Rocket Model 204

Host Language Interface Programming Guide

Version 7 Release 5.0

September 2014
204–75–HLIPROG-01

Notices

Edition

Publication date: September 2014
Book number: 204–75–HLIPROG-01
Product version: Version 7 Release 5.0

Copyright

Trademarks

Examples

This information might contain examples of data and reports. The examples 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.

License agreement

Note: This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Corporate Information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

Contacting Technical Support

If you have current support and maintenance agreements with Rocket Software and CCA, contact Rocket Software Technical support by email or by telephone:

Email: m204support@rocketsoftware.com

Telephone :

North America	+1.800.755.4222
United Kingdom/Europe	+44 (0) 20 8867 6153

Alternatively, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. You will be prompted to log in with the credentials supplied as part of your product maintenance agreement.

To log in to the Rocket Customer Portal, go to:

www.rocketsoftware.com/support

Contents

II Model 204 Database Processing

5 Model 204 Parameters

6 Model 204 Files and Records

7 Model 204 Fields and Variables

15 HLI Transactions

16 Recovery and Checkpoints

Index

About this Guide

Model 204 provides a functionally complete Host Language Interface (HLI), which enables you to invoke nearly all the system functions from applications written in programming languages such as COBOL, FORTRAN, PL/1, and Assembler.

Using the HLI facility, you can access Model 204 from a host language application and process against the database.

Audience

This guide is directed primarily to the application programmer who is using the HLI facility for the first time. The information about multiple cursor IFSTRT threads is provided for both first-time and experienced HLI application programmers who are using that functionality for the first time.

Rocket Model 204 documentation

To access the Rocket Model 204 documentation, see the Rocket Documentation Library (http://docs.rocketsoftware.com/), or go directly to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/).

A note about User Language and SOUL

Model 204 version 7.5 provides a significantly enhanced, object-oriented, version of User Language called SOUL. All existing User Language programs will continue to work under SOUL, so User Language can be considered to be a subset of SOUL, though the name "User Language" is now deprecated. In this manual, the name "User Language" has been replaced with "SOUL."

Documentation conventions

This guide uses the following standard notation conventions in statement syntax and examples:

Convention	Description
`TABLE`	Uppercase represents a keyword that you must enter exactly as shown.
`TABLE` *`tablename`*	In text, italics are used for variables and for emphasis. In examples, italics denote a variable value that you must supply. In this example, you must supply a value for *tablename*.
`READ [SCREEN]`	Square brackets ([ ]) enclose an optional argument or portion of an argument. In this case, specify READ or READ SCREEN.

Convention	Description
`UNIQUE | PRIMARY KEY`	A vertical bar (|) separates alternative options. In this example, specify either UNIQUE or PRIMARY KEY.
`TRUST | NOTRUST`	Underlining indicates the default. In this example, NOTRUST is the default.
`IS {NOT | LIKE}`	Braces ({ }) indicate that one of the enclosed alternatives is required. In this example, you must specify either IS NOT or IS LIKE.
`item ...`	An ellipsis (. . .) indicates that you can repeat the preceding item.
`item ,...`	An ellipsis preceded by a comma indicates that a comma is required to separate repeated items.
`All other symbols`	In syntax, all other symbols (such as parentheses) are literal syntactic elements and must appear as shown.
`nested-key` `::=` *`column_name`*	A double colon followed by an equal sign indicates an equivalence. In this case, *nested-key* is equivalent to *column_name*.
`Enter your account:` `sales11`	In examples that include both system-supplied and user-entered text, or system prompts and user commands, boldface indicates what you enter. In this example, the system prompts for an account and the user enters **sales11**.
File > Save As	A right angle bracket (>) identifies the sequence of actions that you perform to select a command from a pull-down menu. In this example, select the Save As command from the File menu.
EDIT	Partial bolding indicates a usable abbreviation, such as E for EDIT in this example.

Part I
Basic HLI Processing

This part describes the different ways to use the Host Language Interface. It provides information that is useful for designing and coding HLI applications.

1

HLI Threads

Overview

A thread is a connection to Model 204. An HLI application must start a thread to do processing under Model 204. This chapter describes Model 204 threads for application programmers who are using the Host Language Interface facility.

Read the section “Multiple cursor IFSTRT threads” on page 4 if you are using multiple cursor functionality in your host language program for the first time.

For more information

Refer to Chapter 15 for information about HLI threads and Model 204 transactions. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for examples of HLI applications that use different types of threads and for information about IFAM1, IFAM2, and IFAM4.

Types of HLI threads

A thread is a connection to Model 204. A thread is required to connect a user to Model 204.

The total number of Model 204 threads available to connect users is set by the system administrator. The total number of threads that are available is equal to the number of IODEV statements specified in the Model 204 online run. A host language application must start at least one thread in order to access Model 204 and process against the database.

Refer to the Rocket *Model 204 Host Language Interface Reference Manual* for information about IODEVs required for HLI threads.

The host language application initiates a request to start a connection to Model 204 by issuing a call to IFSTRT (or IFSTRTN) or IFDIAL (or IFDIALN). If a connection is available for the host language program, Model 204 starts the thread.

Once a thread is started, the host language application can issue other HLI calls to Model 204.

IFSTRT and IFDIAL threads

There is a fundamental difference between IFSTRT and IFDIAL threads, based on the underlying call protocols, which provide the host language interface to Model 204. The protocols operate differently and provide different types of host language functionality.

You must know which type of functionality is required for a particular application to determine which type of thread to use. You must code only the calls, specifications, and corresponding program logic that are available for use with the type of thread that you start.

The functionality of IFSTRT and IFDIAL threads is described in the following sections.

Note: In IFAM2 under z/OS or z/VSE, you can elect to start both types of threads in a host language job; however, IFDIAL and IFSTRT threads function independently of one another as separate transactions. Refer to Chapter 15 for a description of HLI threads and Model 204 transactions in IFAM1, IFAM2, and IFAM4.

IFSTRT threads

An IFSTRT thread provides a user interface between a host language application and Model 204 that allows the program to issue calls that perform functions against the database similarly to Model 204 SOUL and Command Language.

For example, the IFOPEN call, which is comparable to the Model 204 OPEN command, can be issued on an IFSTRT thread to open a file for processing. An IFFIND call, which is comparable to the SOUL FIND statement, can be issued to create a found set of records. The IFCLOSE call, which is comparable to the Model 204 CLOSE command, closes a file on an IFSTRT thread.

A number of HLI calls perform retrieval and update functions that are available for use with an IFSTRT thread.

Using an IFSTRT thread, you can run a batch host language application that functions similarly to a SOUL request. The IFSTRT application can issue HLI calls that access the database and process data.

Different types of IFSTRT threads

The following types of IFSTRT threads are available using the HLI facility:

- Multiple cursor IFSTRT thread
- Single cursor IFSTRT thread, with update privileges
- Single cursor IFSTRT thread, with read-only privileges (available only in IFAM2 and IFAM4)

Each IFSTRT thread supports one type of functionality. You must code only the calls, specifications, and corresponding program logic that are valid for the particular type of IFSTRT thread that you start.

For example, the call that opens a cursor, IFOCUR, is valid only on a multiple cursor IFSTRT thread; a particular updating call, IFPUT, is valid only on a single cursor IFSTRT thread with update privileges. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of individual calls.

Comparison of multiple and single cursor IFSTRT threads

There is a basic difference in functionality between a multiple cursor IFSTRT thread, which functions very much like SOUL by allowing access to multiple files and record sets, and a single cursor IFSTRT thread, which limits access to one file and one record set at a time.

Table 1-1 summarizes the differences in access to files and data between multiple cursor and single cursor IFSTRT threads.

Table 1-1. Comparison of multiple cursor and single cursor IFSTRT threads

Item	Multiple cursor thread	Single cursor thread
Number of threads per application	One	• One (IFAM1) • Multiple (IFAM2, IFAM4)
Number of active update units	Only between first update and commit	Continuously
Transaction commit	Explicitly, at user option	Explicitly, at user option IFCMMT or IFCMTR should always be called before an IFDTHRD
Number of open files or groups concurrently per thread	Multiple	One (last one opened)
Current file is...	1. Default 2. Specify (IN FILE)	Last one opened

Table 1-1. Comparison of multiple cursor and single cursor IFSTRT threads (Continued)

Item	Multiple cursor thread	Single cursor thread
Number of found sets per thread	Multiple	One
Found set availability	Retained until explicitly released	Not retained
Retrieval of records or values from sets	Multiple times	Once
Current set is...	Specify (set name)	Last one found
Number of current records per thread	Multiple	One
Current record is...	Specify (cursor)	Next record in current set

Note: Update units must begin and end on the same thread. If using IFDTHRD, assure that any in-progress update unit ends on the current thread by issuing an IFCMMT or IFCMTR on the current thread before the IFDTHRD call.

For host language applications that use IFSTRT threads, Technical Support recommends that you use a multiple cursor IFSTRT thread. See “Advantages of a multiple cursor IFSTRT thread” on page 6.

Multiple cursor IFSTRT threads

A multiple cursor IFSTRT thread supports access to multiple:

- Files or groups, using functions that specify which file or group
- Record or value sets, using functions that specify which set
- Records or values, using functions that specify which record or value (that is, which cursor)

With a multiple cursor IFSTRT thread, the file, set, record, or value that is specified in the HLI call is the one that is current for processing. For calls that use a set or cursor parameter, you must always code the specification to indicate which set, record, or value is current.

For calls that provide an optional file specification, if you do not specify a file, the file that was opened last is current by default.

You can use a multiple cursor IFSTRT thread in an IFAM1, IFAM2, or IFAM4 host language job.

Starting a multiple cursor IFSTRT thread

To start a multiple cursor IFSTRT thread, you must specify a value of 2 for the thread type indicator (THRD-TYP is 2).

For example, the following COBOL statement in an IFAM2 job starts a multiple cursor IFSTRT thread:

```
CALL "IFSTRT" USING RETCODE,LANG-IND,LOGIN,THRD-TYP,THRD-
ID.
```

where the following variables are specified in the WORKING-STORAGE SECTION of the program:

- LOGIN is USERABC;ECP; (login account name is USERABC, and password is ECP)
- LANG-IND is 2 (COBOL)
- THRD-TYP is 2 (a multiple cursor IFSTRT thread)
- THRD-ID is a required output integer variable for the thread identifier

Note: In IFAM1, if you do not specify a thread type, the thread defaults to a single cursor IFSTRT thread (THRD-TYP is 0). Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of the IFSTRT call.

Sample coding sequence

Using a multiple cursor IFSTRT thread, a host language application might perform database operations in the following order:

1. Start the multiple cursor thread (IFSTRT)
2. Open Customer file (IFOPEN)
3. Open Orders file (IFOPEN)
4. In Customers, find all records (IFFIND)
5. Open the cursor to the Customers found set (IFOCUR)
6. Loop until there are no more Customer records:
   - Fetch the customer name (IFFTCH)
   - In Orders, find Order records for this customer (IFFIND)
   - Open a cursor to the Orders found set (IFOCUR)
   - Loop until there are no more Order records:

     Fetch the order name (IFFTCH)

     Print a report line.

     Close the cursor to the Orders found set (IFCCUR)
7. Close the cursor to the Customers found set (IFCCUR)
8. Finish processing (IFFNSH)

Advantages of a multiple cursor IFSTRT thread

Compared to a single cursor IFSTRT thread, a multiple cursor IFSTRT thread provides the more powerful host language interface to Model 204. A multiple cursor IFSTRT thread:

- Always uses a single connection, which allows a more efficient host language program to be coded.
- Activates a transaction only for updating calls allowing for checkpoints to be taken between update units.
- Allows more facile access to the Model 204 database, with access to multiple files or groups, sets, records, and values on a single thread.
- Provides a broader range of database operations with certain calls for list and record processing that are only available using this type of thread.
- Provides a processing environment that is similar to SOUL by allowing a record set to be:
  - Retained until explicitly released, so that, once created, a record set is available for processing until it is released by the host language program.
  - Symbolically referenced using the name of the function that created it, which is analogous to using a statement label in User Language (SOUL).

Using a multiple cursor IFSTRT thread

For an application that requires access to multiple files concurrently, Technical Support recommends that you use a multiple cursor IFSTRT thread and that you do not run a multithreaded HLI job (in IFAM2 or IFAM4).

For an application that requires access to a single file, or to one file at a time, you can use either a multiple cursor IFSTRT thread or a single cursor IFSTRT thread.

Note: If you are performing update processing using one thread, checkpoints are easier on a multiple cursor IFSTRT thread.

Single cursor IFSTRT threads

A single cursor IFSTRT thread allows access to logical database entities, such as files and record sets, in a serial processing mode. Each single cursor IFSTRT thread processes a single file or group at a time and, for the current file or group, can have active one current record set, one current record, one current value set, and one current value at a time.

For each single cursor IFSTRT thread, all processing against one file or group is completed before processing against the next file or group is begun.

Note that, in IFAM2 and IFAM4, a host language application can process records from different files and groups or different sets of records from the same file or group by using a technique called multithreading. See “Multithreaded IFSTRT application” on page 8 for more information about multithreading.

Read-only or update privileges

In IFAM1, a single cursor IFSTRT thread provides update privileges. In IFAM2 and IFAM4, you can specify a single cursor IFSTRT thread with update privileges by specifying a value of 1 for the THRD-TYP parameter.

For example, the following COBOL statement in an IFAM2 application starts a single cursor IFSTRT thread with update privileges:

```
CALL “IFSTRT” USING RETCODE,LANG-IND,LOGIN,THRD-TYP,THRD-
ID.
```

where the following variables are specified in the WORKING-STORAGE SECTION of the program:

- LOGIN is USERABC;ECP; (login account name is USERABC and password is ECP)
- LANG-IND is 2 (COBOL)
- THRD-TYP is 1 (a single cursor IFSTRT thread with update privileges)
- THRD-ID is a required output integer variable for the thread identifier

Note: In IFAM2 and IFAM4 with a read-only IFSTRT thread (THRD-TYP is 0), you cannot issue calls that perform updating functions. You can open files or groups using passwords with update privileges, but no updates are allowed. If a host language program issues an updating call on a read-only IFSTRT thread, Model 204 returns an error code of 40.

Sample coding sequence using a single cursor IFSTRT thread

Using a single cursor IFSTRT thread, a host language application might perform database operations in the following order:

1. Start a standard thread (IFSTRT)
2. Open Customer file (IFOPEN)
3. In Customers, find all records (IFFIND)
4. Loop until there are no more Customer records:
   - Get the customer record (IFGET)
   - Update the customer record (IFPUT)
   - Print a report line

5. Finish processing (IFFNSH)

Multithreaded IFSTRT application

You can use two or more single cursor IFSTRT threads to process data from multiple files and record sets in an IFAM2 or IFAM4 application.

In a multithreaded application, using two or more single cursor IFSTRT threads, a host language program can perform parallel processing of records from different files and groups, or different sets of records from the same file or group.

Each single cursor IFSTRT thread processes one single file or group at a time and, for the current file or group, can have active one current record set, one current record, one current value set, and one current value. The IFSTHRD call switches from one thread to another, thereby establishing the current thread.

Update units must begin and end on the same thread. If using IFDTHRD, assure that any in-progress update unit ends on the current thread by issuing an IFCMMT or IFCMTR on the current thread before the IFDTHRD call.

Refer to Chapter 15 for information about a multithreaded transaction.

Sample coding sequence for a multithreaded IFSTRT application

Assume, for example, that you want to cross-reference a customer file with an accounts receivable file. The host language program uses two single cursor IFSTRT threads, one to process each file.

Using two single cursor IFSTRT threads, a host language application performs database operations in the following order:

1. Start a standard thread (IFSTRT)
2. Open Customer file (IFOPEN)
3. In Customers, find all records (IFFIND)
4. Start a standard thread (IFSTRT)
5. Open Accounts Receivable file (IFOPEN)
6. Switch to the Customer thread (IFSTHRD)
7. Loop until there are no more Customer records:
   - Get the Customer record (IFGET)
   - Save the value of the Customer account number.
   - Switch to the Accounts Receivable thread (IFSTHRD)
   - Find any matching Account Receivable records (IFFIND)
   - Loop until there are no more matching records:

Get the Accounts Receivable record (IFGET)

Update the Accounts Receivable record (IFPUT)

– Switch back to the Customer thread (IFSTHRD)

8. Finish processing (IFFNSH)

Note: To relate multiple records within the same file, you apply the same logic used to process records in separate files.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a sample HLI application that uses multiple IFSTRT threads.

IFDIAL thread line-at-a-time interface

An IFDIAL thread provides a line-at-a-time terminal type interface between a host language application and Model 204. With an IFDIAL thread, a host language program functions as a terminal, transmitting a line of input to Model 204 or receiving a line of output from Model 204.

An IFDIAL thread allows a host language application to issue Model 204 commands, such as LOGIN, LOGWHO, LOGCTL, and MONITOR, or other commands that perform operations against the database, for example, to execute a SOUL request. An IFDIAL thread also allows the host language program to receive messages and data from Model 204.

An IFDIAL thread supports the use of the companion calls, IFREAD and IFWRITE, to communicate with Model 204.

Note: If you are using an IFDIAL thread, you must code the calls, specifications, and corresponding program logic that are valid for use with this thread.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for an example of an HLI application that uses an IFDIAL thread. Also refer to that manual for descriptions of HLI calls.

Starting an IFDIAL thread

To start an IFDIAL thread, you must specify the IFDIAL call in your host language program. For example, the following COBOL statement starts an IFDIAL thread:

```
CALL "IFDIAL" USING RETCODE,LANG-IND.
```

where the following variable is specified in the WORKING-STORAGE SECTION of the program:

```
LANG-IND is 2 (COBOL)
```

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of the IFDIAL call.

Sample coding sequence

Using an IFDIAL thread, a host language application might perform database operations in the following order:

1. Start a Model 204 terminal interface thread (IFDIAL)
2. Log in to Model 204 (IFWRITE)
3. Receive the login message from Model 204(IFREAD)
4. Read the first record from an input file that contains a SOUL request into a program storage area (named, for example, IN-AREA)

   For example, the input file might contain the following records:

```
BEGIN
PRINT 'HELLO'
END
```

5. Send a line of input (BEGIN) to Model 204 (IFWRITE using IN-AREA).
6. Loop until there are no more records in the input file:
   - Read the next record from the input file into the program storage area.
   - Send a line of input to Model 204 (IFWRITE using IN-AREA).
7. Receive a line of output from Model 204 (IFREAD).
8. Finish processing. (IFHNGUP)

Note: If the SOUL request generated multiple lines of output, the IFDIAL application loops using IFREAD to get each line of output.

Checking the IFREAD and IFWRITE return codes

Before issuing IFREAD or IFWRITE, the IFDIAL application must check the Model 204 completion return code from the previous IFREAD or IFWRITE to determine which call is required next by Model 204. For a return code of 1, IFWRITE is required next; for a return code of 2, IFREAD is required next.

The previous sample coding sequence shows just the basic program logic without the necessary return code checking.

See Chapter 3 for more information about checking IFREAD and IFWRITE return codes.

2

IFSTRT Processing

Overview

This chapter describes how to perform basic IFSTRT data processing operations for application programmers who are using the Host Language Interface facility. Read the section “Using cursors on a multiple cursor IFSTRT thread” on page 15, if you are using multiple cursor IFSTRT thread functionality in your HLI application for the first time.

For more information

Refer to Chapter 6 for a description of files and records. Refer to Chapter 7 for a description of fields and variables. Refer to Chapter 8 for more information about specifying find criteria.

Using record sets

A set of records is a group of records found in a file or group that meet the conditions of selection criteria specified in an HLI call. For example:

```
NAME = SMITH

AGE = 60
```

Refer to Chapter 8 for information about specifying find conditions.

Creating a record set

Use the following HLI calls to select records from a file or group and create a record set for processing:

- IFFAC
- IFFIND

- IFFNDX
- IFFWOL

Current record set

The set of records currently being processed is the current set.

On a multiple cursor IFSTRT thread, you must explicitly specify the record set in the HLI call. On a single cursor IFSTRT thread, the current set is the one last created.

