its data structures automatically. This is particularly useful for identifying when a user transaction has written over a CICS data area.
- Set the **SafetyLevel** attribute in the RD to *normal*. This prevents user transactions from writing into CICS private storage. An ASRS abend occurs if an attempt to write to private storage is made. Again, this is useful for identifying when a user transaction has written over a CICS data area.
- Code EXEC CICS ENTER commands in your program to help you to debug it. This is discussed in "Trace and dump" on page 254.
- Use CICS journaling facilities to log application performance. This is discussed in "Journals and error handling" on page 255.
- Use the EXEC CICS DUMP and CEMT PERFORM SNAP DUMP facilities to dump areas of storage in searching for a problem.

- Use EXEC CICS PERFORM SNAP API to dump the system memory from the CICS application code for debugging.
- Use the replaceable **cicsterm** (CICS on Open Systems)process as a vehicle to help automate tests that involve user input and output.

For further information about finding a problem, see the *CICS Problem Determination Guide*.

Preparing your testing environment

Before you begin testing, complete the following tasks:

- Define your resource settings.

  To test your application, or to test a component part of it, definitions must be set in the region's resource definition database. This is described in the "Configuring CDCN" on page 259. Further information is included in *CICS Administration Reference* .
- Set up test data.

  Your application usually needs test data. For extrapartition transient data queues, use operating system facilities to set up the data. For temporary storage queues and intrapartition transient data queues, use Temporary Storage Browse (CEBR) to set up the data. For more information, see the *CICS Administration Reference*.

Using standard CICS facilities to test your application

CICS provides several facilities that help you to debug the program. See the *CICS Problem Determination Guide*.

The following CICS facilities can help you to debug your application:
- Trace and dump
- Journals and error handling

Trace and dump

As described in "Debugging services" on page 247, the CICS trace control facility provides a debugging and monitoring aid. Use the EXEC CICS ENTER command to specify user trace entry points or user Event Monitoring Points (EMPs). Use the EXEC CICS TRACE command to turn the CICS trace facility on or off. The offline program **cicstfmt** formats CICS trace data, outputting the results to standard output. See the *CICS Administration Reference* for related information.

The CICS dump control facilities allows the dumping of specified areas of main storage onto a sequential file. Use the offline utility **cicsdfmt** to format and print the required dump file. See the *CICS Administration Reference* for related information.

Note that when the TRACE, DUMP, and CEMT PERFORM SNAP commands are used, and the application program that is coded with EXEC CICS PERFORM SNAP API is run, the IBM Application Debugging Program (xldb) must be turned off because it prevents the process from completing.

Journals and error handling

CICS provides facilities for creating and managing journals during application and transaction execution. A journal is a special purpose byte-oriented file. The content of journal records is specified by the application writing to the journal. The CICS journaling facility provides:

- A system journal (log)
- User journals 1 to 99
- Checking of access permissions
- Error handling. It raises appropriate condition or abnormal termination codes, and generates messages to appropriate destinations.

See "CICS journaling" on page 138 and "Error-handling services" on page 233.

Using CICS-supplied transactions to test your application

The following CICS-supplied transactions can be used to help you test and debug your application programs:

- Temporary Storage Browse (CEBR), which is used to browse and manipulate temporary storage queues.
- Command Level Interpreter (CECI) and Syntax Checker (CECS), which are used to test the syntax of EXEC CICS commands.
- Execution Diagnostic Facility (CEDF), which is used to invoke the Execution Diagnostic Facility (EDF).
- Application Diagnosis Configuration (CDCN), which is used to turn the IBM Application Debugging tool (**xldb**) on and off. (This tool is used on CICS for AIX only.)
- CADB can be used to configure CICS to run Micro Focus Server Express COBOL applications through Animator. (Animator is the debugger that is supplied with Micro Focus Server Express COBOL). CADB is supported with CICS on open systems only. See the *CICS Administration Reference* for more information about CADB.
- CJDB, which invokes the Java debugging facilities. See "Accessing Java debugging facilities with the CJDB transaction" on page 271 for more information.

Using Temporary Storage Browse (CEBR)

Temporary Storage Browse (CEBR) is a debugging aid that you can use to browse (read without changing) and manipulate the contents of CICS temporary storage queues. CEBR allows you to answer the following questions:

- Did the record go to the correct queue?
- Are the correct number of records in the queue?
- Exactly what data is held in the temporary storage queue?
- What is the current content of the transient data queue to which this application program has just written?
- Do the transient data records contain the correct information?

You can also use CEBR to copy transient data queues to temporary storage (although you cannot read an output extrapartition transient data queue) and to copy temporary storage to transient data queues (although you cannot write to an input extrapartition transient data queue).

To use CEBR, enter:

```
CEBR [queuename]
```

where *queuename* is the name of the temporary storage queue that you wish to browse. If you do not specify a queue name, the CEBR transaction generates one for you. The generated queue name is CEBR*name*, where *name* is your terminal identifier.

Using CEBR with transient data

The GET command reads each record in the specified transient data queue and writes it at the end of the temporary storage queue you are browsing, until the transient data queue is empty. You can then view the records that were in the transient data queue. When you have finished your inspection, you can copy the temporary storage queue back to the transient data queue (using the PUT command) if you wish. This usually leaves the transient data queue as you found it, but not always. Here are some things you need to be aware of when using the GET and PUT commands:

- If you want to restore the transient data queue unchanged after you have browsed it, make sure that the temporary storage queue on display is empty when the GET command is issued. Otherwise the existing temporary storage records will be copied to transient data queue on the subsequent issue of the PUT command.
- After you GET a transient data queue and before you PUT it back, other tasks may write to that transient data queue. When you issue your PUT command, the records in the temporary storage queue are copied after the new records, so that the records in the queue are no longer in the order in which they were originally created. Some applications depend on sequential processing of the records in a queue.
- After you issue the GET command on a recoverable transient data queue, no other task can read that queue until your transaction ends. If you entered CEBR from CEDF, the CEDF transaction must end; although, you can respond *yes* to the `continue` question if you are debugging a pseudoconversational sequence of transactions. If you invoked CEBR directly, you must end CEBR.
- Similarly, after you issue a PUT command to a recoverable transient data queue, no other task can write to that queue until your transaction ends.

The GET and PUT commands do not need to be used as a pair. You can create or add to a transient data queue from a temporary storage queue with the PUT command at any time. If you are debugging code that reads a transient data queue, you can create the queue in temporary storage (with CECI, or CEBR GET, or by program), and then refresh the transient data queue as many times as you like from temporary storage. Similarly, you can delete a transient data queue by issuing a GET command without issuing a corresponding PUT command.

For more information, see the *CICS Administration Reference*.

Using Command Level Interpreter (CECI) and Syntax Checker (CECS)

CECI allows you to check the syntax of, interpret, and run EXEC CICS commands. Syntax Checker (CECS) allows you to check the syntax of EXEC CICS commands, but does not allow you to invoke them.

The CECI and CECS transactions perform a dual role in the operation of a CICS system.

- When writing application programs, you can check the syntax of the whole CICS command level application programming interface. If you are using CECI, you can carry through most of the commands to invocation, and you can request to see the results of the invocation.
- Using CECI provides a means of interaction with the system. For example, you can correct a file control record that has been overwritten with invalid data, create or delete a temporary storage queue, and so on. CECI provides a useful extension to the facilities provided by the runtime resource management transaction, CEMT.

To run the command interpreter and syntax checker, you must have a transaction security level (TSL) key that matches the transaction security key defined in the Transaction Definitions (TD).