The COBOL coding excerpt below shows an IFFIND call on a multiple cursor IFSTRT thread that creates a found set of records (named WODOCS) and a call to IFRELR, which releases the record set when processing against the records is complete. The IFRELR call specifies the WODOCS record set.

```
WORKING-STORAGE SECTION.
01  CALL-PARMS.
    05  RETCODE    PIC 9(5) COMP SYNC.
    05  DOCS-SPEC  PIC X(61) VALUE 'IN FILE PROFS
        FD SEX=FEMALE;OCCUPATION=DOCTOR OR DENTIST;END;'.
    05  DOCS-NAME  PIC X(7) VALUE 'WODOCS;'.
•
•
•

PROCEDURE DIVISION.
•
•
•
* FIND CREATES RECORD SET
*
CALL "IFFIND" USING RETCODE,DOCS-SPEC,DOCS-NAME.
*
* PERFORM USER SUBROUTINE RECORD PROCESSING LOOP
* THEN RELEASE RECORD SET
*
CALL "IFRELR" USING RETCODE,DOCS-NAME.
•
•
•
```

See “Example of cursor processing” on page 17 for an example that illustrates using a cursor to reference a found set. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of HLI calls used to process record sets.

Using value sets

A set of values is a group of unique values of a field found in a file that meet the conditions of selection criteria in an HLI call. For example, for a field named AGENT, the following data values may be found stored in the file: SMITH, JONES, and GREEN.

Refer to Chapter 8 for information about specifying find conditions.

Creating a value set

Use the IFFDV call to select field values from a file and create a value set for processing.

You can create value sets using only fields with the FRV (For Each Value) or the ORDERED attribute.

Refer to Chapter 7 for information about field attributes.

Current value set

The set of values currently being processed is the current set.

On a multiple cursor IFSTRT thread, you must explicitly specify the value set in the HLI call. On a single cursor IFSTRT thread, the current set is the one last created.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of HLI calls used to process value sets.

Using lists

A list is a user-defined entity that holds copies of a found set of records for processing.

List processing is similar to set processing. However, placing records in a found set on a list allows the enqueue locks to be freed so that records can be accessed by other users. Also, you can modify the record set by adding additional records to the list or removing records from the list.

Creating a list

Use the following HLI calls to place records from a found set on a list for processing:

- IFPROLS, on a multiple cursor IFSTRT thread
- IFLIST, on a single cursor IFSTRT thread

Use IFPROL to add a record to a list, or IFRRFL to remove a record from a list. On a multiple cursor IFSTRT thread, use IFCLST to clear a list and IFRRFLS to remove records from a list.

Refer to Chapter 9 for information about record locking. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of IFPROLS, IFLIST, IFPROL, and IFRRFL calls used to process lists.

Example of list processing

The following COBOL coding excerpt calls IFPROLS to place records from a found set (named WODOCS) that contains female doctors and dentists on a list (called INCOME).

Subsequent calls to IFOCUR and IFFTCH allow the application to process records on the list. For example, a record processing loop might examine a record for INCOME values and report data in different annual income ranges.

When record processing is finished, IFRRFL removes the records from list INCOME.

```
WORKING-STORAGE SECTION.
01  CALL-PARMS.
    05  RETCODE    PIC 9(5) COMP SYNC.
01  FIND-PARMS.
    05  FIND-NAME  PIC X(7) VALUE 'WODOCS;'.
    05  DOCS-SPEC  PIC X(44) VALUE
        'SEX=FEMALE;OCCUPATION=DOCTOR OR DENTIST;END;'.
01  LIST-PARMS.
    05  LIST-SPEC  PIC X(24) VALUE
        'IN WODOCS ON LIST INCOME;'.
    05  LIST-NAME  PIC X(7) VALUE 'INCOME;'.
01  CURSOR-PARMS.
    05  CURSOR-SPEC      PIC X(38) VALUE
        'ON LIST INCOME IN ORDER BY INCOME AMT;'.
    05  CURSOR-NAME      PIC X(7) VALUE 'DOCINC;'.
01  FETCH-PARMS.
    05  DIRECTION        PIC 9 COMP SYNC VALUE '1'.
    05  EDIT-SPEC        PIC X(44) VALUE
        'EDIT (SSN,NAME,INCOME AMT)(A(9),A(30),J(8));'.
01  WORK-REC.
    05  WORK-SSN         PIC 9(9).
    05  WORK-NAME        PIC X(30).
    05  WORK-INCOME-AMT  PIC 9(8).
    •
    •
    •
PROCEDURE DIVISION.
•
•
•
* FIND CREATES RECORD SET
*
CALL "IFFIND" USING RETCODE,FIND-SPEC,FIND-NAME.
```

```
CALL "IFPROLS" USING RETCODE,LIST-SPEC.
CALL "IFOCUR" USING RETCODE,CURSOR-SPEC,CURSOR-NAME.
*
* PERFORM USER SUBROUTINE LOOP TO PROCESS RECORDS IN LIST
* INCLUDES CALLS SHOWN BELOW TO IFFTCH AND IFRRFL
* TO REMOVE RECORD FROM LIST WHEN FINISHED PROCESSING
*
CALL "IFFTCH" USING RETCODE,WORK-REC,DIRECTION,
     CURSOR-NAME,EDIT-SPEC.
•
•
•
CALL "IFRRFL" USING RETCODE,LIST-NAME,CURSOR-NAME.
•
•
•
```

Using cursors on a multiple cursor IFSTRT thread

A cursor is a user-defined entity that identifies an existing record or value set that has been named on a multiple cursor IFSTRT thread.

Opening and closing a cursor

Two basic functions are required to manipulate each cursor on a multiple cursor IFSTRT thread:

- IFOCUR specifies a cursor name and opens the cursor to a set that has been previously established by the successful execution of one of the following calls using the Compiled IFAM feature:
  - IFFAC
  - IFFDV
  - IFFIND
  - IFFNDX
  - IFFWOL
  - IFSORT
  - IFSRTV

  You can open more than one cursor against the same named set to maintain different positions within the set; and you can open several cursors against several different record sets. You can also open a cursor on a list.

  You can also establish a cursor by using IFFRN or IFSTOR and then reference that cursor by using the name of the saved compilation.
- IFCCUR closes the named cursor and indicates that processing against the cursor is complete.

Naming a cursor

The following guidelines apply for naming cursors:

- Name must be unique.
- Specify the cursor name as a short character string; the maximum length is 32 characters.
- Cursor name must begin with a letter (A-Z or a-z), which can be followed by one or more occurrences of:
  - Letter (A-Z or a-z)
  - Digit (0-9)
  - Period (.)
  - Underscore (_)
- Avoid using a SOUL keyword for a cursor name. If in another specification you refer to a cursor name that is a keyword, Model 204 might incorrectly interpret the name.

Cursor processing

Positioning a cursor

In order to process records from a found set or a list on a multiple cursor IFSTRT thread, you must first open a cursor by issuing a call to IFOCUR that specifies the found set or list. Then use IFFTCH to advance to the next record.

Once you use IFFTCH to position the cursor, issue any of the following single record function calls:

- IFDALL
- IFDREC
- IFDVAL
- IFOCC
- IFPROL
- IFRRFL
- IFUPDT

If you need to obtain the internal database number of the record in the current cursor, use IFRNUM.

The IFFRN and IFSTOR functions implicitly allocate and open a cursor. The current record is the record whose number is specified in IFFRN, or the record just stored by IFSTOR. You can then manipulate the record using one of the single record function calls listed above.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of the IFOCUR, IFFTCH, IFRNUM, IFFRN, IFSTOR, and the single record level IFSTRT thread calls, and for coding examples that process against the database using a multiple cursor IFSTRT thread.

Example of cursor processing

The following COBOL coding excerpt opens a cursor to a found set (named FDFORD), which contains FORD records from a CARS file.

The application processes records inside a record processing loop (UPDATE-LOOP) using IFFTCH and performs different actions depending on the year. If the record is 1980 or later, the application updates the record by deleting the color blue. The application also deletes any CARS record that is older than (that is, year is earlier than) 1980.

Note: All calls in the records processing loop reference the found set using the cursor name (CRFORD).

```
WORKING-STORAGE SECTION.
01  CALL-PARMS.
    05  RETCODE       PIC 9(5) COMP SYNC.
01  FIND-PARMS.
    05  FIND-NAME     PIC X(7) VALUE 'FDFORD;'.
    05  DOCS-SPEC     PIC X(30) VALUE
        'IN FILE CARS FD MAKE=FORD;END;'.
    05  COUNT         PIC 9(5) COMP SYNC.
01  CURSOR-PARMS.
    05  CURSOR-SPEC   PIC X(10) VALUE 'IN FDFORD;'.
    05  CURSOR-NAME   PIC X(7) VALUE 'CRFORD;'.
01  FETCH-UPDATE-PARMS.
    05  DIRECTION     PIC 9 COMP SYNC VALUE '1'.
    05  EDIT-SPEC     PIC X(53) VALUE
        'EDIT(MAKE,MODEL,YEAR,COLOR)(A(15),A(15),J(4),A(10));'.
    05  COMP-NAME     PIC X(7) VALUE 'EDFORD;'.
01  WORK-REC.
    05  WORK-MAKE     PIC X(15).
    05  WORK-MODEL    PIC X(15).
    05  WORK-YEAR     PIC 9(4).
    05  WORK-COLOR    PIC X(10).
    •
    •
    •
01  FIELDS-PARMS.
    05  FIELD-LIST    PIC X(6) VALUE 'COLOR;'.
    05  DFIELD-VALUE  PIC X(5) VALUE 'BLUE;'.
    05  DFIELD-NAME   PIC X(6) VALUE 'COLOR;'.
    05  FIELD-COUNT   PIC X(8).
```

```
PROCEDURE DIVISION.
•
•
•
* FIND CREATES RECORD SET
*
CALL "IFFAC" USING RETCODE,DOCS-SPEC,COUNT,FIND-NAME.
MOVE COUNT TO TOT-RECS.
PRINT 'TOTAL NUMBER OF FORD CARS IS ' TOT-RECS.
*
* OPEN CURSOR TO FOUND SET AND DO PROCESSING LOOP
*
CALL "IFOCUR" USING RETCODE,CURSOR-SPEC,CURSOR-NAME.
PERFORM UPDATE-LOOP UNTIL TOT-RECS IS EQUAL TO ZERO.
•
•
•

*
* UPDATE-LOOP SUBROUTINE TO PROCESS RECORDS IN CURSOR
* FETCH A RECORD,
* IF 1980 OR LATER, COUNT OCCURRENCES OF COLOR
* IF ONE OR MORE, THEN DELETE VALUE OF BLUE AND UPDATE REC
* ELSE, IF EARLIER THAN 1980, DELETE THE RECORD
*
UPDATE-LOOP.
  CALL "IFFTCH" USING RETCODE,WORK-REC,DIRECTION,
    CURSOR-NAME,EDIT-SPEC.
  IF WORK-YEAR IS GT 1980 THEN
    CALL "IFOCC" USING RETCODE,FIELD-COUNT,CURSOR-NAME,
      FIELD-LIST.
    MOVE FIELD-COUNT TO TOT-FIELDS.
    IF TOT-FIELDS IS GT ZERO THEN
      CALL "IFDVAL" USING RETCODE,DFIELD-NAME,DFIELD-VALUE,
        CURSOR-NAME.
  ELSE
     CALL "IFDREC" USING RETCODE,CURSOR-NAME.
  SUBTRACT 1 FROM TOT-RECS.

•
•
•
```

Using the compiled IFAM facility

The Compiled Inverted File Access Method (IFAM) facility allows certain functions executed on IFSTRT threads to be compiled and stored. You can execute a compilation at a later time by specifying the name under which it was stored. You do not need to recompile the stored call.

Advantage of using Compiled IFAM calls

In the standard HLI implementation (without Compiled IFAM), Model 204 handles each call separately, looking up field names in the data dictionary, and parsing character parameter strings for each execution of a call.

Using the Compiled IFAM facility, you can request that Model 204 perform the initial parsing and dictionary reference once and then refer to the stored information in later calls.

Using the Compiled IFAM facility reduces the amount of CPU time and disk I/O that Model 204 uses to satisfy program calls.

Stored compilations and server tables

Model 204 stores compilations of HLI calls in the server tables: NTBL, QTBL, VTBL, and STBL. Refer to Chapter 12 for information about the Model 204 server tables and HLI calls.

You can use the IFFLUSH call to delete compilations that are no longer needed from these tables to make room for new compilations. IFFLUSH functions differently on standard and multiple cursor IFSTRT threads. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of IFFLUSH.

Certain calls, when issued on a multiple cursor IFSTRT thread, require that you use Compiled IFAM. You must specify a compilation name for the following HLI calls on a multiple cursor IFSTRT thread:

- Find functions: IFFAC, IFFDV, IFFIND, IFFNDX, IFFWOL
- Sort functions: IFSKEY, IFSORT, IFSRTV

Each of these calls establishes a found set and IFOCUR uses the compilation name of the previously compiled call to reference the named set. For example, IFOCUR might open a cursor to a found set (named FDFORD) that was established by a previously compiled IFFIND call.

You must also specify a compilation name for IFFRN, but this name is not referenced by IFOCUR. It is referenced by subsequent single record functions.

Compilation name parameter

A name parameter is required in all Compiled IFAM calls.

The name parameter specifies a character string that is used to identify the compilation. For example, you might specify a compilation name of ORDER1 for an IFFIND call as shown in the following example:

```
CALL "IFFIND" USING STATUS-IND, ORDER-SPEC, ORDER1.
```

where:

- STATUS-IND is an integer variable for the Model 204 return code.
- ORDER-SPEC is a find specification that creates a found set of records using the ORDERS file.
- ORDER1 is the name that uniquely identifies the IFFIND compilation.

The Compiled IFAM form of IFFIND differs from the standard form of the call, which does not include the compilation name parameter. For example, on a single cursor IFSTRT thread you can specify the IFFIND call without using the Compiled IFAM facility, as shown in the following example:

```
CALL "IFFIND" USING STATUS-IND, ORDER-SPEC.
```

See Table 2-1 on page 21 for a list of HLI calls that can be used with the Compiled IFAM facility.

Naming a compilation

The following guidelines apply for compilation names on any type of IFSTRT thread:

- The compilation name is a required input parameter for all Compiled IFAM calls. A null name string is the same as an omitted parameter.
- The name must be unique.
- Specify the compilation name as a short character string; the maximum length is 32 characters.

In addition, on a multiple cursor IFSTRT thread, the following guidelines apply for compilation names:

- Compilation name must begin with a letter (A-Z or a-z), which can be followed by one or more occurrences of:
  - Letter (A-Z or a-z)
  - Digit (0-9)
  - Period (.)
  - Underscore (_)
- Avoid using a SOUL keyword for a compilation name. If in another specification you refer to a compilation name that is a keyword, Model 204 might incorrectly interpret the name.

In addition, on a single cursor IFSTRT thread, any characters except the following are valid in the compilation name:

- Blank space
- Comma (,)
- Left parenthesis (()

- Right parenthesis ())
- Equal sign (=)
- Semicolon (;)

Three forms of Compiled IFAM calls

Three forms of IFSTRT thread calls are available using the Compiled IFAM facility. These calls function in different ways. The following options are available to accommodate different programming styles:

- You can use a single call that compiles and executes with the name parameter that identifies the compilation. When the call executes, Model 204 saves the compiled version of the call.

  For example, when the following IFFIND call is executed, Model 204 stores the compilation as ORDER1:

  ```
  CALL “IFFIND” USING STATUS-IND,ORDER-SPEC,ORDER1.
  ```

  When the same IFFIND call is executed again or when another IFFIND call containing the same name parameter (ORDER1) is executed, Model 204 ignores the find specification (ORDER-SPEC) and uses the stored compilation without requiring recompilation.

- You can use two calls, one is compile-only (HLI call with C suffix) and one is execute-only (HLI call with E suffix), with the name parameter that identifies the compilation for the two phases of Compiled IFAM processing: compilation and execution.

  This option involves a two-call procedure, useful in loop processing. Use the compilation form of the call outside the loop to compile (but not execute) the call specification. For example, you might issue the compilation-only form of IFFIND as shown in the following example:

  ```
  CALL “IFFINDC” USING STATUS-IND,ORDER-SPEC,ORDER1.
  ```

  Within the loop, issue the execution form of the call, thereby executing the previously compiled call. For example, you might issue the execute-only form of IFFIND as shown in the following example:

  ```
  CALL “IFFINDE” USING STATUS-IND,ORDER1.
  ```

Table 2-1 lists the HLI calls that are used with the Compiled IFAM facility. An asterisk (*) indicates that you must use the compiled form of the call with a multiple cursor IFSTRT thread.

Table 2-1. Compiled IFAM calls

Compute and execute	Compile-only	Execute-only
IFCTO	IFCTOC	IFCTOE
IFFAC*	IFFACC	IFFACE

Table 2-1. Compiled IFAM calls (Continued)

Compute and execute	Compile-only	Execute-only
IFFDV*	IFFDVC	IFFDVE
IFFIND*	IFFINDC	IFFINDE
IFFNDX*	IFFNDXC	IFFNDXE
IFFRN*	IFFRNC	IFFRNE
IFFTCH	IFFTCHC	IFFTCHE
IFFWOL*	IFFWOLC	IFFWOLE
IFGET	IFGETC	IFGETE
IFGETV	IFGTVC	IFGTVE
IFGETX	IFGETC	IFGETXE
IFMORE	IFMOREC	IFMOREE
IFMOREX	IFMOREC	IFMORXE
IFOCC	IFOCCC	IFOCCE
IFFOCUR*	IFOCURC	IFOCURE
IFPUT	IFPUTC	IFPUTE
IFSKEY*	IFSKYC	IFSKYE
IFSORT*	IFSRTC	IFSRTE
IFSRTV*	IFSTVC	IFSTVE
IFSTOR	IFSTRC	IFSTRE
IFUPDT	IFUPDTC	IFUPDTE

Sharing a compilation

Some calls that use the Compiled IFAM facility can share the specifications that Model 204 compiles for other functions.

The following calls can share precompiled specifications:

- IFGET, IFMORE, and IFPUT

  When an IFGET, IFMORE, or IFPUT call is compiled, Model 204 does not save the data area address of the HLI application program. The application program can manipulate its buffers and data freely without losing any of the benefits of Compiled IFAM.

- IFFAC and IFFIND
- IFUPDT with a precompiled IFFTCH

Example of a shared compilation

An example of a shared compilation on a multiple cursor IFSTRT thread is an IFUPDTE (execute-only) call that updates the current record using data previously returned by IFFTCH.

In this example, using Compiled IFAM and specifying certain of the same parameter values as the previously compiled IFFTCH, IFUPDTE does the following:

- Points to the same buffer area. (Both calls use WORK-REC, which contains the data fields defined in working storage.)
- References the same cursor. (In this example, the cursor name is NAMERECS.)
- Uses an identical edit specification, which describes the format of the data in the buffer.

IFUPDTE executes using the name of the previously compiled IFFTCH call. In this example, the IFFTCH compilation name is CUSTNAME.

The following COBOL coding excerpt shows the IFUPDTE and IFFTCH shared compilation. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for descriptions of individual HLI calls.

```
WORKING-STORAGE SECTION.
01  WORK-REC.
    05  WORK-SSN     PIC 9(9).
    05  WORK-NAME    PIC X(30).
•
•
•
01  CALL-PARMS.
    05  RETCODE      PIC 9(5) COMP SYNC.
    05  CURSOR-NAME  PIC X(9) VALUE 'CUSTNAME;'.
    05  EDIT-SPEC    PIC X(28) VALUE
        'EDIT (SSN,NAME)(A(9),A(30));'.
    05  COMP-NAME    PIC X(9) VALUE 'NAMERECS;'.
    05  DIRECTION    PIC 9 COMP SYNC VALUE '1'.
•
•
•
PROCEDURE DIVISION.
•
•
•
CALL "IFFTCH" USING RETCODE,WORK-REC,DIRECTION,CURSOR-NAME,
      EDIT-SPEC,COMP-NAME.
*
* PERFORM UPDATE OPERATION MOVE SPACES TO WORK-NAME
*
```

```
CALL "IFUPDTE" USING RETCODE,WORK-REC,COMP-NAME.
```
•
•
•

Using variables with precompiled specifications

Use %variables and field name variables (that is, %%variables) to make small but regular changes to precompiled specifications. You can assign values for %variables by including the variable buffer and variable specification parameters in HLI calls.