The TD entry for the CECI transaction specifies, by default, that resource level security checking is required for any resources referenced with the interpreter. This checking applies to files, transient data queues, temporary storage queues, programs, transaction identifiers of the EXEC CICS START command, and journal file identifiers.

To use CECI or CECS, enter:

```
{CECI | CECS} command
```

where *command* is the CICS API command whose syntax you wish to check, or that you wish to interpret.

Note: If the resource security level (RSL) key specified in the appropriate resource definition is not matched by RSL keys of the user that is signed on, the resource security check fails, and the response to the command is the NOTAUTH condition.

For more information, see the *CICS Administration Reference*.

Using Execution Diagnostic Facility (CEDF)

Execution Diagnostic Facility (CEDF) is used to start or stop an Execution Diagnostic Facility (EDF) session on a terminal or a region. EDF enables you to test an application program that has been preprocessed using the **-e** option of the CICS translator (**cicstran**), without modifying the program. If the **-d** option of the **cicstran** command is also used, the translator source listing also has line numbers that EDF can use. CICS passes control to EDF at specific interception points. EDF displays the state of the application program at the interception points, and allows you to interact with EDF screens to both view additional information about the program, and to overtype certain areas of the screen to test the execution of the program, before returning control to the application code.

In CICS, you can use EDF only in dual screen mode; that is, you cannot run the CEDF transaction and the application being debugged on the same terminal. However, if you have a terminal that supports a graphical user interface (such as an X-terminal), you can use two windows on the same physical terminal. You can use EDF only from a terminal that has a screen width of 80 columns or more, and a screen depth of 24 lines or more. This functionality is available on the AIX, HP-UX, and Solaris platforms. It is not available on the Windows platform.

When you run CEDF to debug a transaction, the transaction runs, sets a flag, and exits, all in a very short time. Therefore, CEDF is not running in the system as you run your transaction (in a similar way that a pseudoconversational transaction is

not running and most of the time does not appear on INQ TASK displays). On CICS, both terminals are under control of a single task (the user task), which is effectively debugging itself and sending the output to the terminal running under CEDF.

On the Windows platform, you cannot use CEDF to debug a transaction on a remote CICS system. This is only possible if EDF can be used in single screen mode on the remote system. You would need to run the CRTE transaction first, to establish a routing session to the remote system; then, you could transaction route CEDF to that system.

Similarly, you cannot use CEDF to debug a transaction on a remote terminal.

You can use EDF to test user transactions (application programs) and the function shipping mirror transaction supplied by CICS. You cannot use it for any other transaction provided by CICS.

To use CEDF, enter:

```
CEDF { termid | sysid } [,ON | ,OFF ]
```

where *termid* is the four-character identifier of the terminal (*termid*) on which the transaction to be tested is being run and *sysid* is the four-character identifier of the remote region (*sysid*) when you want to test inbound transactions from a different region. The transaction must be defined in the Terminal Definitions (WD), unless it is generated from a terminal autoinstall. The remote region must be defined in the Communications Definitions (CD).

The `ON` setting specifies that the EDF session is to be started, and the `OFF` setting specifies that the EDF session is to be ended.

For more information, see the *CICS Administration Reference*.

Using CDCN and the IBM Application Debugging Program (xldb) with CICS for AIX only

The IBM Application Debugging Program (**xldb**) provides the ability to debug IBM COBOL, C, C++, or PL/I programs on the AIX platform.

To debug a program you must:

1. Change the **AllowDebugging** flag in the region definition stanza file to **yes**.
2. Set the **RSLCheck** setting of the CDCN transaction definition to **none**.
3. Compile the program to be debugged with the **-a** flag, for example:

   ```
   cicstcl -a -lC prog.ccs
   ```
4. Run CDCN to turn on the debugging tool. (See the *CICS Administration Reference* for related information.)
5. Run the transaction.

Note: Refer to the documentation provided with the IBM Application Debugging Program for information on installing product.

Only programs compiled for debug will produce symbolic output from the IBM Application Debugging Program when run under control of the IBM Application Debugging Program. If the IBM Application Debugging Program is triggered for a

program that has not been compiled for debug, the IBM Application Debugging Program displays the program as assembler statements.

Note: While the IBM Application Debugging Program is active, do not use the CICS CEMT command PERFORM SNAP DUMP and run applications that are coded with EXEC CICS PERFORM SNAP API . When CICS takes a snap dump, it dumps out to all the application servers. If the IBM Application Debugging Program is running, CICS does not have control of the application servers, and the dump cannot complete.

Note:

Configuring CDCN

To ensure that the IBM Application Debugging Program functions with minimal disruption to CICS, or any other software, two preparation steps are required:

- Resource settings must be indicated for the use of the IBM Application Debugging Program
- Development conventions must be observed in regard to the use of the IBM Application Debugging Program.

"Required resource settings for the IBM Application Debugging Program" discusses the settings that must be made. "Development conventions required for effective use of the IBM Application Debugging Program" on page 261 discusses the conventions required to ensure proper functioning of the IBM Application Debugging Program.

Required resource settings for the IBM Application Debugging Program: Some of the information used to run the IBM Application Debugging Program is obtained from the **.Xdefaults** file. This file determines the defaults for fonts, colors, and the location of the program source. CICS determines the location of the **.Xdefaults** file from the value of the variable XENVIRONMENT. The following lines must be included in the **.Xdefaults** file:

```
xldb.sourceSearchPath: /var/cics_regions/regionName/bin . [directoryNames]
xldb.automaticBreakpoints: NO
xldb.ignoreSignals: USR2
xldb.multiprocessDebug: false
xldb.specialLayout: true
xldb.ThreadsScrollbars: both
xldb.ThreadsGeometry: 760x300+40+100
```

In this example, *regionName* is the name of the region, the period represents the current directory, and *directoryNames* are the names of other directories in which the IBM Application Debugging Program should search for source code.

Each of the resource settings included in the **.Xdefaults** file are discussed in detail in the following sections.

The automaticBreakpoints resource setting: The `automaticBreakpoints` setting prevents breakpointing on system calls. However, it also prevents automatic breakpointing on application calls to other programs. Therefore, you must ensure that breakpoints are explicitly set at the start of all application programs that are to be debugged.

Note: The IBM Application Debugging Program sets the default value for the `automaticBreakpoints` resource to **yes**. It is important that you explicitly set this value to **no**.

The ignoreSignals resource setting: The `ignoreSignals` setting prevents breakpointing on the CICS internal inter-process signal and ensures that the signal is passed on to CICS automatically. All signals (other than CICS inter-process signal) MUST be passed on to CICS manually to avoid the loss of useful information that can be displayed for the user. For example, if application code caused a SIGSEGV signal to occur, you would want to manually pass this signal because it would alert the user to a problem occurring due to an attempt to access an invalid storage location. Ignore the signal by using any of the following commands: **continue**, **next**, **step**, **machine step**, or **return**. Pass on non-CICS internal inter-process signals by using the IBM Application Debugging Program's **signal** command.

The multiprocessDebug resource setting: The `multiprocessDebug` setting ensures that the parent process is the only process followed when the process uses a UNIX **fork** function to create a new process. This setting also prevents signals generated by a child from erroneously being passed on to the IBM Application Debugging Program, rather than to the parent process.

The specialLayout resource setting: The `specialLayout` setting allows you to alter scrollbar settings. This setting must be indicated as **true**, to enable the settings of the `ThreadsScrollbars` and `ThreadsGeometry` resources to be to be registered.

The ThreadsScrollbars and ThreadsGeometry resource settings: The `ThreadsScrollbars` and `ThreadsGeometry` settings are used to make adjustments to the Threads window. These adjustments make it easier to understand what is happening in a multi-threaded CICS Application Server environment by enabling you to label the main thread. A user-written application runs in a single thread of an Application Server. CICS uses the other threads in the Application Server to handle exceptions in the application and to process signals from other CICS processes.

Use the following steps to label the main thread:

1. Locate the instance of your application in the Callers Stack window and in the Source window. When you are able to identify these instances, open the Threads window, and set a label on the current thread. If the IBM Application Debugging Program then indicates a breakpoint, the Caller Stack is likely to look familiar. If the Caller Stack does not look familiar, it indicates that the breakpoint is occurring for the Caller Stack of a different thread.
2. Open the Threads window, and click on the labeled thread. The IBM Application Debugging Program then displays the Caller Stack associated with the user-written application.

Indicating whether IBM Application Debugging Program settings are globally or individually defined: On the AIX platform, CICS determines the location of the **.Xdefaults** file from the value of the variable XENVIRONMENT. Furthermore, the location of the XENVIRONMENT environment variable tells CICS whether the IBM Application Debugging Program settings specified in the **.Xdefaults** file are to be shared by all users in a region, or whether individual users in a region have their own IBM Application Debugging Program settings.

- To configure an environment where all users in a region on an AIX machine share the same IBM Application Debugging Program settings:
  1. Add the following line in the file identified by the CICSDEBUGENV environment variable or in the file **/var/cics_regions/***regionName***/environment**:

     ```
     XENVIRONMENT=XdefaultsPathname
     ```

     In this example, *XdefaultsPathname* is the full pathname of the global **.Xdefaults** file, to the file **/var/cics_regions/***regionName***/environment**.

2. Cold start CICS. All users of the region will then share the same IBM Application Debugging Program settings.

- To configure an environment where individual users in a region have individually-defined IBM Application Debugging Program settings:
  1. Add the following line to a file *envFile* in your personal storage:

     ```
     XENVIRONMENT=XdefaultsPathname
     ```

     In this example, *XdefaultsPathname* is the full pathname of the user's personal **.Xdefaults** file.
  2. In the window where you run cicsterm, export the file *envFile* from your personal storage by entering the following command:

     ```
     export CICSDEBUGENV=envFile
     ```
  3. In addition, you can avoid the need to enter the `DISPLAY=` parameter when you run the CDCN transaction by including the following line in your *envFile*:

     ```
     DISPLAY=XserverName:displayNumber
     ```

     In this example, *XserverName*:*displayNumber* is the name of your display.

This functionality can apply to the Windows platform or to IBM CICS Universal Client implementations if third-party products are employed to enable X-term window capability.

Development conventions required for effective use of the IBM Application Debugging Program

Because the IBM Application Debugging Program can cause processes to stop, or to take unexpectedly long completion times, it can appear to other CICS processes that a process has failed. It is also possible that an actual failed process might not be reported to CICS.

The effects on CICS of these falsely reported process failures and unreported actual process failures are manifested in different ways. Depending on the circumstances, any of the following behaviors can occur: resource deadlocks (file and mutex locks), heuristic (non-deterministic) transaction completions, timeouts and communication failures, and exposure to the possibility of corrupted transactions.

Additionally, because CICS acts to mitigate what appears to be a major failure, further symptoms can include transaction rollbacks, spurious restarts of the processes that are being debugged, and in some cases, the complete halt of a CICS Region.

If any of these symptoms occur while using the IBM Application Debugging Program, and if the scenario is able to be recreated, it is possible that the use of the IBM Application Debugging Program is interfering with the normal operation of CICS. Suspend the use of the IBM Application Debugging Program until you are able to ensure that the problem is not being caused by interaction from the debugging program itself.

Because of the possible occurrence of these symptoms, a general knowledge of CICS internals and the architecture of surrounding systems is required to use the IBM Application Debugging Program effectively and safely. To ensure that the IBM Application Debugging Program functions with minimal disruption to CICS, or any other software, the following development conventions must be observed in regard to the use of the IBM Application Debugging Program:

- Never use the IBM Application Debugging Program within a production CICS Region.
- Always ensure that the values detailed in "Required resource settings for the IBM Application Debugging Program" on page 259 are set for the IBM Application Debugging Program.
- Set a label for the user application thread, as described in "The ThreadsScrollbars and ThreadsGeometry resource settings" on page 260.
- When running an the IBM Application Debugging Program session, ANY signal that is received during the session MUST be passed on to the application. Select **Signal** from the Commands window to pass a signal to the application.
- Avoid having either CICS trace or CICS dump enabled while running an IBM Application Debugging Program session. The trace and dump facilities lock and wait for common buffers, and this can cause a CICS region to run slowly, or even to halt, in response to other processes queueing to the resources.
- Do not use the CICS CEMT command PERFORM SNAP DUMP while the IBM Application Debugging Program is active. When CICS takes a snap dump, it dumps out to all of the application servers. If the IBM Application Debugging Program is running, CICS does not have control of the application servers, and the dump cannot complete.

Activating the IBM Application Debugging Program

The IBM Application Debugging Program does not start immediately when you use CDCN to turn the IBM Application Debugging Program on for a named resource. The resource you name acts as a trigger to turn on the IBM Application Debugging Program for the resource when you meet the following conditions:

- You satisfy the security checks
- You are able to run the CDCN transaction
- You are able to access the region database entries.

The type of resource that is being triggered determines when the actual startup of the IBM Application Debugging Program occurs. The following table details the startup contingencies associated with each type of resource involved:

Resource	Startup of the IBM Application Debugging Program triggered by
Terminal (*termid*)	The next transaction run on that terminal
System (*sysid*)	The next transaction run as a result of a request from that system
Transaction (*transid*)	The next invocation of that transaction
Program (*progid*)	The next invocation of that program

Each trigger resource can have just one session of the IBM Application Debugging Program started for it. Control of the debugging session by the trigger resource adheres to the following order of precedence:

1. Terminal (*termid*)
2. System (*sysid*)
3. Transaction (*tranid*)
4. Program (*progid*)

To illustrate this order of precedence, consider the following example:

You turn on debugging for both a terminal, identified as AR01, and for a transaction, identified as ALAN. You run transaction ALAN on terminal AR01. In this scenario, only one session of the IBM Application Debugging Program is

started— the session associated with terminal AR01. This single debug session occurs because two debug sessions cannot debug the same physical copy of the code, and the terminal trigger takes precedence over the transaction trigger.

It is possible to have multiple copies of the same program code being debugged at the same time. A second invocation of ALAN that is run from a different terminal will start a second session of the IBM Application Debugging Program. This second debug session is associated with transaction ALAN.

For more information, see the *CICS Administration Reference*.

Deactivating the IBM Application Debugging Program

When the IBM Application Debugging Program is first started, it remains attached to the CICS application server (cicsas) process, debugging one or more programs, until the end of the transaction (even when it has been triggered to debug just a single program), or until you choose to quit from the IBM Application Debugging Program. The IBM Application Debugging Program is attached when it connects to the cicsas process (that is, actually debugging one or more programs); the IBM Application Debugging Program is detached when it is no longer attached to the cicsas process (that is, when it has finished debugging a transaction and is waiting to be triggered again). Once the IBM Application Debugging Program has started, it can only be explicitly terminated in one of two ways:

- When the IBM Application Debugging Program is attached, using the quit facility of the IBM Application Debugging Program itself.
- When the IBM Application Debugging Program is detached, using CDCN.