See Chapter 7 for more information about %variables.

3

IFDIAL Processing

Overview

This chapter describes basic processing using an IFDIAL thread for application programmers who are using the Host Language Interface facility.

For more information

Refer to Chapter 1 for more information about IFDIAL threads.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for information about HLI jobs using IFDIAL threads and descriptions of HLI calls that are available using an IFDIAL thread.

Terminal type interface

IFDIAL thread

An IFDIAL thread provides a line-at-a-time terminal emulation type interface between a host language application and Model 204. With an IFDIAL thread, a host language program functions as a terminal, transmitting a line of input to Model 204 and receiving a line of output from Model 204.

An IFDIAL thread supports the use of the companion calls, IFREAD and IFWRITE, to communicate with Model 204. Use an IFREAD call to transmit data (a line of output) from Model 204 to your application program and an IFWRITE call to transmit data (a line of input) back to Model 204.

Using an IFDIAL thread, you can pass Model 204 commands, ad hoc or stored SOUL procedures, or Application Subsystem Management applications from an HLI application (batch) program to Model 204. You can also extract data from Model 204, use the data in your program, and put the results into your Model 204 database.

Note: If you are using an IFDIAL thread, you must code the calls, specifications, and corresponding program logic that are valid for use with this thread.

Refer to Chapter 1 for more information about IFDIAL threads. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of HLI calls.

Communication between Model 204 and an IFDIAL thread

The underlying communications mechanism for a terminal type interface drives IFDIAL thread functioning. A ***parent-child*** protocol directs the interchange between the partner programs, that is, between Model 204 (the parent) and the IFDIAL thread (the child).

In all communications between IFDIAL and Model 204, Model 204 is the parent and directs the interchange. The IFDIAL thread is the child. If the Model 204 side wants input, the IFDIAL thread is required to send input.

Also, an IFDIAL thread must process all output from Model 204 that normally goes to a terminal, such as character prompts for logons and passwords or warning and broadcast messages.

Sample call sequences

Establishing an IFDIAL connection

The HLI application initiates the request to establish an IFDIAL connection to Model 204. To start an IFDIAL thread, the HLI application program must specify the IFDIAL call.

For example, a host language application might issue calls in the following order to establish an IFDIAL connection to Model 204:

1. IFDIAL to start an IFDIAL thread.
2. IFREAD to receive the Model 204 response.
3. IFWRITE to log in to Model 204.
4. IFREAD to receive the Model 204 response.
5. IFWRITE to supply a password.
6. IFREAD to receive the Model 204 response:

```
•
• perform IFDIAL processing
•
```

To disconnect the IFDIAL thread and finish processing, the HLI application issues an IFHNGUP call. The following sections show sample call sequences once the IFDIAL connection is established.

Submitting a SOUL request

Once the connection is established, you can use an IFDIAL thread to submit an ad hoc SOUL request to Model 204.

For example, to submit a SOUL request after an IFDIAL connection is established (see the previous section), an HLI application might read an input file containing a SOUL request into a program storage area, and then issue HLI calls in the following order:

-
- *establish the IFDIAL connection*
-

1. IFWRITE inside a program loop, to issue the SOUL statements until all statements are sent to Model 204 (by referencing the program storage area).
2. IFREAD to receive the Model 204 response.

   The program code might loop, issuing the IFREAD to read each line of output from Model 204 until there is no more output in the storage buffer.

Note: This sample shows the basic IFWRITE and IFREAD sequence for submitting a SOUL request. Always code your IFDIAL application to check the completion return code for these calls. See “Checking IFWRITE and IFREAD return codes” on page 28 for information about checking the return codes for IFREAD and IFWRITE.

Invoking a stored SOUL procedure

Once the connection is established, you can use an IFDIAL thread to invoke a stored SOUL procedure.

For example, to invoke a stored SOUL procedure after an IFDIAL connection is established, an HLI application might issue calls in the following order:

-
- *establish the IFDIAL connection*
-

1. IFWRITE to open a file.
2. IFREAD to receive the Model 204 response.
3. IFWRITE to supply a password, if required.
4. IFREAD to receive open file messages from Model 204.

5. IFWRITE to issue the command to include the stored SOUL procedure.

Note: Always code your IFDIAL application to check the completion return codes for IFWRITE and IFREAD. See “Checking IFWRITE and IFREAD return codes” on page 28 for information about checking these return codes.

Using the Model 204 Application Subsystem facility

Once the connection is established, you can use an IFDIAL thread to access the Model 204 Application Subsystem facility.

For example, to use the Model 204 Application Subsystem facility after an IFDIAL connection is established (see page 26) an HLI application might issue calls in the following order:

-
- *establish the IFDIAL connection*
-

1. IFWRITE to invoke an Application Subsystem (APSY).

2. IFREAD to receive the results of the APSY execution.

Note: Make sure to use an Application Subsystem that is designed to run on a line-at-a-time IFDIAL thread. Also, always code your

Note: IFDIAL application to check the completion return codes for IFWRITE and IFREAD, as described in “Checking IFWRITE and IFREAD return codes”.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Application_Subsystem_development) for information about the Application Subsystem facility.

Checking IFWRITE and IFREAD return codes

Always code your IFDIAL application to check the Model 204 completion return code from the previous IFWRITE or IFREAD call before issuing the next IFWRITE or IFREAD.

The IFDIAL application must issue IFREAD or IFWRITE depending on which call Model 204 expects next. Model 204 indicates which of these calls is required next by returning a particular completion code from the previous IFREAD or IFWRITE.

If the Model 204 completion return code from the previous IFREAD or IFWRITE equals 1, the IFDIAL application must issue an IFWRITE. Or, if the return code equals 2, the application must issue an IFREAD.

This is necessary to ensure proper communication with Model 204 during IFDIAL processing.

In the sample coding sequence on page 27 (and also in Chapter 1) the IFDIAL application loops to send lines of input to Model 204 by issuing multiple calls to IFWRITE.

This loop must be controlled by checking the value of the return code each time before the next IFWRITE call is issued. For example, the following COBOL statement is coded:

```
PERFORM WRITE-LOOP UNTIL RETCODE = 2 OR INRECS = 0.
```

where INRECS is a counter that is decremented for each record sent from the input file, and when INRECS equals zero, all SOUL statements have been sent to Model 204.

If during processing Model 204 encounters an error in one of the SOUL statements from the input file, the IFDIAL application must receive the error message from Model 204 by issuing IFREAD before issuing another call to IFWRITE.

Any number of other types of messages might be issued by Model 204, for example, messages from the operator, that might interrupt an IFWRITE processing loop and must be received by the IFDIAL application.

Also, the same type of checking must be done for an IFREAD processing loop. For example, a loop that issues IFREAD to read each line of output from Model 204 until there is no more output in the storage buffer must be controlled by checking the return code for a value of 1, which indicates that an IFWRITE call is required next.

See Figure 3-2 on page 32 for an example of return code checking in a COBOL program that uses IFDIAL processing.

Using stored procedures

An HLI call in a host language program that invokes another program to perform a specific function, or functions, is called a stored procedure call. The stored procedure can be a SOUL procedure or a set of Model 204 commands.

As briefly outlined on page 27, you can use an IFDIAL thread for stored procedure calls.

Operations against the database

You can use a stored procedure with an IFDIAL thread to perform different types of operations against the database, depending on the needs of the HLI application.

Although you can use IFDIAL applications with stored procedures for any type of terminal activity, the following examples are the most common:

- Sending and receiving Model 204 images, which are defined in the program storage area of the HLI application.

- Issuing Model 204 commands, for any Model 204 command that can be issued at a terminal such as MONITOR, and receiving output generated by Model 204.
- Transferring procedures in and out of Model 204 files, which allows you to maintain SOUL procedures in external software configuration management systems.

The following sections describe IFDIAL processing using stored SOUL procedures with images.

Using stored procedures for image processing

Sending and receiving Model 204 images

Because SOUL allows Model 204 images to be read or written to a terminal, an HLI application using an IFDIAL thread that operates in terminal emulation mode can send and receive Model 204 images.

A Model 204 image is a SOUL feature that allows a request to read and process terminal input or input from sequential files. An image describes the format of an external record.

SOUL statements can refer to each item described in the image definition. Using the images facility, you can open a file, read records to the image, write records to a terminal or to a file, and close the file. This capability allows an application to write multiple output files and reports based on a single pass of the database.

The syntax for reading and writing images with SOUL is as follows:

```
OPEN {TERMINAL | %VAR} FOR {INPUT [OUTPUT]
     | OUTPUT [INPUT] | INOUT}

READ [IMAGE] imagename FROM {TERMINAL | %VAR}

[PROMPT {'string' | %VAR}]

WRITE [IMAGE] imagename ON {TERMINAL | %VAR}

CLOSE {TERMINAL | %VAR}
```

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Images) for more information about images.

Example of a stored procedure used to process images

The SOUL (User Language) example in Figure 3-1 is a stored procedure, named IFDIAL-WRITE, which finds records and writes images to the IFDIAL application in Figure 3-2.

Figure 3-1. User Language stored procedure example

```
*
* OPEN MODEL 204 DATA FILE
*
OPEN VEHICLES
*
BEGIN
* USER LANGUAGE PROGRAM TO SHOW USE
* OF IFDIAL AND IMAGES
*
IMAGE VEHICLES.LIST
     VL.BODY IS STRING LEN 4
     VL.COLOR IS STRING LEN 8
     VL.MAKE IS STRING LEN 10
     VL.MODEL IS STRING LEN 20
     VL.YEAR IS PACKED DIGITS 2
END IMAGE
*
* OPEN PATH TO IFDIAL PROGRAM
*
OPEN TERMINAL FOR OUTPUT
PREPARE IMAGE VEHICLES.LIST
*
* FIND AND WRITE IMAGES TO IFDIAL PROGRAM
*
FOUND_SET: FIND ALL RECORDS FOR WHICH MAKE=FORD AND
COLOR=BLUE
           END FIND
     LOOP: FOR EACH RECORD IN FOUND_SET
*
          %VEHICLES.LIST:VL.BODY = BODY
          %VEHICLES.LIST:VL.COLOR = COLOR
          %VEHICLES.LIST:VL.MAKE = MAKE
          %VEHICLES.LIST:VL.MODEL = MODEL
          %VEHICLES.LIST:VL.YEAR = YEAR
*
          WRITE IMAGE VEHICLES.LIST ON TERMINAL
*
     END FOR
*
END
```

Example of IFDIAL application that processes images

The COBOL example in Figure 3-2 performs IFDIAL communication to Model 204 using the stored procedure, IFDIAL-WRITE, shown in Figure 3-1. See “Using a special purpose subroutine” on page 35 for the CVTFLAG subroutine.

Figure 3-2. Sample COBOL program using a stored procedure

```
************************************************************* THIS
IS A SAMPLE COBOL PROGRAM WHICH DOES
* IFDIAL COMMUNICATION TO M204.
*************************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.
        IFDIALUL.
*
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
        SELECT REPORT-FILE ASSIGN TO UT-S-REPORT.
        SELECT INPUT-FILE ASSIGN TO UT-S-INPUT.
*
 DATA DIVISION.
 FILE SECTION.
 FD REPORT-FILE
        LABEL RECORDS ARE OMITTED
        BLOCK CONTAINS 0 RECORDS
        DATA RECORD IS OUT-BUFFER.
 01 OUT-BUFFER                  PIC X(133).
*
 WORKING-STORAGE SECTION.
 01 ERROR-FUNCTION              PIC X(8) VALUE SPACES.
 01 DISPLAY-STATUS-IND          PIC 9(5) VALUE ZERO.
 01 WS-OUTPUT-REPORT-LINE.
    05 WS-CCTL-CHAR             PIC X VALUE SPACES.
    05 MAKE                     PIC X(10).
    05 FILLER                   PIC X(5) VALUE SPACES.
    05 MODEL                    PIC X(20).
    05 FILLER                   PIC X(5) VALUE SPACES.
    05 BODY                     PIC X(4).
    05 FILLER                   PIC X(5) VALUE SPACES.
    05 YEAR                     PIC X(2).
    05 FILLER                   PIC X(5) VALUE SPACES.
    05 COLOR                    PIC X(8).
*
 01 OUTPUT-AREA                 PIC X(256) VALUE SPACES.
*
 01 INPUT-AREA                  PIC X(256).
 01 FORMATED-INPUT-AREA REDEFI NES INPUT-AREA.
```

```
    05 BODY                     PIC X(4).
    05 COLOR                    PIC X(8).
    05 MAKE                     PIC X(10).
    05 MODEL                    PIC X(20).
    05 YEAR                     PIC X(2).
*
 01 INTEGER-CALL-ARGS COMP SYNC.
    05 STATUS-IND               PIC 9(5) VALUE 99.
    05 WRITE-STAT               PIC 9(5).
    05 READ-STAT                PIC 9(5).
    05 LANGUAGE-IND             PIC 9(5) VALUE 2.
*
 01 IFREAD-FLAGS.
    05 IFREAD-MSG-LENGTH        PIC 9999 COMP SYNC.
    05 IFDIAL-ERROR-MSG         PIC X VALUE "N".
       88 ERROR-MSG VALUE "Y".
    05 IFDIAL-NEW-PAGE          PIC X VALUE "N".
       88 NEW-PAGE VALUE "Y".
    05 IFDIAL-USER-RESTARTED    PIC X VALUE "N".
       88 USER-RESTARTED VALUE "Y".
    05 IFDIAL-PASSWORD-REQUEST  PIC X VALUE "N".
       88 PASSWORD-REQUEST VALUE "Y".
    05 IFDIAL-READ-REQUEST      PIC X VALUE “N”.
       88 READ-REQUEST VALUE "Y".
    05 IFDIAL-INFO-MSG          PIC X VALUE "N".
       88 INFO-MSG VALUE "Y".
*
 01 STRING-CALL-ARGS.
    05 INPUT-FLAGS              PIC X.
    05 M204-ERR-MESSAGE         PIC X(80) VALUE SPACES.
    05 VMCF-CHANNEL             PIC X(8) VALUE "M204VMIO".
    05 LOGON-MSG                PIC X(16) VALUE "LOGON SUPERKLUGE".
    05 LOGON-PASSWORD           PIC X(8) VALUE "PIGFLOUR".
    05 REQUEST-NAME                PIC X(14) VALUE "I IFDIAL-WRITE".

 PROCEDURE DIVISION.
*
 MAIN-ROUTINE.
*
 INITIALIZATION.
    OPEN OUTPUT REPORT-FILE
    CALL "IFDIALN"
        USING STATUS-IND, LANGUAGE-IND, VMCF-CHANNEL.
    DISPLAY "IFDIALN STATUS ", STATUS-IND.
    IF STATUS-IND IS NOT EQUAL ZERO
        MOVE "IFDIALN " TO ERROR-FUNCTION
        PERFORM ERROR-ROUTINE.
*
* LOGON-PROCESS
*
```

```
    MOVE LOGON-MSG TO OUTPUT-AREA.
    CALL "IFWRITE" USING WRITE-STAT, OUTPUT-AREA.
    DISPLAY "IFWRITE STATUS ", WRITE-STAT.
    IF WRITE-STAT = 2
       PERFORM READ-LINE THRU READ-LINE-EXIT
         UNTIL READ-STAT = 1 OR READ-STAT = 12.
       IF PASSWORD-REQUEST
         DISPLAY "PASSWORD REQUEST ", IFDIAL-PASSWORD-REQUEST.
         MOVE SPACES TO OUTPUT-AREA
         MOVE LOGON-PASSWORD TO OUTPUT-AREA
         CALL "IFWRITE" USING WRITE-STAT, OUTPUT-AREA
         PERFORM READ-LINE THRU READ-LINE-EXIT
           UNTIL READ-STAT = 1 OR READ-STAT = 12.
*
* MAIN-LOOP
*       START USER LANGUAGE TRANSACTION TO RETRIEVE
*       RECORDS.
*       IFWRITE USES REQUEST-NAME TO EXECUTE
*       STORED USER LANGUAGE PROCEDURE IFDIAL-WRITE
*       AT NEXT INPUT REQUEST, TERMINATE.
*
        MOVE SPACES TO OUTPUT-AREA.
        MOVE REQUEST-NAME TO OUTPUT-AREA.
        CALL "IFWRITE" USING WRITE-STAT, OUTPUT-AREA.
        PERFORM MAIN-LOOP UNTIL READ-STAT NOT = 2
        GO TO TERMINATION.
*
 MAIN-LOOP.
    PERFORM READ-LINE THRU READ-LINE-EXIT.
    MOVE CORRESPONDING FORMATED-INPUT-AREA TO
          WS-OUTPUT-REPORT-LINE.
   WRITE OUT-BUFFER FROM WS-OUTPUT-REPORT-LINE.

              *
              *  READ-LINE CALLS CVTFLAG (CONVERT FLAGS)
              *  TO PROCESS MODEL 204 OUTPUT FROM IFREAD
              *
               READ-LINE.
                  MOVE SPACES TO INPUT-AREA.
                  CALL "IFREAD" USING READ-STAT, INPUT-AREA, INPUT-
             FLAGS.
                  IF READ-STAT = 100 GO TO TERMINATION.
                  DISPLAY "IFREAD STATUS ", READ-STAT, " DATA ", INPUT-
             AREA.
                  CALL "CVTFLAG" USING INPUT-FLAGS, IFREAD-FLAGS.
                  MOVE ZERO TO INPUT-FLAGS.
                  IF USER-RESTARTED
                      GO TO TERMINATION.
                  IF ERROR-MSG OR INFO-MSG
                      GO TO READ-LINE.
```

```
 READ-LINE-EXIT.
    EXIT.
*
 ERROR-ROUTINE.
    MOVE STATUS-IND TO DISPLAY-STATUS-IND.
    DISPLAY "CRITICAL ERROR ENCOUNTERED WITH FUNCTION: "
        ERROR-FUNCTION ", WITH A RETURN CODE OF: "
        DISPLAY-STATUS-IND.
    MOVE SPACES TO ERROR-FUNCTION.
    MOVE SPACES TO M204-ERR-MESSAGE.
    IF ERROR-FUNCTION NOT EQUAL "IFHNGUP" THEN
        PERFORM TERMINATION.
*
 TERMINATION.
    CLOSE REPORT-FILE.
    CALL "IFHNGUP" USING STATUS-IND.
    IF STATUS-IND NOT EQUAL 0 THEN
        MOVE "IFHNGUP" TO ERROR-FUNCTION
        PERFORM ERROR-ROUTINE.
    STOP RUN.
```

Using a special purpose subroutine

Sample subroutine to convert IFREAD flags

IFREAD returns a message descriptor flag as a series of bits. Each bit indicates the type of data received, the password prompt, error messages, and so on. The CVTFLAG (convert flags) subroutine example in Figure 3-3 translates the IFREAD flags to COBOL character strings and refers to them with Level 88 statements.

See Figure 3-3 on page 35 for a sample COBOL program that calls this subroutine after issuing the IFREAD call. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of the IFREAD call.