Using a compiler's integrated debugging tool to debug CICS applications

Using debugging tools integrated with compilers running on CICS for Windows

The following table indicates the debugging tools supported by compilers used for (CICS for Windows):

Language	Compiler	Debugger
C	IBM VisualAge C++	idebug
C++	IBM VisualAge C++	idebug
IBM COBOL	IBM VisualAge for COBOL	idebug
C	MicroSoft Visual C++	windbg/msdev
C++	MicroSoft Visual C++	windbg/msdev
COBOL	Micro Focus Net Express COBOL	Animator

The debugging options may be switched on in either of two ways:

- Use the **-a** option with **cicstcl** compile command to enable the IBM Application Debugging Program to be used for debugging purposes. See "Using CDCN and the IBM Application Debugging Program (xldb) with CICS for AIX only" on page 258 for more information on this method.

- Use the debugging tools integrated with each language compiler. If you are building the application in separate steps, each individual compiler has debugging options which can be switched on. These options are discussed for each compiler in the following sections.

Preparing to use the debugging tool integrated with IBM VisualAge C/C++

1. The following compile and link debug parameters must be specified:

Option	Description
-O-	Combine with the `idebug` command to switch code optimization off.
-Oi-	Combine with the `idebug` command to switch inlining off.
-Ti+	Combine with the `idebug` command to generate line numbers and symbol table.
Or, if performing the link step separately:	
-DE	Combine with the `idebug` command to generate line numbers and symbol table.

2. Ensure that the PATH system environment variable includes the directory that contains your dynamic link library (DLL).
3. Create a new system environment variable DEBUG_PATH that is set to the directory path containing the source file (*progname.c*) for your DLL.
4. For C++, make a copy of the *progname.ibmcpp* program and name it *progname.dll*. Place this file in a directory that is before the *progname.ibmcpp* copy in the PATH environment variable, but not where CICS will use it in preference to the *progname.ibmcpp* file.

   The following example explains this:

```
PATH=d:\prog_dir;e:\var\cics_regions\alex1\bin.
PathName=progname is in the  PD definition.
The file progname.dll is in the d:\prog_dir directory.
The file progname.ibmcpp is in the e:\var\cics_regions\alex1\bin dir
```

Note: If you have both IBM VisualAge C++ (Windows only) and IBM VisualAge COBOL installed, the COBOL directories must be in the PATH system environment variable before the C++ directories.

Procedure for using the debugging tool integrated with IBM VisualAge C/C++:
Perform the following steps to set breakpoints, to step through the program, and to look at storage for the purpose of debugging your program.

1. Start CICS and install a terminal.
2. Use the **idebug** command from the command-line to start the IBM VisualAge Debugger.
3. Select the **Process list** button from the Debug Session Control window.
4. Select the CICS Application Server (cicsas) process that will run your transaction. Click the **Attach** button, and wait for the debugger to attach to the cicsas.
5. Select **Breakpoints** from the Debug Session Control window. Set load occurrence.
6. Enter your DLL name in the box (eg. *progname.dll*) and press **OK**.
7. Enter your transaction ID at the terminal, and press **Enter**. The transaction does not start yet because the `idebug` command has stopped the cicsas process.

8. Select the **Run** menu from the Debug Session Control window, and click **Run**. The process will continue until the DLL is loaded. A message window appears to confirm this. Click **OK**.
9. Select **Breakpoints** from the Debug Session Control window. Enter the following information; then, click **OK**:

```
Set function breakpoint
Executable:progname.dll, Source:progname.c,
                Function:  main.
```

10. Select the **Run** menu from the Debug Session Control window, and click **Run**. The process will continue until the application program is about to start.

Ending the debugging session: To end the debugging session, select the **File** menu from any of the Debug Session Control windows, and click on **Close Debugger**. This shuts down the cicsas that the debugger was attached to, but does not otherwise affect the region.

The debugger can be restarted and attached to a new cicsas.

Note: Use the CEMT INQUIRE TASK command to identify the process number being used by a particular cicsas.

Preparing to use the debugging tool integrated with IBM VisualAge for COBOL

1. The following compile and link debug parameters must be specified:

Option	Description
-g	Combine with the **cob2** compile command to switch on idebug.
/de	Combine with the **ilink** command to generate line numbers and symbol table.
or, if performing the link step separately:	
-de	Generate line numbers and symbol table

2. Ensure that the PATH system environment variable includes the directory that contains your dynamic link library (DLL), *progname.ibmcob*.
3. Create a new system environment variable CAT_OVERRIDE that is set to the directory path containing the source file (*progname.cbl*) for your DLL.
4. Set MAXSERVER=1 in your Region Definitions (RD). To do this, follow these steps:
   a. Open the IBM TXSeries Administration Tool
   b. Highlight your region, and right-click on it
   c. Select **Properties** from the popup menu
   d. Select the **Tuning** tab
   e. Change the **Maximum Application Servers** value to 1.

Note: If you have both IBM VisualAge C++ (Windows only) and IBM VisualAge COBOL installed, the COBOL directories must be in the PATH system environment variable before the C++ directories.

Procedure for using the debugging tool integrated with IBM VisualAge for COBOL: Perform™ the following steps to set breakpoints, to step through the program, and to look at storage for the purpose of debugging your program:

1. Start CICS, and install a terminal.

2. Start the IBM VisualAge Debugger by issuing the `idebug` command at the command-line.
3. Select the **Process list** button from the Debug Session Control window.
4. Select the cicsas process that will run your transaction. (If you set the MAXSERVER=1, you only have one process called cicsas to select.)
5. Click the **Attach** button. Wait for the debugger to attach to cicsas.
6. There are two alternative procedures at this step:
   a. To debug an individual program, select **Breakpoints** from the Debug Session Control window, then:
      - Select **Entry**
      - Set Executable:progname.ibmcob
      - Set Source: (leave blank)
      - Set Entry: PROGNAME (must be in upper case)
      - Click **Defer Breakpoint**
      - Click **Set**
      - Click **OK**
   b. To debug all programs compiled for debug in this process, select **Options** from the Debug Session Control window, then:
      - Debugger settings
      - Debugger properties
      - Check **Set breakpoints at entry points**
      - Click **OK**
7. Enter your transaction ID at the terminal and press **Enter**. The transaction does not start yet, because the `idebug` command has stopped the cicsas process.
8. Select the **Run** menu from the Debug Session Control window, and click on **Run**. The process will continue until the entry point of the application program is reached.

Ending the debugging session: To end the debugging session, select the **File** menu from any of the Debug Session Control windows, and click **Close Debugger**. This shuts down the cicsas that the debugger was attached to, but does not otherwise affect the region.

The debugger can be restarted and attached to a new cicsas.

Note: Use the CEMT INQUIRE TASK command to identify the process number being used by a particular cicsas.

Using the debugging tool integrated with Microsoft Visual C/C++ (windbg)

The **windbg** command is included with the Microsoft Windows Platform Software Development Kit (MSSDK). The command is located in the \mssdk\bin directory. The following compile and link debug parameters must be specified for use with this tool:

Option	Description
-Zi	Combine with the **windbg** command to generate debugging information.
-Od	Combine with the **windbg** command to switch code optimization off.

The following options should also be used for the linker: `-link -debug:full -debugtype:cv -PDB:none`

Procedure for using the debugging tool integrated with Microsoft Visual C/C++ (windbg): Perform the following steps to set breakpoints, to step through the program, and to look at storage for the purpose of debugging your program:

1. Start CICS and identify the cicsas which will run the application.

   Note: The process number being used by a particular cicsas can be identified by using CEMT INQUIRE TASK.
2. Start **windbg**, attaching it to the cicsas process using the command:

   ```
   windbg -p <cicsas pid>
   ```
3. In the Windbg main window, select **File** then **Open**, and enter the path and program name of your application source . The source is displayed in a new window.
4. Place the cursor on the line of source where the first breakpoint is desired and select **DEBUG** from the toolbar, then **Breakpoints**. The Location box displays the path and name of your program, and an offset for the breakpoint, for example:

   ```
   {,E:\var\cics_regions\region1\bin\app1.c,}@17
   ```

   Enter the name of your DLL after the source name, for example:

   ```
   {,E:\var\cics_regions\region1\bin\app1.c,app1.dll}@17
   ```

   Click **Add**, then **OK**. If the DLL is not yet loaded, a message appears indicating that the breakpoint is not instantiated.
5. Enter your transaction ID at the terminal and press **Enter**. The transaction does not start yet.
6. From the windbg control window, select the **Run** menu, and click **Go**. The process continues until the breakpoint is reached.

Ending the debugging session: To end debug, select the **File** menu from any of the debug windows, and click **Exit**. This shuts down the CICS application server that the debugger was attached to, but does not otherwise affect the region.

The debugger can be restarted and attached to a new cicsas.

Using the debugging tool integrated with Microsoft Visual C/C++ (msdev)

Currently applications need to be built within the MSDEV Studio environment to make use of the msdev debugger.

Procedure for using the debugging tool integrated with Microsoft Visual C/C++ (msdev):

1. Translate the application using **cicstran -lC** *progname*.
2. Double click the **MSDEV Studio** icon in the MS Visual C++ window.
3. Create a project workspace following instructions in the Visual C++ User's Guide. Select **Dynamic-Link Library** as the Type and enter your preferred directory in the **Location** box.
4. Add your translated application program to the project by selecting the **Insert** menu, then select **Insert Files into Project**. Select the path and file that you want.
5. Edit the file by selecting **File**, then **Open**, then specifying your file. Add the line at the top of the code:

   ```
   (void)DebugBreak();
   ```
6. Save the changed file by selecting **File** then **Save**.

7. Set the build options by selecting **Build** then **Settings**:
   - General: Not using MFC
   - C/C++
     - General: Warning level = Level 3, Debug info = C7 Compatible
     - Code generation: Use run-time lib = Multithreaded DLL, Calling convention = _stdcall
     - Optimizations: Disable (Debug)
     - Precompiled headers: Not using.
     - Preprocessor: Additional include dirs = *prodDir*\include.

     These options should generate:

     Project Options: /nologo /Gz /MD /W3 /GX /Z7 /Od i/I″*prodDir*\include″ /Fo″Debug″ /c
   - Link
     - General: Output file name = prog.dll, Object/lib modules: *prodDir*\lib\cicsprC.lib mvscrt.lib kernel32.lib, tick Generate debug info ONLY

       These options should generate:

       Project Options: *prodDir*\lib\cicsprc.lib mvscrt.lib kernel32.lib /nologo /dll /pdb:″none″ /debug /machine:I386 /out:″filename.dll″ /implib:″Debug/filename.lib″ /EXPORT:main
8. Stop the MSDEV studio.
9. Start CICS and identify the cicsas process ID which will run the application.
10. Start the MSDEV attached to the process by **msdev -p** *cicsas pid*.
11. Select **File** then **Open** and select your source file.
12. Enter your transid - the transaction does not run, but is displayed by the debugger.
13. Debug the transaction as desired.
14. To end the debugger select **File** then **Exit**. This terminates the cicsas it was attached to, but the region remains running.

Using the debugging tool integrated with Micro Focus Net Express on CICS for Windows (Animator)

Animator enables you to test an application program online without modifying the program. The tool intercepts execution of the application program at various points before displaying information about the program. Any screens sent by the application program are displayed by the tool, so that you can converse with the application program during testing, just as you would on the production system.

You cannot use Animator to debug a distributed transaction processing (DTP) back-end transaction.

Using Animator with a Micro Focus Net Express Program: There are two ways to advise the COBOL compiler that the program is to be used with Animator:

- The **-a** flag can be used with **cicstcl** command.
- The /ANIM and /INITCALL″CBL_DEBUGBREAK″ options can be used directly with the compiler.

The following is an example of using the **cicstcl** command:

```
cicstcl -a myprog.ccp
```

This produces the files **myprog.cbmfnt**, **myprog.idy**, and **myprog.gdy**.

The **cicstcl** command produces a .cbmfnt file. If you want to use one of the other types of files that can be used by Micro Focus Net Express, you must invoke the **cicstran** command and the compiler directly.

To produce a **.cbmfnt** file, follow the steps shown in the following example:

1. `cicstran myprog.ccp` — this produces the the file **myprog.cbl**
2. `cobol /ANIM /CALL-RECOVERY /DATA-CONTEXT /INITCALL"CBL_DEBUGBREAK" myprob.cbl` this produces the files **myprog.obj**, **myprog.idy**, and **myprog.gdy**.
3. `cbllink -D -Mmyprog -Omyprog.cbmfnt myprog.obj` *prodDir*`\lib\cicsprCBMFNT.lib` – this produces the file **myprog.cbmfnt**.

To produce a .int file, follow the steps shown in the following example:

1. `cicstran myprog.ccp` — this produces the the file **myprog.cbl**
2. `cobol /ANIM /NOGNT /CALL-RECOVERY /DATA-CONTEXT /INITCALL"CBL_DEBUGBREAK" myprob.cbl` — this produces the file **myprog.int**.

Procedure for using Animator with Micro Focus Net Express:

1. Place the **.cbmfnt** file or the **.int** file in the region **bin** directory, or wherever the pathname in the PD stanza indicates as normal.
2. Set the COBIDY variable in the region's environment file to point to the location of the **.idy** files. This must be on a local drive.
3. Set the COBCPY variable in the region's environment file to point to the source, that is, the**.cbl** files. This must be on a local drive.

The regions can now be started, and any COBOL program conforming to the above procedure is able to use Animator automatically.

For more information on invoking Animator, see the Micro Focus Net Express documentation.

Using debugging tools integrated with compilers running on CICS on Open Systems

Using the debugging tool integrated with Micro Focus Server Express COBOL on CICS on Open Systems (Animator)

Animator enables you to test a Micro Focus Server Express COBOL application program online without modifying the program. The tool intercepts execution of the application program at various points before displaying information about the program. Any screens that the application program sends are displayed by the tool, so that you can converse with the application program during testing, just as you would on the production system.

CICS supports cross-session debugging with Micro Focus Server Express COBOL. Cross-session debugging enables the user to use Animator in a different terminal window from that in which the program to be debugged is running. Animator is first started in one session and remains in waiting state until it attaches to a Micro Focus Server Express COBOL program that has been started in another session. Details of setting up the cross-session debugging are explained below in the setup procedure.

Use the following procedure to configure CICS to run Micro Focus Server Express COBOL programs with Animator.

1. Set the AllowDebugging attribute in the region definition stanza to **yes**. This setting allows you to run CADB in that CICS region.
2. Using the CADB transaction, enable CICS to run Micro Focus Server Express COBOL programs with Animator. See the *CICS Administration Reference* for more information about CADB. You must run CADB before you run the Micro Focus Server Express COBOL program; otherwise, the program runs to completion without attaching to the Animator.
3. Compile a Micro Focus Server Express COBOL program. Use the **cicstcl** command with the -a flag. For example:

   ```
   cicstcl -a -lCOBOL TEST.ccp
   ```

   This command produces TEST.idy and TEST.int.
4. Setup the cross-session debugging with the following steps.
   a. Open a new terminal window to start the Animator.
   b. Ensure that the basic environment for CICS and Micro Focus Server Express COBOL has been set up. (PATH, LIBPATH, NLSPATH, and so on for CICS; COBDIR and so on for Micro Focus Server Express COBOL). The LIBPATH (LD_LIBRARY_PATH on Solaris and SHLIB_PATH on HP-UX) should also include the Micro Focus Server Express COBOL library paths.
   c. Set the environment variable COBANIMSRV. Use the same string that has been entered for the COBANIMSRV ID field in the CADB screen.
   d. Use the cicsanimsrv program to start the Animator. The user that starts the Animator process should be part of the cics group. The Animator waits for the Micro Focus Server Express COBOL programs to attach.