Figure 3-3. Sample assembly language subroutine to convert IFREAD flags

```
CVTFLAG CSECT 0
        ENTRY CVTFLAG
*
* SAMPLE ASSEMBLY LANGUAGE PROGRAM FOR TRANSLATING THE
* MESSAGE DESCRIPTOR FIELD RETURNED AS THE THIRD ARGUMENT
* OF THE IFREAD CALL.
*
* THIS MODULE EXPECTS TWO PARAMETERS. THE FIRST IS THE
* THIRD PARAMETER RETURNED BY THE IFREAD CALL, THE MESSAGE
* DESCRIPTOR FIELD. THIS FIELD SHOULD BE DEFINED AS:
*   01 MSG-DESC-FIELD PICTURE 9(8) COMP SYNC.
*
```

```
* THE SECOND PARAMETER IS AN 01 LEVEL WORKING STORAGE AREA
* DEFINED AS FOLLOWS:
*
*   01 IFREAD-FLAGS.
*      05 IFREAD-MSG-LENGTH        PICTURE 9999 COMP SYNC.
*      05 IFDIAL-ERROR-MSG         PICTURE X  VALUE 'N'.
*      88 ERROR-MSG                           VALUE 'Y'
*      05 IFDIAL-NEW-PAGE          PICTURE X  VALUE 'N'.
*      88 NEW-PAGE                            VALUE 'Y'
*      05 IFDIAL-USER-RESTARTED    PICTURE X  VALUE 'N'.
*      88 USER-RESTARTED                      VALUE 'Y'
*      05 IFDIAL-PASSWORD-REQUEST  PICTURE X  VALUE 'N'.
*      88 PASSWORD-REQUEST                    VALUE 'Y'
*      05 IFDIAL-READ-REQUEST      PICTURE X  VALUE 'N'.
*      88 READ-REQUEST                        VALUE 'Y'
*      05 IFDIAL-INFO-MSG          PICTURE X  VALUE 'N'.
*      88 INFO-MSG                           VALUE 'Y'

* THE CVTFLAGS SUBROUTINE MOVES THE FIRST HALF WORD OF THE
* MESSAGE DESCRIPTOR FIELD MINUS 4 TO THE FIELD DESCRIBED
* ABOVE AS 'IFREAD-MSG-LENGTH'. THIS IS THE TRUE LENGTH
* OF THE DATA RECEIVED. THE SUBROUTINE THEN EXAMINES THE
* FLAGS IN THE THIRD BYTE OF THE MESSAGE DESCRIPTOR FIELD
* AND SET THE CORRESPONDING FLAGS IN THE COBOL WORKING
* STORAGE AREA TO 'Y'. EACH TIME THE SUBROUTINE IS CALLED
* ALL THE FLAGS ARE SET TO 'N' THEN THE APPROPRIATE FLAGS
* ARE SET TO 'Y'. THE WORKING STORAGE FIELDS MUST BE
ARRANGED
* AS DESCRIBED ABOVE. THE LEVEL 88 FIELDS ARE OPTIONAL.
*
* NOTE: THIS SUBROUTINE IS AN EXAMPLE ONLY AND IS NOT SUP-
PORTED
*       BY COMPUTER CORPORATION OF AMERICA.
*

**********
         REGEQU                 SET REGISTER SYMBOLS
         STM  R14,R12,12(R13)   SAVE REGISTER
         LR   12,15             SET BASE REGISTER
        USING CVTFLAG,R12       ESTABLISH ADDRESSING
         LR   R11,R13           SET UP SAVE AREA LINKAGE
         LA   R13,SAVEAREA
         ST   R11,4(R13)
         ST   R13,8(R11)
*
*        PROGRAM LOGIC STARTS HERE
*
* ADDRESS PARAMETERS
         L    R2,0(R1)       R2, INPUT PARAMETER ADDRESS
         L    R3,4(R1)       R3, OUTPUT PARAMETER LIST
```

```
        USING WSPARM,R3      ADDRESSING TO WS PARM AREA
* RETURN LENGTH TO CALLER
         L    R4,0(R2)       R4, PARAMETER WORD
         SRL  R4,16          SHIFT 16 BITS LEFT, LEAVING
LENGTH
        SH   R4,=Y(4)       DECREMENT LENGTH BY 4 ACTUAL
LEN.
         STH  R4,LENGTH      MOVE ACTUAL LENGTH TO USER
* RETURN FLAGS TO CALLER
         MVI  EMSG,C'N'      CLEAR ALL ERROR MESSAGES
        MVC  EMSG+1(FLAGLEN-1),EMSG PROPAGATE 'N' TO ALL
FLAGS
         TM   2(R2),X'80'    CLASS E MSG ?
         BZ   *+4+4          NO, NEXT TEST
         MVI  EMSG,C'Y'      YES, TURN ON FLAG
         TM   2(R2),X'40'    NEW PAGE ?
         BZ   *+4+4          NO, NEXT TEST
         MVI  NPAGE,C'Y'     YES, TURN ON FLAG
         TM   2(R2),X'20'    RESTARTED ?
         BZ   *+4+4          NO, NEXT TEST
         MVI  RESTART,C'Y'   YES, TURN ON FLAG
         TM   2(R2),X'10'    PASSWORD ?
         BZ   *+4+4          NO, NEXT TEST
         MVI  PASSWORD,C'Y'  YES, TURN ON FLAG
         TM   2(R2),X'08'    READ PROMPT ?
         BZ   *+4+4          NO, NEXT TEST
         MVI  READ,C'Y'      YES, TURN ON FLAG
         TM   2(R2),X'04'    CLASS I MESSAGE
         BZ   *+4+4 NO,      ALL DONE
         MVI  INFO,C'Y'      YES, TURN ON FLAG
*
* RETURN TO CALLER
*
         L    R13,SAVEAREA+4 RESTORE CALLERS REGISTERS
         LM   R14,R12,12(R13)
         SR   R15,R15        CLEAR R15 JUST FOR GOOD FORM
         BR   14             RETURN TO CALLER
        DROP R12,R3 END ADDRESSING TO BASE, WSPARM
         LTORG

SAVEAREA DS            18F'0'
WSPARM   DSECT         MAP OF COBOL WORKING STORAGE RET
ARGS.
LENGTH   DS H'0'       LENGTH OF MESSAGE
EMSG     DS X          ERROR CLASS OF MSG
NPAGE    DS X          NEW PAGE INDICATOR
RESTART  DS X          USER RESTARTED
PASSWORD DS X          PASSWORD PROMPT
READ     DS X          READ PROMPT
INFO     DS X          I CLASS MESSAGE
```

```
FLAGLEN   EQU *-EMSG   NUMBER OF 1 BYTE FLAGS
          END
```

Coding guidelines for IFDIAL applications

Designing your IFDIAL application

Design your HLI application program and SOUL procedure to work together.

For example, your HLI application reads a list of accounting numbers to generate a report. The HLI application sends an account number to the SOUL procedure. After the SOUL procedure receives the account record and performs data manipulation, Model 204 returns the data to the HLI program for final reporting.

When the HLI program sends an account number, the SOUL procedure must be expecting it. Conversely, when the SOUL procedure sends a completed record, the HLI program must be expecting it. The two programs must be synchronous so that the data is usable.

To minimize programming time and effort and to maximize processing efficiency, write a general purpose IFDIAL application that uses basic IFREAD and IFWRITE logic but does not perform data operations.

Design your generic IFDIAL application so that it can be used with a variety of SOUL procedures. Allow the SOUL procedure to do the work of manipulating data and formatting reports.

Note that Model 204 provides a BATCH2 utility that allows you to run a SOUL procedure without having to code an IFDIAL application. Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Program_communications_facilities#BATCH2_facility) for information about the BATCH2 utility.

Checking the Model 204 completion return code

Always check the Model 204 completion return code from the IFREAD or IFWRITE call previously issued to ensure that the next call issued by the HLI application is the one that is expected by Model 204.

A return code of 1 from IFREAD or IFWRITE requires an IFWRITE next; or, a return code of 2 requires an IFREAD next.

Writing special purpose subroutines

Write special purpose subroutines to simplify IFDIAL logic, such as for login or user restart.

Formatting data

The format of the data to be exchanged in your HLI application and SOUL programs must agree.

Sending and receiving Model 204 images

When sending and receiving Model 204 images, be sure that the working storage definition and the image correspond.

To send images from your SOUL procedure to your IFDIAL host language program, use the following statement:

```
WRITE IMAGE imagename TO TERMINAL
```

To transmit data from your IFDIAL host language program to your SOUL procedure, use the following statement:

```
READ IMAGE imagename FROM TERMINAL
```

See “Sending and receiving Model 204 images” on page 39 for more information about images.

Handling terminal messages and prompt strings

Because IFDIAL uses a terminal-type interface, you are responsible for messages and prompt strings.

Encapsulate IFREAD and IFWRITE calls in your subroutines or paragraphs to filter out messages and error codes and return only the expected data to the application.

Note: IFREAD returns message descriptor flags as a series of bits. Each bit indicates the type of data received, the password prompt, error messages, and so on.

The CVTFLAG (convert flags) subroutine example starting on page 35 translates the IFREAD flags to COBOL character strings and refers to them with Level 88 statements.

Use IFATTN to activate ON attention

As with terminal applications, you can use the ON attention SOUL function. The IFATTN call activates ON attention. This attention interrupt is useful to escape out of any processing loop that two programs may engage in.

Using stored procedure calls

When using stored procedure calls, if your HLI application has many distinct functions, construct a separate SOUL procedure to handle each function.

For example, if your HLI program adds, deletes, and modifies records based on a transaction file, invoke a separate SOUL procedure for each function.

Using an application subsystem

When using stored procedure calls, if your HIL program invokes many SOUL procedures, install the procedures as Application Subsystem (APSY) to minimize overload.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Application_Subsystem_development) for information about the Application Subsystem facility.

4

Using Completion Return Codes

Overview

This chapter briefly describes the Model 204 completion return codes for application programmers who are using the Host Language Interface facility.

For more information

Refer to Chapter 13 for a description of the CCAJRNL, CCAJLOG, and CCAAUDIT data sets. Refer to Chapter 16 for information about using the Model 204 audit trail and journals.

Refer to Chapter 10 for coding examples that test the completion return code after HLI call.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of HLI calls and a complete listing and description of the Model 204 completion return codes.

Refer to Chapter 10 for coding examples of how to handle enqueue errors using completion return codes. Refer to Chapter 3 for coding examples that show how to check completion return codes for an IFDIAL application.

Using completion return codes for HLI calls

The first parameter of most HLI calls is a binary integer completion code that Model 204 returns to the application program. The completion code indicates whether or not the call finished successfully.

In general, completion codes of less than four indicate normal operation of a call. Completion codes equal to or greater than four usually indicate that the call was not successful and that Model 204 logged a message in the journal.

Note, however, that some return codes greater than 4 indicate normal completion. For example, for IFOPEN, a return code of 16 indicates that a file has been recovered but than processing can proceed without further conditional testing.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a complete listing and description of completion return codes.

Checking the completion return code

Check the completion code after every HLI call. Unless severe errors are detected, Model 204 does not abend an HLI application program that attempts unsuccessfully to execute a call.

If the completion code indicates an unsuccessful operation, the HLI application program can perform one of the following actions:

- Reissue the call
- Stop processing
- Bypass the call and continue processing

However, use the last option with care. For example, the IFSTRT call, which establishes the connection to Model 204, must complete successfully for HLI processing to continue.

The HLI program must check the return code from IFSTRT, and, if the call did not complete successfully (that is, the return code is not 0), the HLI program must either reissue IFSTRT until it is successful or stop processing. If the HLI program attempts to issue another HLI call after an unsuccessful call to IFSTRT, Model 204 abends the job.

Using the audit trail

Technical Support recommends that you run your HLI program with an audit trail, because this, together with the completion return codes for individual HLI calls, is an important debugging aid.

To get HLI audit information, make sure to set the following parameters in your HLI job:

- SYSOPT parameter setting to include RK lines
- LAUDIT parameter with proper setting for logical input lines

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/List_of_Model_204_parameters) for a description of SYSOPT and LAUDIT parameters.

Note: On an IFSTRT thread, the last message generated by the system is available to the application program with the IFGERR call. This message is available whether or not a journal is used in the HLI job.

Part II
Model 204 Database Processing

This part describes Model 204 database structures and processing. It gives the HLI user necessary background information about processing against the Model 204 database.

5

Model 204 Parameters

Overview

This chapter provides summary information about Model 204 parameters for application programmers who are using the Host Language Interface facility.

For more information

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/List_of_Model_204_parameters#List_of_parameters) for more information about the use and specifications of all Model 204 parameters.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for more information about IFAM1, IFAM2, and IFAM4 jobs.

Model 204 Parameters

Parameters are variables that control or describe the Model 204 system.

The HLI application program passes to Model 204 the information pertinent to each HLI call, such as what file to open or what records to count, in a parameter list with the call. Model 204 passes information back to the application program, such as the count, as output parameters.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of HLI calls and their particular parameter lists.

All parameters have default values. You can set parameters to different values to tailor system performance to a particular installation's or user's requirements.

On an IFSTRT thread, you can retrieve the current value of any parameter using the IFEPRM call. You can modify the value of certain parameters using the IFRPRM and IFUTBL calls.

On an IFDIAL thread, you can use these Model 204 commands: VIEW, RESET, and UTABLE.

The basic types of Model 204 parameters are listed in Table 5-1.

Table 5-1. Types of Model 204 parameters

Parameter type	Function
System	Controls the operation of Model 204 as a whole.
User environment control	Controls the operation of a particular user's terminal or the characteristics of the system responses toward that user.
File	Affects the organization or structures of Model 204 files.

The following sections describe these types of parameters.

System parameters

System parameters control the operation of the Model 204 system as a whole.

System parameter values affect such characteristics as the frequency of checkpoints, the maximum number of files and groups that can be opened concurrently, the size of the buffer pool, the size and number of user work areas, and the maximum length of HLI call parameters.

For example, the following parameters control the maximum length of HLI call parameters and are critical for proper HLI processing:

Parameter	Meaning	Function	Default
LIBUFF	Input buffer length	Specifies the maximum length allowed for string values passed to Model 204 in HLI call parameters.	255 bytes
LOBUFF	Logical line output buffer length	Specifies the length of the logical line output buffer for output parameters returned by Model 204 to the HLI program.	256 bytes

Other system parameters, such as LAUDIT, SYSOPT, and IFAMBS, are also important for HLI processing.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/List_of_Model_204_parameters#List_of_parameters) for a description of the LIBUFF, LOBUFF, LAUDIT, SYSOPT, and IFAMBS system parameters.

Setting system parameters for an HLI job

System parameters that can be set for an HLI job are set on the EXEC statement of the Model 204 region or on a parameter line read at system initialization.

In an IFAM1 job, you can specify system parameters in the IFSTRT call or in the IFSETUP call for an IFDIAL thread. In an IFAM4 job, you can specify system parameters in the EXEC statement in the job setup.

Note: After the Model 204 region has been initialized, most system parameters cannot be reset.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/List_of_Model_204_parameters) for information about the system parameters, and in particular, those that affect using the Host Language Interface:

http://m204wiki.rocketsoftware.com/index.php/Using_HLI_and_batch_configurations

User environment control parameters

User environment control parameters (that is, User 0 parameters) affect the operation of a particular user's terminal or the characteristics of system responses toward that user. For example, the access method and device type are required parameter specifications for a user. Each user must be defined to Model 204.

In an IFAM1 job, specify the user parameters in the IFSTRT call. In an IFAM4 job, specify the user parameters in the CCAIN input file in the job setup. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of IFAM1 and IFAM4 jobs.

Refer to the Rocket Model 204 documentation wiki for information about user environment control parameters:

http://m204wiki.rocketsoftware.com/index.php/Defining_the_user_environment_(CCAIN)#User_environment_control_parameters

Buffer size and IODEV parameters for an IFDIAL thread

The maximum length of a data area that can be transferred over an IFDIAL thread is 32763 bytes. The maximum length affects application developers who use the IFDIAL thread as well as Model 204 system administrators who must set up the size parameters.

The system administrator of a Model 204 installation can tune the size parameters so that the best performance versus memory ratio can be achieved. Model 204 determines the size of buffers in the User 0 MAX(OUTMRL,INMRL) parameter.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for detailed information about transferring data on an IFDIAL thread and related size considerations.

There are performance considerations in implementing the communication buffer for an IFDIAL thread by setting User 0 parameters for IODEV=29 (in z/OS and z/VSE) or IODEV=39 (in z/VM). For more information, refer to the Rocket Model 204 documentation wiki:

http://m204wiki.rocketsoftware.com/index.php/Defining_the_user_environment_(CCAIN)#User_environment_control_parameters

User table parameters

User table parameters affect the size of the tables in a user's work area.

Model 204 holds an area of memory known as server area, which contains the information needed to describe the operation of an individual user. The server is divided into tables, each of which contains a specific type of data.

The user table parameters determine the size of these tables. The size of user tables can be reset with the IFUTBL call.

Refer to Chapter 12 for more information about user work areas and table parameters.

File parameters

File parameters affect the organization, structure, and allocation of Model 204 files.

Some file parameters are set when a file is created. The Model 204 file manager determines these parameters, which usually cannot be changed after the file is created.

In addition to file parameters set during creation, Model 204 maintains a set of parameters that reflect the changing condition of each file. These parameters contain such information as the number of records added and deleted since the file was created, the space used and free space available in the data and index areas, and the number of records in the sorted file overflow areas.

With Model 204 file manager privileges, you can use IFRPRM on an IFSTRT thread to set the following file parameters:

- FISTAT
- FOPT
- FRCVOPT
- PRIVDEF
- OPENCTL

- BRESERVE
- DRESERVE

For more information about file parameters, refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/FPARMS_and_TABLES_file_parameters).

CURFILE parameter and the current file

You can also use the IFEPRM call on an IFSTRT thread to get the name of the file last accessed, which is held by the CURFILE parameter.

See Chapter 6 for more information about the CURFILE parameter.

6

Model 204 Files and Records

Overview

This chapter describes basic Model 204 database structures, that is, files and records, for application programmers who are using the Host Language Interface facility.

For more information

See Chapter 7 for information about Model 204 fields and variables. Refer to Chapter 2 for information about HLI processing using Model 204 files and records.

For more information about files and records, refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/File_design and http://m204wiki.rocketsoftware.com/index.php/Introduction_to_User_Language).

Data files

A Model 204 file is a collection of as many as 16.7 million records. The maximum number of files that can be accessed in one Model 204 job is 32,767.

Model 204 files have an inverted file structure, which means that they contain, in addition to the data records themselves, an index that points to records containing particular values of fields designated as retrieval or KEY fields. The index and data areas are physically separate parts of the file.

The data representation of an inverted file format is similar to the organization of a reference book such as *Bartlett's Familiar Quotations*. The main body of the book, that is, the data area, consists of groups of quotations. To help you find the quotations, the book provides an index that contains subject entries for

as many subjects as are pertinent to the quotation, interspersed with title and author entries for a poem, speech, or other work.

The following Model 204 file types determine how data is stored:

- Entry order
- Unordered
- Sorted
- Hashed key

You can use the four file types in any combination to make up a database. Each type of organization provides the inverted file capability, and any Model 204 file can be cross-referenced to any other Model 204 file.

The following sections describe the different types of data files.

Entry order file

In an entry order file, Model 204 stores each new record in the next available space. During processing, Model 204 retrieves and processes the records in file storage order. Usually, the order is chronological.

Note: Entry order files are the most widely used Model 204 files. Entry order files provide all inverted file capabilities except sort and hashed key. To generate sorted output when processing entry order files, use the IFSORT call.

Unordered file

In an unordered file, Model 204 stores new records in any location where enough space is available to hold the records. Model 204 maintains for an unordered file a queue of pages with available space. When records are deleted, storage space becomes available. Model 204 uses pages from the queue to store new records.

When using unordered files, note that these files utilize disk space more efficiently if you delete records frequently in your HLI application.

Sorted file

When you specify the sorted file option and designate one field in a file as the sort key, the file is a sorted file. Model 204 uses an indexed sequential scheme for a sorted file, storing and processing records according to the values of the sort key.

Specify the sorted file option and designate the sort key field at file initialization. You can designate the sort key field as either mandatory or optional. When a sort key field is mandatory, Model 204 requires all records to contain this field. When the sort key field is optional, Model 204 accepts records in the file that do not contain the field.

You can retrieve records in a sorted file with a combination of special sort key retrieval conditions and the standard key field value conditions. Sorted files are useful for applications that produce many reports in the same sort sequence.

Hashed key file

When you specify the hashed key option and designate a particular field name as the hash key, the file is a hashed key file. Specify the hashed key file option and designate the hash key field at file initialization. Model 204 stores records in locations based on the value of the designated hash key. You can designate the hash key field as either mandatory or optional.

For example, you might designate SOCIAL SECURITY NUMBER as the hash key. You can then retrieve records on the basis of this key without performing an index search.

Note: Use a hashed key file if all data records contain a unique or fairly unique key and if your HLI application performs most retrievals on the basis of that key alone.

File groups

A file group is a named collection of Model 204 files that Model 204 treats logically as a single file.