      Note: cicsanimsrv is a CICS-supplied program that is a wrapper to the cobanimsrv program that Micro Focus Server Express COBOL supplies. In a CICS environment, the CICS Application Server Process runs programs under the cics user ID. The cicsanimsrv program internally runs the Animator process also under the cics user ID. This action ensures that a program that the CICS Application Server Process is running, successfully attaches to the Animator. The user and group IDs for the cicsanimsrv program should match those of the user cics. For example, if the cics user has a user and group ID of cics : cics, cicsanimsrv should also be cics:cics (which is the default). Otherwise, cicsanimsrv permissions must be changed as necessary.

When you have done these steps, the Animator process waits for the Micro Focus Server Express COBOL programs to attach. Run the transaction that executes a Micro Focus Server Express COBOL program.

By default, when a Micro Focus Server Express COBOL program that is to be debugged is run, it waits indefinitely until an Animator process is started. However, you can use environment variable CICS_ANIMATOR_TIMEOUT to configure the wait time. Timeout values are set in milliseconds. A value of -1 means wait indefinitely (which is default in CICS); a value of 0 (zero) means DO NOT WAIT.

Note: You cannot use Animator to debug Micro Focus Server Express COBOL programs that are remote to this region. For more information on cross-session debugging and Animator, refer to Micro Focus Server Express COBOL documentation.

Accessing Java debugging facilities with the CJDB transaction

The CJDB transaction can be used to access Java debugging facilities.To use this transaction, do the following:

1. Compile the Java program with debugging enabled.
2. Stop the region where the program is to run, as described in the *CICS Administration Guide for Open Systems*.
3. Enable debugging in the region by setting the following property in the /var/cics_regions/*region_name*/database/RD/RD.stanza file:

   ```
   AllowDebugging=yes
   ```

4. Use the cicsadd command to add transaction and program definitions to the region:

   ```
   cicsadd -c td txn_name ProgName=program_name RSLKey=public
   cicsadd -c pd program_name PathName=program_path
   ```

   where *txn_name* is the name of a transaction, *program_name* is the name of the program, and *program_path* is the path name of the program.
5. Cold start the region, as described in the *CICS Administration Guide for Open Systems*.
6. Set up an X terminal or cicsterm window as the target window for output from the debugger. See "Using the debugging tool integrated with Micro Focus Server Express COBOL on CICS on Open Systems (Animator)" on page 269 for instructions on how to specify a window for displaying debugger output.
7. Start a cicsterm window.
8. Run the CJDB transaction to display the Java debugging window.
9. Enter the name of the device where debugger output is to be displayed.
10. Enter the transaction ID, program name, or terminal that serves as input to the debugger.

Part 5. Appendixes

Appendix. CICS commands used in application programming

This appendix describes the CICS commands used during the preparation of CICS application programs.

cicsmap - generate BMS map files

The **cicsmap** command processes a source file containing Basic Mapping Support (BMS) macros, and generates either a symbolic map or maps, or a physical map or maps, or both as specified by the map input. You use command-line options to control the generation of symbolic or physical maps.

Syntax

cicsmap [-a -r] [-s | -p] [-e] [-f *code_page*] {*file* | *file*.bms} [-x]

Description

The output physical map file, in the current working directory, is *x.map* where *x* is determined by the SUFFIX and TERM BMS macros. CICS overwrites any previous physical map file for the same map source file with the newly generated map.

CICS places the symbolic map file in *mapset.e* for COBOL, *mapset.h* for C or C++, *mapset.inc* for PL/I. CICS produces the symbolic map file in the current working directory. CICS overwrites any previous symbolic map file for the same map set in the current working directory with the newly generated map.

CICS does not generate maps if errors are detected in the map source file. The contents of the map source file determine the operation of the **cicsmap** command. The LANG option associated with the map set macro (DFHMSD) determines the output of **cicsmap**.

cicsmap produces both the symbolic and physical maps by default or you can use the **-p** or **-s** flags to restrict **cicsmap** to producing a single map type.

cicsmap ignores TYPE=DSECT, TYPE=MAP, and TYPE=&SYSPARM options. If you include any other value for TYPE, **cicsmap** produces an error message. See "Migrating maps from your CICS system to other family members" on page 203 for more information.

Errors are written to **stderr**.

Note: For important information about permissions, refer to "Access permissions for maps and transaction programs" on page 37.

Help information is available when an invalid flag or -? is specified.

Options

-a **cicsmap** generates the alternative symbolic map.

-e Affects the interpretation of the hexadecimal data presented in the XINIT option. When specified, **cicsmap** converts the hexadecimal data in the XINIT option from the code set specified by **-f** to the current code set.

-f *code_set*
Code page from which **cicsmap** is to translate XINIT strings. If you do not specify this, CICS assumes the user's locale.

file Name of the input map source file. You can include an optional pathname with this name. The source file must either have the extension **bms**, or have no extension, in which case **cicsmap** appends the suffix **bms**.

-p Generates only a physical map.

-r Generates return codes.

-s Generates only a symbolic (logical) map.

-x Prevents PIC G type COBOL code or GRAPHIC type code from being generated.

If PS=8 is specified in a named DFHMDF BMS macro of a BMS COBOL or PL/I map, the **cicsmap** command generates the PIC G type COBOL code or GRAPHIC type PL/I code. For example, if LENGTH=12 is specified in the field BMS macro, PIC G(6) is generated instead of PIC X(12) in a COBOL map, and GRAPHIC(6) is generated instead of CHARACTER(12) in a PL/I map. However, the PIC G type is possibly unsupported on your version of COBOL compiler, or you may wish not to have PIC G or GRAPHIC types generated. If this is the case, use the **-x** flag to prevent the PIC G or GRAPHIC type code from being generated. (Refer to the documentation for your COBOL compiler to determine if the PIC G type is supported.)

Restrictions

Standard operating system access permissions for files and directories apply.

Returned Values

cicsmap has the following exit values:

0 **cicsmap** successfully processed the source map set.

1 **cicsmap** (default) encountered at least one error which it has written to **stderr**.

4 **cicsmap** (**-r** set) encountered at least one error of moderate severity which it has written to **stderr**.

8 **cicsmap** (**-r** set) encountered at least one error of high severity which it has written to **stderr**.

Examples

1. To run the command from the current directory to obtain a symbolic map:
   ```
   cicsmap -s temp.bms
   ```
2. The **-e** option is required where you have converted maps from EBCDIC, for example from a CICS/MVS system, and the maps contain the XINIT option. So that **cicsmap** produces the correct physical map, you need to specify the **-e** flag.
   ```
   cicsmap -p -e temp2.bms
   ```

cicstran - translates source code

The command language translator converts source code written in a supported language to an equivalent source program in which each CICS command has been converted into a statement for the supporting language.

Syntax

Syntax on CICS on Open Systems:

cicstran [-l {COBOL | IBMCOB | C | IBMCPP} | CPP}] [-q] [-s] [-e] [-c *number*] [-v] [-d] [-g *locale*] *file*

Syntax on CICS for Windows

cicstran [-l {COBOL | IBMCOB | C | IBMC | CPP | IBMCPP} [-q] [-s] [-e][-c *number*] [-v] [-d] [-g *locale*] *file*

Description

To translate EXEC CICS commands, DFHRESP macros, and DFHVALUE macros within an application program into source language statements that interface with the CICS runtime environment, invoke the CICS command language translator **cicstran** from an operating system shell. A C application source file of the name **file.ccs** produces a C source file in the current working directory with the name *file***.c**. A C++ application source file of the name **file.ccs** produces a C++ source file in the current working directory with the name *File***.C.** A COBOL source file is produced in the current working directory, with the name *file***.cbl** from an application source file of the name *file***.ccp**. Errors, warnings, and information messages are written to the operating system file stream **stderr**.

You must invoke the translator process prior to the compilation and link-edit steps when generating program executable code.

You specify options for the command language translation process on the command-line.

For important information about permissions, refer to "Access permissions for maps and transaction programs" on page 37.

Help information is available when an invalid flag or -? is specified.

Options

-c ***number***
: Specifies the number of lines to be included in each page of the translator listing, including heading and blank lines, of the output file created with the **-v** and **-s** flags. *number* must be an integer in the range 7 through 255. If 7 or less, the heading and one line of listing is included on each page. The default is 60.

  Note: This option is null when it is not used with the **-v** or **-s** flags.

-e
: Indicates that CEDF is to be used to debug the program.

Note: If you use this option, the program's performance may be degraded because of the extra activity required to determine the EDF status of the terminal on every EXEC CICS call.

-d Produces code that passes line numbers through CICS to be used by the Execution Diagnostic Facility (CEDF) and to be included in Transaction Dump information.

-g *locale*
Sets the locale the translator is to work in, where *locale* is a string that provides information to certain set conventions in the locale category.

file Name of application source file.

-l Identifies the source language of the program input to the translator:

CICS on Open Systems
'C' for C, 'IBMCPP' or 'CPP' for C++, COBOL for Micro Focus Server Express COBOL, IBMCOB for IBM COBOL (the default is COBOL).

CICS for Windows
'C' for Microsoft C, 'IBMC' for IBM C, 'CPP' for Microsoft C++, 'IBMCPP' for IBM C++, 'COBOL' for Micro Focus Net Express COBOL, and 'IBMCOB' for IBM COBOL

-q identifies COBOL string literal delimiter; APOST (default) or QUOTE.

-s Produces a listing file *file***.lis.**

-v Produces a cross-reference listing of all EXEC CICS commands in *file***.xrf.**

Restrictions

Standard operating system access permissions for files and directories apply.

Examples

1. To translate a C program to utilize CEDF, and produce a cross-reference listing:

```
cicstran -v -l C -e -c 60 Applic2.ccs
```

The interaction between the command level translator and the application programmer is through a set of error codes and conditions output by the translator. These codes are dependent on whether you invoke the translator for COBOL or C.

The translator highlights, on an error report, all EXEC CICS commands found to be in error.

Note: The error report includes line numbers up to 65535. If the line number is greater than 65535, it is erroneously reported as line number 65535.

The EIBLABEL field, used only for COBOL programs, is used to contain values relating to handled conditions or abends. The values are:
-1 RETURN or XCTL
0 normal sequential command processing
1 abend

2 to *nn*
a value identifying a label for a handled condition or abend

Returned Values

cicstran has the following exit values:

0 **cicstran** has successfully translated the application program and has generated a **.c** or **.cbl** file.

1 **cicstran** has detected errors or warnings during translation of the application program, and has written messages to the operating system file stream **stderr**. **cicstran** has generated a **.c** or **.cbl** file.

2 **cicstran** has detected translator errors and has written messages to the operating system file stream stderr.

Refer to Chapter 8, "Translating, compiling, and link-editing CICS application programs," on page 213.

cicstcl - translate, compile, and link

The **cicstcl** command performs the translation, compiles the translated program, and links the resulting object by using the appropriate commands.

Syntax

Syntax on CICS on Open Systems:

cicstcl [-l {COBOL | IBMCOB | C | IBMPLI | IBMCPP | CPP}] [-q] [-s] [-a] [-e] [-c *number*] [-v] [-d] [-t (HP-UX only)] [-g *locale*] [-x *"ext cmd"*] [-X *"ext cmd"*] *file*

Syntax on CICS for Windows:

cicstcl [-l {COBOL | IBMCOB | C | IBMPLI | IBMC | CPP | IBMCPP}] [-q] [-s] [-a] [-e] [-c *number*] [-v] [-d] [-x *"ext cmd"*] [-X {*"ext cmd"* | cicscobinsert}] *file*

Note: The **-X cicscobinsert** option works only on COBOL **.cpp** files to be compiled by using Micro Focus Net Express version 3.0 or higher.

Description

This program accepts all the flags taken by the **cicstran** program, as well as four additional flags, **-a**, **-x**, **-X**, and, on HP-UX systems only, **-t**. You specify compile and link options by using the environment variables as shown in Table 41 on page 214.On C, C++, and COBOL files, the **cicstcl** command runs the **cicstran** program automatically to provide the compile and link directives required by CICS. On IBM PL/I files, the **cicstcl** command calls the PL/I compiler that handles the preprocessing, compiling, and linking of PL/I files.

On CICS on Open Systems, an application program must be contained in a file with a suffix based on file type. The **cicstran** program (run automatically on COBOL, C, and C++ files as part of the **cicstcl** command) translates *file***.ccp** into *file***.cbl** or *file***.ccs** into *file***.c** or *file***.C**. The **cicstcl** command then compiles and links these **.cbl**, **.c**, or **.C** files. (PL/I programs are not translated prior to compilation. On these files, the **cicstcl** command invokes the PL/I compiler, which handles all of the processes.) Table 46 and Table 47 on page 282 summarize the extensions of incoming files and their resulting intermediate files and translation programs.

Table 46. Extensions of incoming files and resulting intermediate files and transaction programs on CICS on Open Systems

File type	Extension of incoming file	Extension of intermediate file (after translation)	Extension of resulting transaction program in working directory
Micro Focus Server Express COBOL	.ccp	.cbl	.gnt or .int
IBM COBOL	.ccp	.cbl	.ibmcob
C	.ccs	.c	None
PL/I	.pli	None	.ibmpli
IBM C++	.ccs	.C	.ibmcpp

Table 46. Extensions of incoming files and resulting intermediate files and transaction programs on CICS on Open Systems (continued)

C++	.ccs	.C	.cpp

Table 47. Extensions of incoming files and resulting intermediate files and transaction programs on CICS for Windows

File type	Extension of incoming file	Extension of intermediate file (after translation)	Extension of resulting transaction program in working directory
Micro Focus Net Express COBOL	.ccp	.cbl	.cbmfnt
IBM COBOL	.ccp	.cbl	.ibmcob
Microsoft C	.ccs	.c	.dll
PL/I	.pli	None	.ibmpli
IBM C	.ccs	.c	.ibmc
Microsoft C++	.ccs	.C	.cpp
IBM C++	.ccs	.C	.ibmcpp

Specifying the **-x** flag causes the **cicstcl** command to run an external command (for example, an SQL translation step) before the **cicstran** translation. Specifying the **-X** flag causes the **cicstcl** command to run an external command or, on the Windows platform, the **cicscobinsert** utility (which compiles EBCDIC-enabled COBOL programs that run on a Windows workstation—see "Compiling EBCDIC-enabled COBOL programs" on page 57 and "Using Micro Focus Net Express to compile EBCDIC-enabled COBOL programs" on page 227 for more information). The pretranslation step specified by the **-x** flag must produce a file with the appropriate **.ccp** or **.ccs** suffix. The posttranslation step specified by the **-X** flag must be able to handle the **.cbl**, **.c**, or **.C** file produced by the translator.