For example, you might define the file group BANK containing the files CHECKING, SAVINGS, and CARLOAN. Each file is accessible directly and is called by its own name (CHECKING, SAVINGS, and CARLOAN). All data in the three files is also available under the name BANK. Model 204 does not duplicate the data. Instead, a special table relates the group BANK to its member files.

You can define any number of file groups. File groups can have overlapping membership.

For example, nine individual state files can be processed as members of a file group, SOUTH, which represents a particular regional area. The same files can also be processed as members of a file group, USA, which represents the entire United States.

File groups are particularly useful for processing under the following conditions:

- Regrouping and archiving files in data aging applications
- Providing alias names for test and production files
- Processing similar data that must be maintained in independent files

File model options

The following file models allow you to enforce filewide constraints on files and fields by setting the FILEMODL parameter when creating a file:

- The NUMERIC VALIDATION file model, which causes Model 204 to perform numeric data type validation on fields defined as FLOAT or BINARY.
- The First-Normal Form (1NF) file model, which ensures that the data within a file conforms to the rules for 1NF.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/FILEMODL_parameter) for more information on the FILEMODL parameter.

Records

A record is a combination of related data fields that are defined in a Model 204 file. Records are variable in length and format. Model 204 does not restrict records to any predefined or fixed structure.

A Model 204 record can contain a virtually limitless number of fields. Also, the structure of a Model 204 record does not limit the number of occurrences of a field in a record. Fields that appear more than once in a record are known as multiply occurring fields.

To save space under some conditions, a Model 204 file manager can preallocate space for occurrences of certain fields in a record. If fields are preallocated, only the specified number of occurrences can be added to a record. If fields are not preallocated, no space is reserved until data is added to the record. Within a record, fields can appear in any sequence. Within a file, the sequence and number of fields might vary from record to record.

Application programs retrieve data from Model 204 files strictly on the basis of individual field names and values. Therefore, applications are independent of the physical structure of the files and are not affected by changes or additions to the database.

Internal database record number

Model 204 uniquely identifies every record in the database by its file name and its internal record number. Each record in a file remains associated with a unique record number until the file is reloaded or until the record is deleted and its number reused by a new record.

Current record and the current file

During HLI job processing, Model 204 maintains information that identifies the current record, using the CURREC parameter, and the current file, using the CURFILE parameter.

No current record

On a single cursor IFSTRT thread, certain HLI calls set CURREC to -1, which is an invalid internal database record number, to indicate that there is no current record on the thread. Upon completion, the following HLI calls set CURREC
to -1:

- IFDREC
- IFDSET
- IFINIT
- IFOPEN

Current record on a multiple cursor IFSTRT thread

On a multiple cursor IFSTRT thread, Model 204 assigns a value to CURREC, which is the internal database record number of the current record in the open cursor that was referenced in the call just processed.

Note that a cursor has no current record associated with it until one of the following HLI calls executes successfully and positions the cursor:

- IFFTCH
- IFFRN
- IFSTOR

Once the open cursor is positioned at the current record, the HLI application can perform individual record processing functions. Any of the individual record HLI calls allowed on a multiple cursor IFSTRT thread can process the current record by referencing the name of the open cursor in the call.

Current record on a single cursor IFSTRT thread

On a single cursor IFSTRT thread, the CURREC value is the internal database number of the current record relative to the last set created for the last file opened on the current thread.

The current record is the one last referenced by IFGET, IFPOINT, or IFBREC.

Specifying a record number

Use the following calls to specify a particular record number:

- IFFRN, specifying the cursor name, on a multiple cursor IFSTRT thread.

  Note: IFFRN opens a cursor to the current record.

- IFPOINT on a single cursor IFSTRT thread.

The record specified in the HLI call becomes the current record for processing.

7

Model 204 Fields and Variables

Overview

This chapter describes two basic Model 204 database constructs, fields and variables, for application programmers who are using the Host Language Interface facility.

For more information

Refer to Chapter 6 for information about Model 204 files and records. Refer to Chapter 7 for information about using fields and variables in HLI calls.

Refer to *Rocket Model 204 Host Language Interface Reference Manual* for a description of HLI calls.

For complete information about Model 204 fields and variables, refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Field_design and http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation).

Field names and values

The smallest addressable element of the Model 204 system is the field. To identify and retrieve data, assign meaningful names, such as AGE, ADDRESS, OCCUPATION, and SEX, to fields. Each Model 204 file can have as many as 4000 different field names defined.

Model 204 stores each field as a name=value pair, for example, SEX=FEMALE and SEX=MALE. Field name=value pairs are variable in length to provide for file compression and text processing capabilities.

The structure of a Model 204 record does not limit the number of occurrences of a field in a record. Fields that appear more than once in a record (for

example, CHILD or HOBBY fields) are known as multiply occurring fields. When referring to a field that is multiply occurring, specify whether, for example, the first, fifth, or all occurrences of the field are being addressed.

Rules for naming fields

The following rules apply for naming Model 204 fields:

- A field name must begin with a letter, and can be up to 255 characters in length.
- Except for the following characters, you can use any word or character (including a space) as part of a field name:

  ```
  ?? ?$ ?& @ # ; :
  ```

Note: The at sign (@) and number sign (#) characters are the default terminal correction characters. If you specify some other characters in the ERASE and FLUSH parameters, the restriction on using @ and # in a field name applies instead to the symbols that you specify.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/List_of_Model_204_parameters) for information about the ERASE and FLUSH parameters.

- When more than one consecutive space appears in a field name, Model 204 ignores the extra spaces.
- Any reserved word or operator can be:
  - Part of an unquoted string, as long as it is not surrounded by spaces. For example, ZCOUNT is valid; Z COUNT is invalid.
  - Part of a quoted string, as long as it does not stand alone. For example, A 'OR' B is valid; 'OR' is invalid.

For more information about quotation marks, see "Using quotation marks" on page 62.

The following terms have special meaning in the Model 204 system, and are reserved. To avoid errors, do not use reserved terms when forming field names and values.

AFTER	ALL	AND	AT
BEFORE	BY	COUNT	EACH
EDIT	END	FROM	IN
IS	LIKE	NOR	NOT
OCC	OCCURRENCE	OR	RECORD

RECORDS	TAB	THEN	TO
VALUE	VALUES	WHERE	WITH
EQ GE	GT LE	LT	NE

- Any of the following reserved characters, if embedded in a field name, must be part of a string enclosed in quotation marks. Avoid the following characters when forming field names and values:

$	*	/	#	@	=
<	>	;	:	,	ÿ
+ (plus sign)		- (minus sign)		. . . (period in series)	
(	)				

Examples of valid field names

The following examples are valid Model 204 field names:

```
A345

ANNUAL.%INTEREST

A' = B'
```

Examples of invalid field names

The following examples are invalid Model 204 field names:

```
%INTEREST

YEAR TO DATE

USE COUNT

NAME??

'VALUE'
```

Forming field values

The following rules apply for forming Model 204 field values:

- A field value can be up to 255 characters in length.
- Except for the following characters, you can use any word or character (including a space) as part of a field value:

  ```
  ?? ?$ ?& @ # ; :
  ```

Note: The restriction on using at sign (@) and number sign (#) characters is the same as for field names. See “Rules for naming fields” on page 60.

- When more than one consecutive space appears in a field value, Model 204 ignores the extra spaces.
- When using a reserved word or character in a value used with the EDIT form of IFUPDT or IFPUT or a %variable assignment, do not enclose the value with quotation marks.
- Enclose values that contain reserved words with quotation marks in all other instances (such as with IFFIND).

Examples of valid field values

The following examples are valid values for Model 204 fields:

```
PARENTS='MARY AND JOHN SMITH'

CITY=BOSTON
```

In these examples, PARENT and CITY are fields; their values appear to the right of the equal sign (=).

Using quotation marks

The following rules apply for using quotation marks:

- When quotation marks are used, an even number is required.
- A pair of single quotation marks denotes a quoted string, for example, 'TEXT'.
- Model 204 stores and uses quoted strings with quotation marks dropped.
- Model 204 replaces a pair of consecutive single quotation marks inside a quoted string with a single quotation mark upon storing or printing the string.

  For example, the quoted strings shown below on the left result in the output on the right:

Quoted string:	Output:
`PRINT 'FATHER''S NAME'`	`FATHER'S NAME`
`PRINT ''AND''`	`'AND'`

- Model 204 converts a '' (a pair of consecutive single quotation marks that is not included in a quoted string) to a character string of zero length, called a null string.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Request_composition_rules) for more information about quotation marks.

Field definitions and attributes

Defining fields

You can use IFDFLD to define a new field, and IFRFLD, which requires Model 204 file manager privileges, to redefine a field. When using IFDFLD or IFRFLD, a knowledge of field attributes is necessary.

Note: Field definition is usually the responsibility of the Model 204 file manager. Technical Support recommends that a site restrict tasks to selected personnel to prevent the proliferation of unnecessary fields and guarantee consistency of field names and field types among files of a group.

When to assign field attributes

You assign field attributes when you:

- Initially load a file
- Add a new field name to an established file (IFDFLD)
- Redefine a field description (IFRFLD)

  Note: If you define a field but do not specify field attributes, Model 204 assigns default attributes. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of the IFDFLD call.

Once a field attribute has been assigned, it applies whenever the field name appears in the file.

Field attributes

You can assign each field in a Model 204 file certain field attributes, including storage and security options, based on the characteristics of the data.

You can combine field attributes and options. The combination of attributes for a field governs the operational characteristics of a field and the amount of space required to store the field value internally. The operational characteristics and storage characteristics of a field usually are independent considerations.

The following sections describe briefly Model 204 field attributes. For complete information about field attributes, refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Field_design).

Operational characteristics of a field

Use the following field attributes to control the operational characteristics, that is, how the field is used:

Field attribute	Meaning
KEY	KEY fields provide quick access to data. Pointers to records that contain specific values of KEY fields are maintained in the index area. You can specify fields designated as KEY in record selection specifications to retrieve records based on index entry. Use IFFIND to select KEY fields. See page 68 for information about KEY inconsistencies.
NON-KEY	NONKEY fields do not contain pointers in the index area. You can use NONKEY fields to retrieve records with IFFIND, but selection is performed by sequential search and is less efficient than with KEY fields. See page 68 for information about NONKEY inconsistencies.
NUMERIC RANGE or RANGE	NUMERIC RANGE or RANGE fields contain certain numeric values. The index area contains multiple pointers to fields within a range of values. You can specify numeric range retrievals for these fields. NUMERIC RANGE fields cannot be defined to a file with the First-Normal Form (1NF) file model option set. See page 68 for information about NUMERIC RANGE inconsistencies.
NONRANGE	Model 204 does not store multiple pointers for NONRANGE fields. You can specify numeric range retrievals for these fields, but the selection is less efficient than for NUMERIC RANGE fields. See page 68 for information about NONRANGE inconsistencies.
ORDERED	Fields designated as ORDERED make up the Ordered Index. Model 204 maintains the values of Ordered Index fields in a particular collating sequence for alphabetic, alphanumeric, and numeric range retrievals to make range retrievals more efficient. An ORDERED field can have an order type of either CHARACTER or NUMERIC. You can access file record keys in order.
NONORDERED	NONORDERED is the default if you do not specify the ORDERED attribute when you define a field.
VISIBLE	VISIBLE fields appear in the data area and can be printed, noted, sorted, or used in arithmetic expressions. VISIBLE is the default field designation. See page 68 for information about VISIBLE inconsistencies.

Field attribute	Meaning
INVISIBLE	You can use fields designated as INVISIBLE to retrieve records. However, because INVISIBLE fields do not appear in the data area (only in the index areas), you cannot print, note, or sort INVISIBLE fields, or use them in an arithmetic expression. IINVISIBLE fields cannot be defined to a file having the First-Normal Form (1NF) file model option set, except in conjunction with the REPEATABLE field attribute. See page 68 for information about INVISIBLE inconsistencies.
FRV (For Each Value)	FRV fields have a list of all unique values assigned to the field. Special statements are available in SOUL to manipulate the values of FRV fields; IFFDV, IFGETV, and IFFTCH are the corresponding HLI calls.
NON-FRV	NON-FRV fields do not have a list of all unique values for the field.
DEFERRABLE	When you open a file in deferred update mode, Model 204 defers changes to DEFERRABLE field index entries until a later time.
NONDEFERRABLE	Model 204 changes index entries for NONDEFERRABLE fields as the fields are updated, even when you open the file in deferred update mode.
UNIQUE	UNIQUE fields have a uniqueness constraint enforced by the Unique Key. A given field name=value pair occurs in one record in a file. See page 71 for information about UNIQUE violations.
NONUNIQUE	NONUNIQUE fields do not have a uniqueness constraint on them.
AT-MOST-ONE	AT-MOST-ONE fields enforce a single occurrence constraint. The field may appear only once in a given record. Note that AT-MOST-ONE is the default for fields defined in a file having the First-Normal Form (1NF) file model option set. See page 71 for information about AT-MOST-ONE violations.
REPEATABLE	REPEATABLE fields do not have a single occurrence per record constraint associated with them. See page 71 for information about REPEATABLE violations.

Storage characteristics of a field

Use the following field attributes to control the storage characteristics, that is, how Model 204 stores field values in a file:

Field Attribute	Meaning
STRING	Model 204 stores STRING fields as character strings rather than in compressed numerical form.
BINARY	Model 204 compresses BINARY fields for efficient use of storage space. Note that Model 204 cancels a request to store invalid binary data in a file that has the NUMERIC VALIDATION file model option set. Seepage 68 for information about BINARY violations.
BLOB	Binary strings of type Binary Large Object. Limited to two gigabytes.
CLOB	Character string of type Character Large Object. Limited to two gigabytes.
CODED	Model 204 stores fields designated as CODED as codes in the data area to reduce storage space requirements, and as character strings in the file dictionary. Model 204 decodes CODED fields internally when retrieving or processing them. You can further designate CODED fields as FEW-VALUED or MANY-VALUED to optimize the coding procedure.
NONCODED	Model 204 stores NONCODED fields as character strings in the data area. See“Compression violations” on page 70 for information about NONCODED violations.
FLOAT	Model 204 stores FLOAT fields in floating-point form using IBM hardware floating-point representation. You must supply a LENGTH value with FLOAT to indicate the precision of the floating-point field being defined. Model 204 cancels a request to store invalid floating point data in a file that has the NUMERIC VALIDATION file model option set. See page 71 for information about FLOAT violations. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of floating-point format for IFFTCH, IFUPDT, IFGET, and IFPUT.
FEW-VALUED	Use the FEW-VALUED attribute for a field that has the CODED or FRV attribute and is expected to take on fewer than about 50 different values. Model 204 stores these values in a special section of the file dictionary (Table A).
MANY-VALUED	Use the MANY-VALUED attribute for a CODED or FRV field that is expected to take on at least 50 different values. Model 204 stores these values in a separate special section of the file dictionary (Table A).

Storage options for preallocated fields

Use the following storage options with preallocated fields:

Storage option	Meaning
OCCURS	Indicates the number of times that a field can occur in a record. See page 70 for information about OCCURS violations.
LENGTH	Indicates the maximum length of a field. See page 71 for information about LENGTH violations.
PAD	Indicates the padding character for short fields.

Field updating options

Use the following options to control the type of field updating performed:

Field updating option	Meaning
UPDATE IN PLACE	Causes the field to be changed. Its order in the record is preserved.
UPDATE AT END	Causes the field to be deleted and the new field added at the end of the record when the field is changed.

Field security option

Use the LEVEL option to assign a security level for a field.

Field definitions for group files

The following considerations apply when using fields defined in group file context:

- A given field is not required to have the same properties in all the files of a group. For example, a field can have the NUMERIC RANGE attribute in some files but not in others.
- A field that appears in one or more files of a group need not exist in others.
- If none of the files in a group contain a field name used in a selection specification, Model 204 rejects the specification during compilation.

When inconsistencies in field definition occur among files used in the group context, Model 204 performs certain actions, as described in the following section.

KEY and NONKEY inconsistencies

When you use a field name in selection specifications defined as KEY in some files and NONKEY in others, Model 204 carries out the following actions:

1. Performs a KEY search in files for which the field is defined as KEY.
2. Performs a direct search in files in which the field is defined as NONKEY.

NUMERIC RANGE and NONRANGE inconsistencies

When you use retrieval specifications based on fields that have NUMERIC RANGE or NONRANGE attributes, Model 204 handles them in the same way as for KEY and NONKEY specifications when they occur in a group context.

VISIBLE and INVISIBLE inconsistencies

You can retrieve only VISIBLE fields with IFFTCH, IFGET or IFMORE. If a field is INVISIBLE in one or more files in a group, Model 204 returns a null string for the field, as if it did not exist in the record.

Field access violations

The following types of violations can occur when you attempt to access a field:

- Field-level security
- LENGTH
- OCCURS
- Compression
- UNIQUE
- AT-MOST-ONE

The following sections describe each of these violations related to HLI processing. See “Field name variable errors” on page 78 for information about violations with %%variables.

Field-level security violations

Field-level security controls access to the individual fields of a Model 204 file. You can assign a security level to a field by using IFDFLD or IFRFLD.

You can specify any of the following types of field access:

- SELECT, which allows the field to be used in an IFFIND or IFFILE call.
- READ, which allows the value of a field to be examined by IFFTCH, IFGET, or IFMORE.

- UPDATE, which allows the value of a previously stored occurrence of a field to be changed by IFUPDT, IFPUT, IFDVAL, or IFFILE.
- ADD, which allows new occurrences of a field to be added by IFUPDT, IFPUT, or IFFILE.

Model 204 handles field-level violations

When you attempt to access a field having field-level security, access succeeds only if your user access level is equal to or greater than the level defined for the field. Otherwise, a field-level security violation occurs.

Model 204 returns a completion code of 4 for a field-level security violation and an error message. Note that you can retrieve the error message using IFGERR.

In general, Model 204 handles field-level security violations differently depending on the type of access violation and whether it occurs in file or group context.

Model 204 takes different actions for an access violation depending on the type of access attempted when the violation occurred:

- For SELECT or READ access, Model 204 performs the operation as if the field did not exist.
- For UPDATE or ADD access, Model 204 cancels the HLI call and returns a completion code of 4 to the HLI program.

Although a user's access rights are fixed for a group, the field's access level can vary from file to file. This might make access to a field legal in some files and illegal in others. Model 204 rejects a HLI call that contains a field name reference under the following conditions:

- In file context, if the access is not allowed in the file.
- In group context, if the access is not allowed in any file in the group.

  Note: Use IFFLS in your HLI program to check for field-level security violations before they occur.

Field-level violations for IFFTCH, IFUPDT, IFGET, and IFPUT

Model 204 treats HLI calls that compile IFGET (or IFMORE) and IFPUT specifications in a special way, because a specification compiled by IFGET (or IFMORE) can be executed by IFPUT, and vice versa. The same is true for IFFTCH and IFUPDT specifications on a multiple cursor IFSTRT thread.

If an HLI thread has READ, UPDATE, or ADD access to a field, you can reference the field using IFFTCH or IFUPDT on a multiple cursor IFSTRT thread, or IFGET or IFPUT on a single cursor IFSTRT thread.

For field-level violations involving IFFTCH, IFUPDT, IFGET, or IFPUT, Model 204 performs the following actions:

- If a violation occurs during IFFTCH, IFGET, or IFMORE processing, Model 204 returns a null value for the affected field or fields and continues processing the call.
- For the retrieve-all-fields form of the data specification with IFFTCH or IFGET, Model 204 returns only fields for which the user has READ access.
- If a violation occurs during IFUPDT or IFPUT processing, Model 204 stops processing the call and returns a completion code of 4 to the HLI program.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of IFFTCH, IFUPDT, IFGET, IFMORE, and IFPUT calls.

LENGTH violations

A field can be defined as preallocated and having a maximum LENGTH ***m***. Space for this length is preallocated in each record in a file, and this space cannot be expanded.

Note: If a field is defined as having a LENGTH of ***m***, that you can assign that field only values that are between 1 and ***m*** bytes long.

Model 204 rejects any attempt to store values greater than the specified maximum length with IFUPDT, IFSTOR, IFPUT, or IFBREC, by canceling the call and returning a completion code of 4 to the HLI program.

Model 204 treats any attempt to locate or delete unacceptable values as references to nonexistent values with the following actions:

- For IFFIND, the selection criteria fail to locate any records.
- IFDVAL does not delete any fields.

OCCURS violations

If a field is defined as occurring a particular number of times, that is, OCCURS ***n***, it can be stored at most ***n*** times in any record.

Model 204 cancels the call and returns an error completion code to the HLI program for either of the following conditions:

- If the HLI program issues either IFUPDT or IFPUT and attempts to add a new occurrence of an OCCURS n field to a record that already contains the maximum number.
- During processing, if Model 204 encounters a nonnumeric value in an OCCURS FLOAT field.

Compression violations

If a field is defined as having the attributes OCCURS n, BINARY, and NONCODED, only compressible values can be stored in it.

A small amount of space is preallocated for such a field. A compressible value is a decimal integer of as many as nine digits, but without a plus sign, leading zeroes, embedded blanks following a minus sign, or a decimal point.

Note: If you use IFUPDT or IFPUT and attempt to store an incompressible value in an OCCURS n, BINARY, NONCODED field, Model 204 cancels the call and returns a completion code of 4 to the HLI program.

UNIQUE violations

If a field is defined as UNIQUE, only one record in the file can contain any given value of that field.

Note: If you use IFUPDT, IFSTOR, IFPUT, or IFBREC and attempt to store the same value of a UNIQUE field in another record, Model 204 cancels the call and returns a completion code of 200 to the HLI program.

If a completion code of 200 is returned, use the IFEFCC call to determine which records are creating the uniqueness violation. See the *Rocket Model 204 Host Language Interface Reference Manual* for more information about IFEFCC.

NUMERIC VALIDATION violations

During processing, if Model 204 encounters invalid data for BINARY and FLOAT numeric field types, Model 204 cancels the call and returns an error code to the HLI program.

AT-MOST-ONE violations

If a field is defined as AT-MOST-ONE, only one occurrence of that field can be stored in a particular record.

Note: If the HLI program issues a call to IFSTOR, IFUPDT, or IFPUT, and attempts to store more than one (that is, a second) occurrence of the field within a particular record, Model 204 cancels the call and returns a completion code of 202 to the HLI program.

If a completion code of 202 is returned, use the IFEFCC call to determine which records are creating the AT-MOST-ONE violation. See the *Rocket Model 204 Host Language Interface Reference Manual* for information about IFEFCC.

Using %variables

A %variable is an entity whose name can substitute another value or expression in a SOUL statement or HLI program. %Variables can be used any place in SOUL that a value can be used.

You can change the value of a variable at any time by assigning it a new value. You assign one or more %variables in an HLI call by specifying the %variable parameters (%variable buffer and %variable specification) in the parameter list.

See the following pages for more information about these %variable parameters.

The %variable option is available for use with the following IFSTRT calls:

Compile and execute:	Execute-only:
IFBREC	--
IFCLST	--
IFCTO	IFCTOE
IFFAC	IFFACE
IFFDV	IFFDVE
IFFIND	IFFINDE
IFFNDX	IFFNDXE
IFFTCH	IFFTCHE
IFFWOL	IFFWOLE
IFGET	IFGETE
IFGETX	IFGETXE
IFMORE	IFMOREE
IFMOREX	IFMORXE
IFOCC	IFOCCE
IFOCUR	IFOCURE
IFPUT	IFPUTE
IFSKEY	IFSKYE
IFSORT	IFSRTE
IFSTOR	IFSTRE
IFUPDT	IFUPDTE

Note: Except for IFBREC and IFCLST, the IFSTRT thread calls listed above are used with the Compiled IFAM facility. Refer to Chapter 2 for information about the Compiled IFAM facility.

Specifying a %variable name

The following rules apply for specifying a %variable name:

- The first character of the %variable name must be a percent sign (%).
- Any combination of letters and numbers is valid.

- No embedded blank or colon (:) characters are allowed in the name.
- You can repeatedly reassign a %variable name. However, each assignment affects all specifications in which the %variable is referenced.

Example of when to use a %variable

Use %variables when your HLI application program performs similar and successive data retrievals.

For example, an IFFIND call frequently appears inside an IFGET loop. The IFFIND uses the value retrieved by the IFGET as part of the selection criteria.

To cross-reference without using the %variable facility, you must code your program to move the value retrieved by the IFGET into the IFFIND selection criteria. Since the IFFIND specification is modified during execution, it cannot be compiled and saved from one call to the next.

In contrast, using the %variable facility you can use %variables to replace field values in IFFIND criteria that are not known at compilation time.

For example, a specification that is referred to in an IFFIND call might have the following form:

```
LAST NAME=%NAME AND RECTYPE=CHILD;END;
```

Using this specification, during the execution of IFFIND, the appropriate last name substitutes for %NAME. In the next execution of IFFIND, you can specify a new last name or you can reuse the old one. See “Specifying the %variable parameters” for information about specifying %variables in an HLI call.

Specifying the %variable parameters

You assign one or more %variables in an HLI call by specifying the %variable input parameters, that is, %variable buffer and %variable specification, in the parameter list.

Note that if you specify one, you must specify the other. Specify each of the %variable parameters as a character string, whose maximum length is the input buffer size. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of HLI calls and their parameter lists.

The %variable buffer parameter (%VARBUF in the parameter list descriptions) contains the values to be assigned to the %variables that are used by the call.

The %variable specification (%VARSPEC in the parameter list descriptions) describes the format of the data contained in the %variable buffer and, in some cases, the list of %variables to be assigned. The %variable specification follows the syntax of the LIST, DATA, or EDIT specifications used with IFUPDT or IFPUT.

See “Assignment of %variables” on page 75 for more information about the %variable specification. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of IFUPDT and IFPUT.

Using %variables in EDIT and LIST specifications

You can use a %variable in the field name list of an EDIT or LIST specification for the IFFTCH, IFGET, IFGETX, IFMORE, IFMOREX, IFPUT, IFSTOR, and IFUPDT calls. See the specific calls in the *Rocket Model 204 Host Language Interface Reference Manual* for more details.

Example of using a %variable

The following excerpt shows how to use a %variable in an IFFIND call to retrieve a series of Social Security numbers from records. In this example, the IFFIND is coded inside a program loop that retrieves records.

The sample COBOL program issues the following IFFIND call:

```
CALL "IFFIND" USING RETCODE,FDSPEC,FDNAME,SSN,SSNEDIT.
```

where:

- RETCODE is the Model 204 completion code.
- FDSPEC is the name of a data area that contains selection criteria for IFFIND (also known as the FIND specification).
- FDNAME is the name of the compilation to be used by subsequent executions.
- SSN specifies the %variable buffer parameter; SSN is the name of a data area that contains the value of the Social Security number, which is assigned to the %variable %SSN.
- SSNEDIT specifies the %variable specification parameter; SSNEDIT describes the format of the SSN data area.

  The DATA DIVISION of the sample COBOL program contains the following definitions of the IFFIND parameters:

```
WORKING-STORAGE SECTION.
01  CALL-PARMS       COMP SYNC.
02  RETCODE          PICTURE 9(5).
•
•
•
01  ALPHA-PARMS.
    02  FDSPEC        PICTURE X(18) VALUE
    'SOCSECNO=%SSN;END;'.
    02  FDNAME        PICTURE X(5) VALUE 'FIND;'.
    02  SSN           PICTURE 9(9).
```

```
    02  SSNEDIT       PICTURE X(17) VALUE
       'EDIT(%SSN)(A(9));'.
•
•
•
PROCEDURE DIVISION.
CALL "IFFIND" USING RETCODE, FDSPEC, FDNAME, SSN, SSNEDIT.
•
•
•
```

Assignment of %variables

Values in the %variable buffer are assigned to %variables depending on what type of %variable specification you use. You can specify the %variable specification with a LIST, DATA, or EDIT format as is used with IFUPDT or IFPUT.

Model 204 assigns %variables in the following ways:

- For a LIST specification, the format is:

  ```
  LIST(%variable1,%variable2, •••);
  ```

  where, each value in the %variable buffer is assigned to the corresponding %variable in the list. The values in the %variable buffer are stored in the following form:

  ```
  'value1''value2'•••
  ```

- For a DATA specification, the format is:

  ```
  DATA;
  ```

  where, explicit %variable assignments are included in the %variable buffer parameter in the following form:

  ```
  %variable1=''value1'; %variable2=''value2'•••
  ```

- For an EDIT specification, the format is:

  ```
  EDIT(%variable1,%variable2,•••)(format1,format2,•••);
  ```

  where, each value in the %variable buffer is assigned to the corresponding %variable in the list according to the corresponding format. See the COBOL coding excerpt on the preceding page for an example of an EDIT specification using a %variable.

Assignment of %variables for HLI threads

Each HLI thread has its own set of %variables.

Upon the initial reference to a %variable on an HLI thread, Model 204 assigns a null string. The %variable retains the null value until an explicit assignment is made.

Note: If you attempt to assign a value to a previously unreferenced %variable, Model 204 displays a NAME LIST SYNTAX ERROR message.

After a value is assigned to a %variable, the value and the format are carried from one HLI call to the next, making repeated assignment unnecessary for values that do not change. To avoid redundant processing, specify only those %variables in an HLI call that take on new values.

Using field name variables

Field name variables are extensions of percent variables (%variables).

Field name variables enable you to refer indirectly to field names, thereby enabling HLI calls to be generalized. Using field name variables, you specify the actual field names used by a particular HLI call when you execute your HLI program.

Specifying a field variable name (%%variable)

The same rules apply for specifying a field variable name as for a %variable name except that the field variable name begins with two percent signs(%%). For example, %%FIELD can be used to specify a field name variable.

See “Specifying a %variable name” on page 72 for naming rules.

Note: A %variable and a %%variable that have identical names are actually the same variable. When the variable is used in a %variable context, for example, as the value in a FIND specification, it takes the %name form. Or, if the variable is used in a field name context, for example, as part of a field name list in a GET specification, it takes the %%name form.

In the latter case, the value of the variable at the time that the GET specification is executed is substituted in the specification and is then treated as a field name. When a value is assigned to a field name variable, the name of the value that should be assigned to the field name variable is actually a percent variable, that is, a %name rather than a %%name.

When to use a field name variable

You can reference a field name variable in any of the following IFSTRT thread calls:

- IFCTO
- IFFIND
- IFFNDX

- IFFTCH
- IFGET
- IFMORE
- IFOCC
- IFPUT
- IFUPDT

You can specify a %%variable in a HLI call anywhere you would normally reference a field name. You can also specify a %%variable in a DATA input specification in an IFUPDT or IFPUT call.

Example of using %%variables

The following COBOL code excerpt contains two different record types. The types have nearly all fields in common, but they differ in the following ways:

- Record type A has a STATUS field, while record type B has a PERFORMANCE field.
- Record type A has a SALARY field, while record type B has a COMPENSATION field.

This COBOL example illustrates field name variable processing in the following way:

- In EDITLIST-1, the field name variables (%%STATUS and %%SALARY) represent the actual field names in the two record types.
- The IFGETC call compiles the GET specification. This specification is then executed by the two IFGETE calls.
- The field names are assigned by EDITLIST-2 depending on the record type encountered. For record type A, field names are STATUS and SALARY, for record type B, field names are COMPENSATION and PERFORMANCE.

```
01  WORK-REC.
    02 REC-TYPE          PIC X.
    02 WORD-SSN          PIC 9(9).
    02 WORK-NAME         PIC X(30).
    02 WORK-BDATE        PIC 9(6).
    02 WORK-SCDATE       PIC 9(6).
    02 FILLER            PIC X(2).
    02 WORK-GRADE        PIC 9(2).
    02 WORK-STEP         PIC 9(2).
    02 WORK-STATUS       PIC X(2).
```

•
•

```
•
01  ALL-PARMS.
    02 RETCODE           PIC 9(5) COMP SYNC.
    02 CNAME             PIC X(8) VALUE 'FTCHREC;'.
    02 DIR1              PIC 9 COMP SYNC VALUE 1.
    02 DIR0              PIC 9 COMP SYNC VALUE 0.
    02 EMPCUR            PIC X(7) VALUE 'EMPCUR;'.
    02 REC-A             PIC X(13) VALUE
    'STATUS SALARY'.
    02 REC-B             PIC X(24) VALUE
    'PERFORMANCE COMPENSATION'.
•
•
•
01 PERCENT-PARMS.
   02 EDITLIST-1         PIC X(110) VALUE
      'EDIT (REC-TYPE,SSN,NAME,BDATE,SCDATE,GRADE,
      STEP,%%STATUS,%%SALARY)
      (A(1),A(9),A(30),2A(6),X(2),2J(2),A(2),A(8));'.
   02 EDITLIST-2         PIC X(36) VALUE
      'EDIT (%STATUS,%SALARY)(A(11),A(12));'.
•
•
•
CALL 'IFFTCH' USING RETCODE,WORK-REC,DIR1,EMPCUR,
      CNAME,EDITLIST-1.
•
•
•
IF REC-TYPE='A' THEN
CALL 'IFFTCH' USING RETCODE,WORK-REC,DIR0,EMPCUR,
      CNAME,REC-A,EDITLIST-2.
•
•
•
IF REC-TYPE='B' THEN
CALL 'IFFTCH' USING RETCODE,WORK-REC,DIR0,EMPCUR,
      CNAME,REC-B,EDITLIST-2.
•
•
•
```

Field name variable errors

When you use a field name variable, Model 204 defers validity checks, which would normally be performed when a specification is compiled, until the specification is executed and the actual field name is filled in.

You might encounter the following errors for field name variables:

- If the field name variable specified in an IFFIND call is not valid, Model 204 does not execute the call.
- If a missing field name is encountered when processing IFUPDT or IFPUT, Model 204 aborts processing of the field name list and returns a completion code of 4.
- If a missing field name is encountered when processing IFFTCH or IFGET, Model 204 returns a null string value as if the field were missing from the record.

8

Find Criteria for Model 204 Data

Overview

This chapter describes how to specify find criteria to select data from the Model 204 database for application programmers who are using the Host Language Interface facility.

For more information

Refer to Chapter 7 for a description of fields and variables. Refer to Chapter 6 for a description of files and records.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of HLI calls. Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Basic_SOUL_statements_and_commands#Understanding_retrieval_conditions) for complete information about selecting data.

Find criteria in HLI calls

An HLI application program specifies find criteria in calls that perform find functions. To select data from the Model 204 database, specify find criteria using any of the following HLI calls on an IFSTRT thread. On an IFDIAL thread you can also specify find criteria using the equivalent SOUL command, which is listed beside each HLI call:

HLI call	Equivalent SOUL command
IFFAC	FIND AND PRINT COUNT
IFFDV	FIND ALL VALUES
IFFIND	FIND

HLI call	Equivalent SOUL command
IFFNDX	FIND EXCLUSIVE
IFFWOL	FIND WITHOUT LOCKS

A find specification can include a combination of conditions that might require a mix of index and direct searches against the database. See “File search operations” on page 83 for more information about index and direct searches.

Model 204 selects records in order by one of the following storage schemes, based on the file type:

- In record number sequence, for records in an entry order or hashed file
- In sort key sequence, for records in a sorted file
- In a particular collating sequence, for records in an ordered file

Refer to Chapter 6 for more information about different types of data files.

Specifying all records to be selected

An HLI application program can select all records in the current file or group by omitting criteria in the find specification. To specify all records to be selected, use the following entry in the find specification in the HLI call:

```
;END;
```

Specifying particular records to be selected

To select a particular group of records from a file of group for processing, an HLI application program must specify find criteria. A single HLI call can include multiple conditions for selecting data in a find specification.

Specify find criteria using either:

- Values, to select fields that contain numeric and character values
- Patterns, to select a field value based on a character string pattern

Model 204 performs two basic kinds of find criteria comparisons, numeric and character, based on the type of data that is stored in a file.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Basic_SOUL_statements_and_commands#Understanding_retrieval_conditions) for complete information about find criteria and examples of find specifications.

File search operations

Each Model 204 file contains a data area and an index area. The data area, Table B, contains the data records. The index area contains references to the data records organized by key field and value.

Index search

When Model 204 locates records stored in the data area using the index area, this is called an index search.

Usually, Model 204 searches the index, either the hash index (Tables C and D) or the Ordered index (Table D, that is, the B-tree), for fields having any of the following types of attributes:

- KEY
- ORDERED
- NUMERIC RANGE

When storing fields that have these attributes, Model 204 makes special entries in the index. For certain types of find criteria using values, Model 204 directly accesses the appropriate index entry to find which records satisfy the find criteria without searching through the records in the data area of the file.

Whether or not Model 204 uses the index depends on the type of find criteria specified. See Table 8-1 on page 84 for a summary of file search operations for fields having KEY, ORDERED, and NUMERIC RANGE attributes. Note that, in general, find criteria based on KEY, ORDERED, or NUMERIC RANGE fields are extremely fast and efficient.

Direct data search

In contrast to find criteria that involve an index search, the following types of find conditions result in a direct search of the data in the Table B data area:

- Inequality character conditions
- IS PRESENT condition
- Conditions that specify fields having any of the following types attributes:
  - NONKEY
  - NONORDERED
  - NONRANGE

Note: A direct search can have a significantly adverse effect on performance if the search involves a large number of records. However, the cost can be greatly reduced in some cases where both index and direct find criteria are specified in the same HLI call.

When processing index and direct find criteria for the same HLI call, Model 204 reduces the number of records to be searched directly by performing the index selection first. Model 204 does not search directly records that are either selected or eliminated based on the index search.

File search for a group

When selecting data from a file group, Model 204 first determines which records from the individual member files meet the selection specifications. This yields a set of records that might contain entries from all files.

Model 204 then selects the individual records in order by file.

Summary of file search operations

Table 8-1 summarizes file search operations for an HLI call that finds records. The type of file search depends on field attributes and on find criteria specified in the call.

In Table 8-1, the search operations correspond to find criteria specified in the following HLI calls: IFFIND, IFFAC, IFFWOL, and IFFNDX. The find criteria listed in the table, that is, an equality, a character range, a numeric range, and a character and numeric range combination, specify value criteria.

See “Specifying find criteria: character values” on page 85 for a description of find specifications that specify value criteria. Refer to Chapter 7 for information about field attributes.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Record_retrievals) for information about optimizing file search operations for find specifications.

Table 8-1. File search based on field attributes and find criteria

Type of Find Criteria	Field Attribute						
	Key	Numeric range	Ordered		Key and Ordered		Other
			Number	Character	Number	Character	
Equality	Index1	Index1	Index2	Index2	Index1	Index1	Data
Character range	Data	Data	Data	Index2	Data	Index2	Data
Numeric range	Data	Index1	Index2	Data‘	Index2	Data	Data
Character and numeric range	Data	Data	Data‘	Index2	Data	Index2	Data

Data = Data area search in Table B
Index1 = Hash index search in Table C and, if needed, in Table D
Index2 = Ordered index search in Table D (B-tree)

Specifying find criteria: character values

You can specify selection criteria that tests the values of a field to determine whether a record with a particular value, either character or numeric, is selected.

You can specify any of the following types of find criteria using values:

- Equality (using an equal sign)
- Character range
- Numeric range
- Character and numeric range combination

The following sections give basic guidelines for use, general syntax, and examples of value find criteria. Table 8-2 on page 93 summarizes the operators that are valid for use in find specifications with value criteria.

Specifying find criteria using character values

With character find criteria you can test a field that contains character string values to locate records that have a particular value or range of values. The comparison uses the EBCDIC collating sequence.

Specify character find criteria using any of the following formats:

```
field name = value

field name = NOT value

fieldname {IS | IS NOT} [ALPHABETICALLY] {BEFORE | AFTER
        | operator} value
```

where *operator* specifies one of the range operators in Table 8-2 on page 93.

You can specify character find criteria for a KEY or NONKEY field. Note, however, that selection of KEY fields is substantially more efficient than selection of NONKEY fields.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Record_retrievals) for more information about character values used in find specifications.

Character find criteria with an equality condition

The following examples illustrate character value find criteria that specify an equality condition.