Help information on the **cicstcl** command is written to **stderr** when you run **cicstcl** with an invalid flag or **-?**. If you specify a valid flag that is not supported for the language type, CICS writes a warning to the standard error stream and the option is ignored.

Note: For important information about permissions, refer to "Access permissions for maps and transaction programs" on page 37. The **cicstcl** command accepts multiple source files for IBM VisualAge COBOL for NT.

Options

-a Required for generating debugging information to support compiler debugging products.

-c *number*
(Not PL/I) specifies the number of lines to be included in each page of the translator listing, including heading and blank lines, of the output files created with the **-v** or **-s** flags. The *number* value must be an integer in the range 7 through 255. If the *number* value is 7 or less, the heading and one line of listing is included on each page. The default is 60.

Note: This option must be used with the **-v** or **-s** flags.

-d Produces code that passes line numbers through CICS to be used by the Execution Diagnostic Facility (EDF) and to be included in the Transaction Dump information.

-e Indicates that CEDF is to be used to debug the program.

file Name of application source file.

-g *locale* **(CICS on Open Systems only)**
(Not PL/I) codes page locale in which the translator is to work. If you do not specify **-g**, **cicstcl** uses your locale.

-l Identifies the source language of the program:

CICS on Open Systems
'COBOL' for Micro Focus Server Express COBOL, 'IBMCOB' for IBM COBOL, 'C' for C, 'IBMPLI' for PL/I, 'IBMCPP' for IBM C++, and 'CPP' for C++ (the default is 'COBOL').

CICS for Windows
'COBOL' for Micro Focus Net Express COBOL, 'IBMCOB' for IBM COBOL, 'C' for Microsoft C, 'IBMPLI' for PL/I, 'IBMC' for IBM C, 'CPP' for Microsoft C++, and 'IBMCPP' for IBM C++ (the default is 'COBOL').

-q Identifies "COBOL string literal delimiter"; APOST (default) or QUOTE.

-s (Not PL/I) produces listing files to facilitate debugging. Produces a CICS source listing file, *filename***.lis**, for all languages. For CICS on AIX C and C++ applications, also produces an assembly listing file, *filename***.lst**, and retains the intermediate file, *filename***.c**. For IBM COBOL, also produces an assembly listing file, *filename***.wlist**, and a COBOL source listing and data map list, *filename***.lst**, as well as retaining the intermediate file, *filename***.cbl**.

-t (CICS on HP-UX only)

Generates an **.snt** (shared **.gnt**) Micro Focus Server Express COBOL executable file. Files with the **.snt** extension are larger than their equivalent **.gnt** files but have different characteristics:
- Unlike **.gnt** files, **.snt** files are libraries shared across the system.
- There is a lower memory cost because more processes, such as **cicsas** processes, use the single **.snt** image.
- After the initial load of an **.snt** file, successive loads are faster than when using **.gnt** files.

See the HP documentation for more information. Experiment to achieve the best system performance.

-v (Not PL/I) produces a cross-reference listing of all EXEC CICS commands in *filename***.xrf**.

-X {*ext cmd* **| cicscobinsert} (The cicscobinsert option is only for files on CICS for Windows to be compiled with Micro Focus Net Express 3.0 or higher)**
(Not PL/I) The **-X** *ext cmd* option executes the specified external command for each translated file, *filename***.cbl**, *filename***.c**, or *filename***.C**, after running **cicstran** for the file. The **-X cicscobinsert** option executes the **cicscobinsert**

utility for each translated COBOL file, *filename*.**cbl**, after running **cicstran** for the file. The **-X** ***ext cmd*** command is executed as **ext cmd** *filename*.**cbl**, **ext cmd** *filename*.**c**, or **ext cmd** *filename*.**C**. The **-X cicscobinsert** command is executed as **cicscobinsert** *filename*.**cbl**.

-x ext cmd
(Not PL/I) executes the specified external command for each file specified, *filename*.**ccp** or *filename*.**ccs** , before running **cicstran** for that file. The command is executed as **ext cmd** *filename*.**ccp** or **ext cmd** *filename*.**ccs**.

Restrictions

Standard operating system access permissions for files and directories apply.

The interaction between the command-level translator and the application programmer is through a set of error codes and conditions output by the translator.

All EXEC CICS API commands found to be in error by the translator are highlighted on an error report. You can then amend these and resubmit your source.

The EIBLABEL field, used only for COBOL programs, is used to contain values relating to handled conditions or abends. The values are:

-1 EXEC CICS RETURN or EXEC CICS XCTL
0 normal sequential command processing
1 abend
2 to ***nn***
a value identifying a label for a handled condition or abend

Returned Values

The translation step of **cicstcl** has the following exit values:

0 **cicstcl** has successfully translated the application program and has generated a **.cbl**, **.c**, or **.C** file.

1 **cicstcl** detected errors or warnings during translation of the application program and wrote messages to the standard error stream. **cicstcl** generated a **.cbl**, **.c**, or **.C** file.

2 **cicstcl** has detected translator errors and has written messages to the standard error stream.

Examples

- To compile, translate, and link-edit a C++ application complete with a source listing:

```
cicstcl -l IBMCPP -s Applic2.ccs
```

- To compile, translate, and link-edit an IBM COBOL application with information for debugging from dump:

```
cicstcl -l IBMCOB -d -s Applic3.ccp
```

Refer to Chapter 8, “Translating, compiling, and link-editing CICS application programs,” on page 213.

Bibliography

- *CICS Application Programming Reference*, SC09-4461
- *CICS Intercommunication Guide*, SC09-4462
- *CICS Family: API Structure*, SC33-1007
- *CICS Family: Interproduct Communication*, SC33-0824
- *Concepts and Planning*, SC09-4582
- *TXSeries Release Notes*, GC09-4491
- *CICS Family: Client/Server Programming*, SC33-1435
- *CICS Administration Reference*, SC09-4459
- *CICS Administration Guide*
- *Planning and Installation Guide*
- *CICS Application Programming Guide*, SC09-4460
- *DB2: Administration Guide*,
- *CICS Problem Determination Guide*, SC09-4465
- *CICS Clients: Administration*, SC33-1792
- *IBM 3270 Information Display Programmer's Reference*,
- *CICS Administration Guide for Windows Systems*, SC09-4456
- *SMARTdata UTILITIES for AIX*
- *CICS Messages and Codes*, SC09-4589
- *CICS Administration Reference*
- *CICS Administration Reference*
- *CICS Administration Guide for Open Systems*, SC09-4587

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the document. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
ATTN: Software Licensing
11 Stanwix Street
Pittsburgh, PA 15222
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks and service marks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States, other countries, or both:

AIX	C-ISAM
CICS	CICS/400
CICS/6000®	CICS/ESA
CICS/MVS	CICS/VSE
Database 2™	DB2

DB2 Universal Database™	DFS
Domino™	Encina
IBM	IMS™
Informix	Lotus®
MQSeries	MVS
MVS/ESA	Notes®
OS/2	RACF®
SecureWay	SupportPac™
System/390	TXSeries
VisualAge	VTAM®
WebSphere®	

Domino, Lotus, and LotusScript are trademarks or registered trademarks of Lotus Development Corporation in the United States, other countries, or both.

ActiveX, Microsoft, Visual Basic, Visual C++, Visual J++, Visual Studio, Windows, Windows NT®, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Pentium® is a trademark of Intel™ Corporation in the United States, other countries, or both.

This software contains RSA encryption code.

Other company, product, and service names may be trademarks or service marks of others.

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Printed in USA

SC09-4460-03

Spine information:

TXSeries™ CICS Application Programming Guide Version 5.1 SC09-4460-03