In the following example, the field name is TOWN and the value is CAMBRIDGE. The following find criterion specifies employees who live in CAMBRIDGE:

```
TOWN=CAMBRIDGE;END;
```

Alternatively, the following find criterion specifies employees who do not live in CAMBRIDGE:

```
TOWN=NOT CAMBRIDGE;END;
```

Character find criteria with a range condition

The following examples illustrate character value find criteria that specify a range condition:

- In the following example, the field is NAME and the value is SMITH. The following find criterion specifies names that precede the value SMITH in EBCDIC collating sequence:

  ```
  NAME IS BEFORE SMITH;END;
  ```

  In this example, the BEFORE SMITH specification selects records that contain the value NAME=SMALL but does not select records that contain the value NAME=SMITHIE.

- In the following example, the field is NAME. The following find criterion specifies names that do not follow the value WALKER in EBCDIC collating sequence:

  ```
  NAME IS NOT AFTER WALKER;END;
  ```

- In the following example, the field is NAME. The following find criterion specifies names that do not precede the value JOHNSTON in EBCDIC collating sequence:

  ```
  NAME IS NOT ALPHA LT JOHNSTON;END;
  ```

- In the following example, the field is NAME. The following find criterion specifies names that follow THORNE and precede THYME (such as, THULE) in EBCDIC collating sequence:

  ```
  NAME IS BEFORE THYME AND AFTER THORNE;END;
  ```

- In the following example, the field is NAME. The following find criterion specifies a negative relation and selects names equal to and preceding THORNE and equal to and following THYME in EBCDIC collating sequence:

  ```
  NAME IS NOT BEFORE THYME AND AFTER THORNE;END;
  ```

Specifying find criteria: numeric values

With numeric find criteria you can test a field for numerical values less than, greater than, or equal to a particular value. You can specify negative numbers.

You can specify numeric find criteria for fields that have any of the following attributes:

- ORDERED NUMERIC
- NUMERIC RANGE
- NONRANGE

Specify numeric find criteria using either of the following formats:

```
fieldname = value
```

```
fieldname {IS | IS NOT} [NUMERICALLY] [operator] value
```

where *operator* specifies one of the range operators in Table 8-2 on page 93.

Rules for specifying numeric range find criteria

The following rules apply for specifying numeric range find criteria:

- When you specify IS, Model 204 selects records in which the content of the numerical field is algebraically equivalent to the specified constant. Note that this differs from specifying the equal sign (=), which results in a search for an exact character match.
- A negated comparison operator (IS NOT) for numeric find criteria produces a different result for a NUMERIC RANGE field and a NONRANGE or ORDERED field.

  See “Field attributes and negated numeric find criteria” on page 89 for differences in find criteria for fields having different attributes.
- If a numeric range find is performed on a field that is defined with the NUMERIC RANGE or ORDERED NUMERIC attribute, Model 204 finds only those records where the numerical fields contain the following characters:
  - Optional leading plus (+) or minus (-) sign.
  - Digits 0-9 and an optional decimal point.
  - No more than 10 digits on either side of the decimal point, and a maximum of 20 digits, for NUMERIC RANGE fields.
  - No more than 10 digits on either side of the decimal point for ORDERED NUMERIC fields, and only the first 15 significant digits are used.

  See the next page for examples of specifications that contain the required characters for NUMERIC RANGE and ORDERED NUMERIC fields.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Record_retrievals) for more information about numeric find criteria.

Numeric find criteria with an equality condition

The following examples illustrate numeric value find criteria that specify an equality condition.

There are differences using an equal sign (=) and the word IS in a numeric specification. For example, the following find criterion selects only records that contain a value of 37 in the field WEIGHT:

```
WEIGHT=37;END;
```

Alternatively, the following find criterion selects records that contain values of 37, 0037, 37.0, for WEIGHT, and that might include any other string whose numerical value is 37:

```
WEIGHT IS 37;END;
```

Fieldname=value pairs for numeric find criteria

The following examples are of valid fieldname=value pairs for fields defined with NUMERIC RANGE or ORDERED NUMERIC attributes:

```
PRICE = 3317

AGE = 013

VOLUME = 517.6473

WAVE LENGTH = +.0072

TEMPERATURE = -21
```

See “Specifying find criteria: numeric values” on page 86 for information about valid characters for NUMERIC RANGE and ORDERED NUMERIC field values.

Numeric find criteria with a range condition

The following examples illustrate numeric value find criteria that specify a range condition:

- In the following example, the field is SETTLEMENT DATE. The following find criterion specifies dates that are less than the specified value 800305 (that is, before March 5, 1980):

  ```
  SETTLEMENT DATE IS < 800305
  ```

- In the following example, the field is AGE. The following find criterion specifies ages that are greater than the specified value 20:

  ```
  AGE IS NUM AFTER 20
  ```

In this example, NUM (NUMERICALLY*)* specifies that a numeric type comparison be performed using the operator AFTER, instead of the default, which is character comparison.

- In the following example, the field is AGE. The following find criterion specifies records in which age is numerically less than 21:

  ```
  AGE IS LESS THAN 21;END;
  ```

- In the following example, the field is AGE. The following find criterion specifies records in which age is numerically equal to or greater than 21 (a NUMERIC RANGE field attribute is assumed):

  ```
  AGE IS NOT LESS THAN 21;END;
  ```

- In the following example, the field is AGE. The following find criterion specifies records in which age is numerically greater than 21:

  ```
  AGE IS GREATER THAN 21;END;
  ```

- In the following example, the field is TEMPERATURE. The following find criterion specifies records in which the value of temperature is numerically less than minus 10 (degrees):

  ```
  TEMPERATURE IS LESS THAN -10;END;
  ```

- In the following example, the field is AGE. The following find criterion specifies ages that are numerically greater than 21 and less than 25:

  ```
  AGE IS BETWEEN 21 AND 25;END;
  ```

- In the following example, the field is AGE. The following find criterion specifies ages that are numerically less than or equal to 21 and greater than or equal to 25 (a NUMERIC RANGE field attribute is assumed):

  ```
  AGE IS NOT BETWEEN 21 AND 25;END;
  ```

Field attributes and negated numeric find criteria

For numeric find criteria that specify a negated condition, the set of records selected depends on whether the field is defined with the NUMERIC RANGE, ORDERED, or NONRANGE attribute.

In the following example, the field is AGE. The following negated find criterion specifies records in which age is not less than 21:

```
AGE IS NOT LESS THAN 21;END;
```

Using this example, Model 204 selects records differently based on the field attributes defined for AGE, as follows:

- If AGE is defined as NUMERIC RANGE, this statement selects records containing AGE fields whose values are numerical and greater than or equal to 21.
- If AGE is defined as NONRANGE or NONORDERED, this statement selects records that do not contain AGE fields less than 21. However, the selected records include those that contain nonnumerical AGE fields and those that contain no AGE field.
- If AGE is defined as NONRANGE and INVISIBLE, this statement does not select any records.

Specifying value find criteria using an IN RANGE clause

You can specify beginning and ending values in find criteria or combine conditions in find criteria by using the IN RANGE clause.

Use the following form of the IFFIND, IFFAC, IFFNDX, and IFFWOL call to specify a beginning and ending range of values:

```
fieldname {IS | IS NOT} [NUMERICALLY | ALPHABETICALLY]
   IN RANGE [FROM | AFTER] value1 {TO [AND] BEFORE} value2
```

Note the following uses for the IN RANGE clause:

- The IN RANGE clause is particularly useful when a beginning and ending range of values must be specified for a multiply occurring field.
- If you use the IN RANGE clause for ORDERED fields, the find can be optimized because the search on the Ordered Index is restricted between two values.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Record_retrievals#IN_RANGE_clause) for more information about the IN RANGE clause.

Defining a numeric value in exponent notation

You can define a numeric value in a find specification in exponential notation.

Exponential notation has the following format:

```
[ + | - ]{whole_number | whole_number.fractional_number |
          .fractional_number} E[ + | - ]exponent
```

where:

- A maximum of 15 significant digits is allowed for number values. If the number of significant digits exceeds 15, the remaining precision is lost.
- The *exponent* expression must be between 75 and -75.

- Embedded spaces are not allowed in the exponent string.

The following examples are of valid numeric values expressed in exponent notation:

```
-1322.444E14

15E-47

+99233.0332E-66

222E+11
```

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for descriptions of the IFFTCH and IFGET calls and for more information about conversion of exponential numbers.

Refer to the Rocket Model 204 wiki documentation (http://m204wiki.rocketsoftware.com/index.php/Record_retrievals#Exponent_notation) for more information about exponent notation.

Specifying find criteria: special conditions

You can specify any of the following types of special conditions in find specifications in HLI calls that find records (that is, an IFFIND, IFFAC, IFFWOL, and IFFNDX):

- IS PRESENT condition, which selects only records that contain at least one occurrence of the specified field regardless of its value.

  In the following example, the specification locates all records that contain one or more occurrences of the field CHILD:

  ```
  CHILD IS PRESENT;END;
  ```

  Alternatively, in the following example, the specification locates all records that do not contain a ZIP CODE field:

  ```
  ZIP CODE IS NOT PRESENT;END;
  ```

- FIND$ condition, which selects a set of records previously selected.
- SFL$ condition, which selects all records for which the sort field is less than the stated value.
- SFGE$ condition, which selects all records for which the sort field is greater than or equal to the stated value.

  In the following example, from a file sorted by last name, the specification selects the records for all males starting at JONES:

  ```
  SFGE$ JONES;SEX=MALE;END;
  ```

  And, in the following example, the specification locates records of all last names starting with the letter K:

```
SFGE$ K;SFL$ K999;END;
```

- POINT$ condition, which selects all records in a file that contain internal record numbers greater than or equal to a specified value. (POINT$ is not valid in group context.)

  In the following example, the statement selects all records in which AGE is greater than 20, and in which record numbers range from 288 to 5000 inclusive:

  ```
  AGE IS GREATER THAN 20;POINT$ 288 AND
  NOT POINT$ 5001;END;
  ```

- FILE$ condition, to restrict files from which records meeting the specified criteria are selected.

  In the following example, if the program has opened group REGION, which contains files MARYLAND, VIRGINIA, and DELAWARE, the specification finds all males in the group who do not live in Delaware:

  ```
  SEX=MALE;NOT FILE$ DELAWARE;END;
  ```

  In the following example, if the program has opened group REGION, which contains files MARYLAND, VIRGINIA, and DELAWARE, the specification finds all males in the group who live in Delaware or Virginia:

  ```
  SEX=MALE;FILE$ DELAWARE OR VIRGINIA;END;
  ```

- LIST$ condition, which finds records that are on a list.

  In the following example, to find all males on list A1, the find specification is:

  ```
  SEX=MALE;LIST$ A1;END;
  ```

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Record_retrievals) for complete information about the special find conditions.

Using comparison operators

Table 8-2 summarizes the comparison operators that are valid for use in find specifications with character and numeric values.

Except for AFTER and BEFORE, which default to character-string comparison in EBCDIC sequence, the operators in Table 8-2 default to numeric comparison.

Table 8-2 does not show the IN RANGE clause, which defaults to numeric comparison, and the IS PRESENT condition, which defaults to character-string comparison.

Table 8-2. Comparison Operators

IS operator	Abbreviation	Symbol	The field value must be...
Numeric comparison			
EQUAL	EQ	=	Equal to the value specified.
NOT EQUAL	NE	¬=	Unequal to the value specified.
LESS THAN	LT	<	Less than the value specified.
LESS THAN OR EQUAL TO	LE	<=	Less than or equal to the value specified.
GREATER THAN	GT	>	Greater than the value specified.
GREATER THAN OR EQUAL TO	GE	>=	Greater than or equal to the value specified.
BETWEEN value1 AND value2	--	--	Between value1 and value2. If the field value is equal to value1 or value2, Model 204 does not select the record. **Note:** If value1 is greater than value2, Model 204 does not select any records.
Character-string comparison (EBCDIC collating sequence)			
AFTER	--	--	After the specified value or string
BEFORE	--	--	Before the specified value or string

Operator and value type mismatch

The following results occur when an operator and value type are not matched:

- If a specification uses a numeric operator with a nonnumerical value, Model 204 does not select any records. The following example illustrates this type of mismatch:

  ```
  LAST NAME IS GREATER THAN ANDREWS
  ```

- If a specification uses a numeric operator with a nonnumerical value and is negated, Model 204 selects every record in the file or group. The following example illustrates this type of mismatch:

  ```
  LAST NAME IS NOT GREATER THAN ANDREWS
  ```

- If a specification uses a character string operator with a numeric value, Model 204 converts all numeric values in the field to character strings and performs a character-by-character comparison of field values to the find criteria.

  The following example illustrates this type of mismatch:

  ```
  YEAR IS BEFORE 1986
  ```

  In this example, numbers such as 942, 700, or 2 would not be selected.

Note: To avoid unexpected results when using an operator that defaults to numeric comparison in character value find criteria, specify ALPHA (ALPHABETICALLY).

Interpretation of values used in find criteria

During data selection, Model 204 interprets numbers, strings, and exponential notation in value find criteria. Model 204 determines how to interpret values based on the type of find criteria (equality or range) and operators that the condition specifies.

Model 204 performs either a character or numeric comparison. The following sections describe the comparisons that Model 204 performs for equality and range conditions specified in find criteria. See Table 8-2 on page 93 for a list of operators and default comparisons.

Equality conditions in find criteria

For an equality condition in a find specification (having the form *fieldname* = *value*), Model 204 interprets the value differently depending on whether or not the field is defined with the FLOAT field attribute.

Model 204 performs comparisons for equality find criteria based on the following rules:

- If the field is defined with the FLOAT attribute, Model 204 examines the value to determine whether it is a number, exponent notation, or a character string.

  Depending on the value type, Model 204 performs one of the following comparisons:

  – Numeric comparison, if the value is a number or in exponent notation.

  – Character string comparison, if the value is a character string.

- If the field is not defined with the FLOAT attribute, Model 204 treats the value as a string and performs a character string comparison.

  Note: For a non-FLOAT field, Model 204 does not convert exponent notation to a numerical form. For example, if a field that is not defined with the FLOAT attribute contains a value of .1234E-3, the comparison .1234E-3 = .0001234 is not true.

Range conditions in find criteria

For either of the following types of range find criteria, Model 204 interprets the value differently depending on whether or not the condition specifies the NUMERICALLY or ALPHABETICALLY keyword:

- Having the form:

  fieldname IS *operator value*

- Using the IN RANGE clause

Depending on the keyword, Model 204 performs comparisons for range find criteria based on the following rules:

- If the condition specifies the NUM (NUMERICALLY) keyword, Model 204 interprets the value as a number and performs a numerical comparison.
- If the condition specifies the ALPHA (ALPHABETICALLY) keyword, Model 204 interprets the value as a string and performs a character string comparison.
- If the condition does not specify either the NUMERICALLY nor ALPHABETICALLY keyword, Model 204 interprets the value and performs the comparison according to the default comparison type of the operator.

  See Table 8-2 on page 93 for a list of operators and default comparisons.

Using Boolean operators

The following guidelines apply for using Boolean operators to specify find criteria:

- Find criteria can combine several Boolean operators as well as character and numeric comparison operators. You can use AND, OR, NOT, NOR, and parentheses to combine conditions in find criteria.
- Interpretation of a Boolean expression proceeds from left to right, except where the order of precedence dictates otherwise.
- An operation of higher precedence following an operation of lower precedence is performed first.
- Expressions enclosed inside parentheses are evaluated first. You can nest parentheses, in which case evaluation proceeds from the innermost to the outermost set.

Table 8-3 shows the order of precedence that Model 204 follows when evaluating multiple Boolean operations in a find criterion.

The following examples of find specifications with Boolean operators show the basic form of these specifications. Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Record_retrievals#Boolean_o

perators_in_retrieval_statements) for more information about Boolean operators.

Table 8-3. Order of precedence for Boolean operators

Order	Operator
1st	NOT
2nd	NOR
3rd	AND
4th	OR
5th	AND (implied by a new line)

Using Boolean operators to combine conditions

The following examples illustrate the basic form of a find specification that uses Boolean operators to combine conditions:

- In the following example, the field name is TOWN. The following find criterion specifies employees who live in CAMBRIDGE and in CHICAGO:

  ```
  TOWN=CAMBRIDGE OR TOWN=CHICAGO;END;
  ```

- The following specification shows how to modify the above example and produce the same results; the second TOWN= can be omitted:

  ```
  TOWN=CAMBRIDGE OR CHICAGO;END;
  ```

- In the following example, the field name is TOWN. The following find criterion specifies all employees except those who live in CAMBRIDGE, CHICAGO, and NEWPORT:

  ```
  TOWN=NOT CAMBRIDGE NOR CHICAGO NOR NEWPORT;END;
  ```

- In the following example, the field name is SKILL (in a personnel file). The following find criterion specifies employees who have a certain job skill combination, that is, typing and proficiency in the French language:

  ```
  SKILL=TYPING AND FRENCH;END;
  ```

- In the following example, the field name is SKILL. The following find criterion specifies employees who have a certain job skill (steno) and not another (typing):

  ```
  SKILL=STENO AND NOT TYPING;END;
  ```

Note: You can also combine find criteria using several Boolean operators as well as comparison operators for character and numeric values. Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Record_retrievals#Boolean_o

perators_in_retrieval_statements) for additional examples of find criteria using Boolean operators.

Specifying find criteria: pattern matching

You can specify find criteria using a field value that is in the form of a pattern. Specify patterns using the find calls: IFFIND, IFFAC, IFFWOL, IFFNDX, or IFFDV.

Using pattern find criteria, Model 204 does not process records for which the field specified in the find criteria is not present. For a character string find specification on a multiply occurring field, Model 204 selects all records with at least one occurrence of the field that meets the specified condition.

Note: Model 204 evaluates the pattern for each value in the specified field and selects those values that match the criteria in the pattern. Selected values must match patterns character by character, including blanks, except when special characters are used.

Specify a pattern-matching condition using the following format:

```
fieldname [IS | IS NOT] LIKE "pattern"
```

where:

- The keyword LIKE indicates that a pattern follows.
- *pattern* is enclosed in quotes. Model 204 treats a pattern like a character value that requires quotation marks to conform to SOUL format requirements.

  For more information about using quotes in SOUL, refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Request_composition_rules#Quotation_marks).

Table 8-4 summarizes the character codes that you can use to specify pattern-matching find criteria. Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Record_retrievals) for more information about pattern-matching find criteria and for explanations of the character symbols used to specify pattern-matching conditions.

Table 8-4. Pattern-matching codes for a FIND specification

Character	Symbol	Description
Asterisk	*	Wildcard
Plus sign	+	Placeholder
Comma	,	Or
Parentheses	( )	Sets begin and end
Hyphen	-	Range

Table 8-4. Pattern-matching codes for a FIND specification (Continued)

Character	Symbol	Description
Slash	/	Repeat
Exclamation point	!	Escape
Equal sign	=	Hexadecimal
Number sign	#	Numeric digit
At sign	@	Alphabetic

If the value of either of the following parameters is set as indicated, you must reset it before you can use the corresponding pattern character during line editing:

- ERASE parameter value is an alphabetic character (@)
- FLUSH parameter value is a numeric digit character (#)

For more information about the ERASE and FLUSH parameters, refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/ERASE_parameter and http://m204wiki.rocketsoftware.com/index.php/FLUSH_parameter).

9

Locking Behavior of HLI Calls

Overview

This chapter describes Model 204 resource locking and record locking for application programmers who are using the Host Language Interface facility. Refer to the descriptions of locking behavior for HLI calls when you are coding your host language program.

Refer specifically to information about the record locking behavior of multiple cursor IFSTRT thread calls if you are using multiple cursor features in your HLI program for the first time.

For more information

Refer to Chapter 10 for a description of locking conflicts and how to handle them in your HLI application. Refer to the Rocket Model 204 documentation wiki for more information about record locking:

http://m204wiki.rocketsoftware.com/index.php/Record_level_locking_and_concurrency_control

http://m204wiki.rocketsoftware.com/index.php/Defining_the_runtime_environment_(CCAIN)#Resource_locking

Refer to Chapter 2 for more information about using HLI calls on IFSTRT threads. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of the HLI calls shown in the examples.

Locking facility

Model 204 controls user access to file resources with a facility called resource locking. Record locking prevents conflicting, simultaneous use of records when several users have access to the same files or groups and protects against the concurrent updating of certain data by different users.

With locking, Model 204 allows a particular resource or record to be shared between users or locks access to a resource or record so that it can be used exclusively by a user.

Locking at the thread level

Model 204 locks resources and records for HLI processing at the thread level (for an IFSTRT thread and for an IFDIAL thread).

A resource or record that is shared by more than one thread is locked by each thread whether those threads are used in the same host language program, such as in a multithreaded IFAM4 job, or in different host language programs, for example, in two different IFAM2 jobs.

Enqueuing actions

For HLI functions that lock resources or records, if enqueuing actions are successful, Model 204 locks the resource or record in either of the following modes:

Mode	Description
SHR	In share mode, a lock allows one or more users to perform retrieval functions on a file, record set, or record. Any number of users can have shared control of a file, record set, or record concurrently.
EXC	In exclusive mode, a lock allows a single user to update a file, record set, or record. An exclusive lock is not compatible with other exclusive locks nor with any shared locks. See “EXC lock on the current record” on page 110 for information about different types of exclusive locks on records.

Most HLI functions automatically enqueue on the resources or records to which they refer.

For example, retrieval functions, such as IFFIND or IFFAC, enqueue in SHR mode, which allows simultaneous retrievals of records by different users but prohibits updating of those same records concurrently by other users. File maintenance functions such as IFPUT or IFUPDT enqueue in EXC mode, which locks out all other references.

In addition to the automatic enqueuing facilities, you can explicitly enqueue or dequeue on resources and records in your host language application program. See Table 9-1 on page 118 for a listing of HLI functions and enqueuing actions.

Getting control of a resource or a record

If control of a resource or record cannot be gained, Model 204 either immediately returns control to the application program with a function completion code (RETCODE) of 3 or waits a certain number of seconds.

If a wait occurs, Model 204 then attempts enqueuing again and if the resource or record is still unavailable, returns control to the application program with a function completion code of 3.

For all functions except IFCHKPT, a completion return code of 3 indicates that control of the resource or record could not be obtained. For calls that enqueue on a resource or record, code your host language program to test for a return code of 3 and, if desirable, to reissue the call.

Refer to Chapter 10 for a coding example that uses the Model 204 return code to test for locking conflicts.

Specifying wait time within system limits

You can specify a wait time for any of the following enqueuing calls: IFOPENX, IFFNDX, IFGETX, IFMOREX, IFENQL, and IFENQ.

Caution: Although there is no restriction on the wait time you can specify, setting the time to a very large number might exceed the system's wait-time limit; if the system's time limit is exceeded, a program ABEND occurs.

See "Locking functions" on page 116 for more information about wait times for HLI calls. Check with your Model 204 system administrator for information about system wait times.

Releasing a resource or record

With HLI functions that dequeue (release) locks on resources or records, dequeuing actions are performed immediately; there is no wait time.

Locking behavior of IFSTRT calls

The remaining sections describe the locking behavior of HLI calls that are issued on an IFSTRT thread.

Guidelines to avoid locking conflicts

Design your HLI application to enqueue on the fewest possible resources or records, as needed. Release resources or records as often as possible, when they are no longer needed.

Code your HLI application to use a record locking strategy that avoids logical inconsistencies, but that allows resources or records to be shared. This guideline applies in particular to HLI jobs running in IFAM2 (which supports multiple users) and to multithreaded IFAM4 applications (where conflicts can occur between threads in the same HLI job).

File locking

A file is a resource. Model 204 enqueues a file when an HLI application issues a call to open a file, with IFOPEN or IFOPENX. IFOPEN enqueues a file in SHR mode, while IFOPENX enqueues a file in EXC mode.

Caution: IFOPENX locks out all other Model 204 users and threads from accessing a file. In IFAM2, IFOPENX prevents any SOUL request or other HLI application from processing against the opened file.

Read-only file access in IFAM2 and IFAM4

For certain HLI applications that use IFSTRT threads, the IFSTRT thread type might determine the type of file access. For IFAM2 and IFAM4 jobs, the following IFSTRT thread types are available:

Thread type	Allows...
0	Read-only access (regardless of password used)
1	Updates (password privileges determine type of access)
2	Multiple-cursor functionality (password determines access)

In IFAM2 and IFAM4, a read-only IFSTRT thread allows SHR access to a file. You can use a password that allows either read-only or update privileges, but you cannot issue calls that perform update operations.

For example, if you set the IFSTRT thread indicator to read-only (THRD_IND is 0), Model 204 rejects a call at execution time that attempts to perform an update operation such as IFPUT and returns an error completion code (RETCODE) of 40.

Note: To update files in an IFAM2 or IFAM4 job, you must issue IFSTRT with thread update privileges (THRD_IND is 1 or 2) and you must enter file or group passwords that have update privileges.

Using a password with update privileges

HLI application programs that share the same copy of the HLI Model 204 service program, such as in IFAM2, can share files regardless of password privileges.

Operating system enqueuing

Between HLI applications that use different copies of Model 204, a file resource might be locked. If an HLI application opens a file using a password that allows updating privileges, an attempt to open that file by any other HLI application that uses a different copy of Model 204 is unsuccessful.

The following table summarizes file locking behavior between HLI applications. Each HLI application listed on the left opens a file using a password with update privileges. The HLI applications listed on the right are locked from accessing the open file.

Open file w/ update privileges:	File resource is locked to...
IFAM1	Other IFAM1; IFAM2; IFAM4
IFAM2	IFAM1; IFAM4
IFAM4	IFAM1; IFAM2; other IFAM4

Model 204 performs either of the following actions depending on which type of HLI job attempts to open a locked file.

If the file resource is locked to...	Then Model 204 issues...
IFAM1 application	WAITING FOR ACCESS message and the IFAM1 application waits for the file.
IFAM2 or IFAM4 application	FILE IS IN USE message and returns an IFOPEN completion code of 260 (an error).

In IFAM1 and IFAM4, the same file can be defined (DISP=SHR) in different HLI jobs. Model 204 performs resource locking at the operating-system level to control access to files that are shared by more than one copy of the HLI Model 204 service program in IFAM1 and IFAM4.

Note that enqueuing at the operating-system level is distinct from file-level enqueuing with IFOPEN and IFOPENX.

Refer to the Rocket Model 204 documentation wiki for more information about file resource locking at the operating-system level:

http://m204wiki.rocketsoftware.com/index.php/Defining_the_runtime_environment_(CCAIN)#Resource_locking

Record locking on found sets

Model 204 adheres to a specific set of rules for record locking behavior. The record locking behavior of the following HLI functions, which create found sets, is described in the following sections:

HLI function	Equivalent SOUL command
IFFAC	FIND AND COUNT
IFFIND	FIND
IFFDV	FIND VALUES
IFFNDX	FIND EXCLUSIVE
IFFWOL	FIND WITHOUT LOCKS

The record locking behavior of the IFDSET function, which deletes a found set, is described on page 110.

IFFAC and IFFIND lock in SHR mode

Both the IFFAC (FIND AND COUNT) and IFFIND (FIND) functions immediately lock the set of found records in SHR mode. If the locking is successful, none of the records in the found set can be updated by another user or thread until the records are released.

On a single cursor IFSTRT thread, an IFFIND that is executed in a loop releases the old found set as soon as the function is reexecuted. The final set selected by the IFFIND remains locked (in SHR mode) until the end of the transaction. Also, on a single cursor IFSTRT thread, each record is removed from the found set and the SHR lock released as it is processed by IFGET.

On a multiple cursor IFSTRT thread, records are held until explicitly released by a call to IFRELR or IFRELA, or until the entire transaction has been completed.

Note: When updating records after using IFFIND, you are moving from a SHR lock to an EXC lock, and the update fails, if any other user has obtained a SHR lock on the records. This is likely to occur in a busy system.

IFFDV locks a value set

The IFFDV (FIND ALL VALUES) function operates in the following manner, depending on the field attribute:

- If the field has the FRV attribute, the value set is locked in EXC mode until all values are collected. The EXC lock is released after all values are found.
- If the field has the ORDERED attribute, no locking is performed.

IFFNDX locks in EXC mode

The IFFNDX (FIND EXCLUSIVE) function enqueues in EXC mode to lock a set of records.

You can use IFFNDX to lock a record set for loop processing. However, when using IFFNDX, concurrence is reduced, because records are exclusively locked; none of the records in the found set can be retrieved or updated by another user or thread. Records found using IFFNDX are held in EXC status until they are released.

Issue IFRELR, IFRELA, or IFCMTR to explicitly release records locked in EXC mode by IFFNDX.

Note the following considerations when using IFFNDX:

- Because IFFNDX fails if any of the records that it needs are locked in SHR or EXC mode by other users, it has a good chance of failing in busy systems.

  To update a large set of records without locking the entire set, place the records to be updated on a list and then issue the IFUPDT on each record as you update it.

- IFFNDX guarantees that subsequent record updates succeed, because the records are already exclusively locked.

In the following example, a host language application exclusively locks two record sets. The first IFFNDX prevents access to TOTAL PREMIUM in the CLIENTS file while the corresponding VEHICLE PREMIUMs in the VEHICLES file are being changed. The application updates both files. This example shows HLI processing on a multiple cursor IFSTRT thread.

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE              PIC 9(5) COMP SYNC.
    05  DIR                  PIC 9(5) COMP SYNC VALUE "1".
01  WK-VARS COMP SYNC.
    05  FIND-SWITCH          PIC 9 VALUE ZERO.
    05  SET-SWITCH           PIC 9 VALUE ZERO.
    05  VPREM-AMT            PIC 9(6) VALUE ZERO.
    05  TOTAL-PREM-AMT       PIC 9(7) VALUE ZERO.
    05  VTOT-UPDATES         PIC 9(5) VALUE ZERO.
    05  CTOT-UPDATES         PIC 9(5) VALUE ZERO.
01  CLIENT-FILE-INFO.
    05  CLIENT-FILE-NAME     PIC X(8) VALUE "CLIENTS;".
    05  CLIENT-FILE-PASSW    PIC X(9) VALUE "CUPDATES;".
01  VEHICLE-FILE-INFO.
    05  VEHICLE-FILE-NAME    PIC X(9) VALUE "VEHICLES;".
    05  VEHICLE-FILE-PASSW   PIC X(9) VALUE 'VUPDATES;".
01  FINDX-CLIENTS.
    05  FXSPEC-CLIENTS       PIC X(38) VALUE
        "IN CLIENTS FD;POLICY NO=100015;RECTYPE=POLICYHOLDER;".
    05  WAIT-TIME            PIC 99 VALUE 30.
    05  FXNAME-CLIENTS       PIC X(7) VALUE "FXCLIENTS;".
    05  END-CALL             PIC X(4) VALUE "END;".
01  FINDX-VEHICLES.
    05  FXSPEC-VEHICLES      PIC X(20) VALUE
        "IN VEHICLES FD;OWNER POLICY=100015;".
    05  WAIT-TIME            PIC 99 VALUE 30.
    05  FXNAME-VEHICLES      PIC X(7) VALUE "FXVEHICLES;".
    05  END-CALL             PIC X(4) VALUE "END;".
01  COUNT-VARS.
    05  COUNT                PIC 9(5).
    05  CLIENT-COUNT         PIC 9(5) VALUE ZERO.
    05  VEHICLE-COUNT        PIC 9(5) VALUE ZERO.
01  VCURSOR-PARMS.
```

```
    05  VCURSPEC              PIC X(14) VALUE "IN FXVEHICLES;".
    05  VCURNAME              PIC X(8) VALUE "VCURSOR;".
01  CCURSOR-PARMS.
    05  CCURSPEC              PIC X(13) VALUE "IN FXCLIENTS;".
    05  CCURNAME              PIC X(8) VALUE "CCURSOR;".
01  VEHICLE-REC.
    05  OWNER-POLICY-NO       PIC 9(5) VALUE ZERO.
    05  VEHICLE-ID-NO         PIC 9(2) VALUE ZERO.
    05   VEHICLE-PREMIUM-AMT     PIC ((6) VALUE ZERO.

01  CLIENT-REC.
    05  CLIENT-POLICY-NO      PIC (5) VALUE ZERO.
    05  CLIENT-TYPE           PIC X(15) VALUE SPACES.
    05  CLIENT-PREMIUM-TOTAL  PIC 9(7) VALUE ZERO.
01  VDATA-SPEC                PIC X(55) VALUE
    "EDIT(OWNER POLICY,VIN,VEHICLE PREMIUM)(A(5),A(2),A(6));".
01  CDATA-SPEC                PIC X(55) VALUE
    "EDIT(POLICY NO,RECTYPE,TOTAL PREMIUM)(A(5),A(15),A(7));".
•
•
•
PROCEDURE DIVISION.
CALL "IFSTRT"...
*
* OPEN DATA FILES
*
PERFORM OPEN-FILES UNTIL RETCODE IS NOT EQUAL TO ZERO OR
  SET-SWITCH IS NOT EQUAL TO ZERO.
IF RETCODE IS NOT EQUAL TO ZERO THEN
  GO TO ERROR-RTN.

*
* FIND RECORDS TO BE UPDATED
*
PERFORM FIND-RTN UNTIL RETCODE IS NOT EQUAL TO ZERO OR
  FIND-SWITCH IS NOT EQUAL TO ZERO.
IF RETCODE IS NOT EQUAL TO ZERO THEN
  GO TO ERROR-RTN.
*
* UPDATE VEHICLES FILE
*
PERFORM VEHICLES-UPDATE UNTIL RETCODE IS NOT EQUAL TO ZERO OR
  VEHICLE-COUNT IS EQUAL TO ZERO.
DISPLAY "TOTAL VEHICLE RECORDS UPDATED IS " VTOT-UPDATES.
IF RETCODE IS NOT EQUAL TO ZERO THEN
  GO TO ERROR-RTN.
*
MOVE ZERO TO SET-SWITCH.
PERFORM CURSOR-RTN UNTIL RETCODE IS NOT EQUAL TO ZERO OR
  SET-SWITCH IS NOT EQUAL TO ZERO.
```

```
IF RETCODE IS NOT EQUAL TO ZERO THEN
  GO TO ERROR-RTN.

*
* UPDATE CLIENTS FILE
*
PERFORM CLIENTS-UPDATE UNTIL RETCODE IS NOT EQUAL TO ZERO OR
  CLIENT-COUNT IS EQUAL TO ZERO.
DISPLAY "TOTAL CLIENT RECORDS UPDATED IS " CTOT-UPDATES.
IF RETCODE IS NOT EQUAL TO ZERO THEN
  GO TO ERROR-RTN.
*
MOVE ZERO TO SET-SWITCH.
PERFORM END-UPDATES UNTIL RETCODE IS NOT EQUAL TO ZERO OR
  SET-SWITCH IS NOT EQUAL TO ZERO.
IF RETCODE IS NOT EQUAL TO ZERO THEN
  GO TO ERROR-RTN.
PERFORM END-JOB-RTN.
*
*SUBROUTINE TO OPEN VEHICLES AND CLIENTS FILES
*
OPEN-FILES.
  CALL "IFOPEN" USING RETCODE,CLIENT-FILE-INFO.
  CALL "IFOPEN" USING RETCODE,VEHICLE-FILE-INFO.
  MOVE 1 TO SET-SWITCH.

* SUBROUTINE TO FIND RELATED RECORDS IN BOTH FILES
* HOLDS FILES EXCLUSIVELY
*
FIND-RTN.
  CALL "IFFNDX" USING RETCODE,FXSPEC-CLIENTS,WAIT-TIME,
    FXNAME-CLIENTS,END-CALL.
  CALL "IFCOUNT" USING RETCODE,COUNT,FXNAME-CLIENTS.
  MOVE COUNT TO CLIENT-COUNT.
  DISPLAY "TOTAL CLIENT RECORDS FOUND IS " CLIENT-COUNT.
  CALL "IFFNDX" USING RETCODE,FXSPEC-VEHICLES,WAIT-TIME,
    FXNAME-VEHICLES,END-CALL.
  CALL "IFCOUNT" USING RETCODE,COUNT,FXNAME-VEHICLES.
  MOVE COUNT TO VEHICLE-COUNT.
  DISPLAY "TOTAL VEHICLE RECORD FOUND IS " VEHICLE-COUNT.
  CALL "IFOCUR" USING RETCODE,VCURSPEC,VCURNAME.
  MOVE 1 TO FIND-SWITCH.

CURSOR-RTN.
  CALL "IFCCUR" USING RETCODE,VCURSOR.
  CALL "IFOCUR" USING RETCODE,CCURSPEC,CCURNAME.
  MOVE "1" TO SET-SWITCH.
*
* SUBROUTINE TO UPDATE PREMIUM AMOUNT IN VEHICLES RECORDS
*
```

```
VEHICLES-UPDATE.
  CALL "IFFTCH" USING RETCODE,VEHICLE-REC,DIR,VCURNAME,
    VDATA-SPEC.
  MOVE VEHICLE-PREMIUM-AMT TO VPREM-AMT.
  ADD 100 TO VPREM-AMT.
  ADD VPREM-AMT TO TOTAL-PREM-AMT.
  MOVE VPREM-AMT TO VEHICLE-PREMIUM-AMT.
  CALL "IFUPDT" USING RETCODE,VEHICLE-REC,VCURNAME,
  VDATA-SPEC.
  CALL "IFCMMT" USING RETCODE.
  MOVE ZEROS TO VEHICLE-REC.
  ADD 1 TO VTOT-UPDATES.
  SUBTRACT 1 FROM VEHICLE-COUNT.
*
* SUBROUTINE TO UPDATE TOTAL PREMIUM/RELATED CLIENTS RECORD
*
CLIENTS-UPDATE.
  CALL "IFFTCH" USING RETCODE,CLIENT-REC,DIR,CCURNAME,
    CDATA-SPEC.
  MOVE TOTAL-PREM-AMT TO CLIENT-PREMIUM-TOTAL.
  CALL "IFUPDT" USING RETCODE,CLIENT-REC,CCURNAME,CDATA-SPEC.
  CALL "IFCMMT" USING RETCODE.

*
  MOVE ZEROS TO CLIENT-POLICY-NO.
  MOVE SPACES TO CLIENT-TYPE.
  MOVE ZEROS TO CLIENT-PREMIUM-TOTAL.
  ADD 1 TO CTOT-UPDATES.
  SUBTRACT 1 FROM CLIENT-COUNT.
*
* SUBROUTINE TO END THE PROCESSING LOOP
*
END-UPDATES.
  CALL "IFCCUR" USING RETCODE,CCURSOR.
  CALL "IFRELA" USING RETCODE.
  MOVE "1" TO SET-SWITCH.
*
*  TO HERE FOR ERROR-RTN AND END-JOB-RTN
•
•
•
```

IFFWOL does not lock records

The IFFWOL (FIND WITHOUT LOCKS) function executes an IFFIND without obtaining any record locks.

The found set of records is indistinguishable from a list, except that the record set is referenced with “IN label” syntax. Refer to the *Rocket Model 204 Host*

Language Interface Reference Manual for a detailed description of IFFWOL syntax.

Caution when using IFFWOL

You can use the IFFWOL function to solve specific performance problems. However, use IFFWOL with caution to avoid logical inconsistencies.

When using IFFWOL, design your application to take into account the following usage issues:

- The logical integrity of data is at risk when:
  - Another user is in the middle of changing values that are related.
  - Another user deletes or changes the field that caused the record to be found.
- The thread that issues IFFWOL might encounter data that is temporarily physically inconsistent. Because the thread does not hold any locks and cannot prevent other threads from updating, updates can occur while the IFFWOL thread is examining the record.

  The following error conditions can occur when this happens:
  - SICK RECORD messages are sent when extension records get deleted. In this case, the record is not really sick; it just temporarily appears that way to Model 204.
  - NONEXISTENT RECORD messages are sent when entire records get deleted.

When to use IFFWOL

An advantage to using IFFWOL over any of the other find functions (which lock records) is that IFFWOL never fails. Examples of appropriate use of the IFFWOL function include:

- When there is one user or thread at a time per record, for example, scratch records or bank teller applications, where an account is usually modified by one teller at a time.
- Report programs in a heavy update environment.

Examples of inappropriate uses of the IFFWOL function include:

- Report program in a heavy delete environment. In this case, IFFWOL results in many NONEXISTENT RECORD messages.
- Retrievals in which the selection criteria can be changed by other users.
- Reuse record number (RRN) files, with the possible exception of scratch files keyed on the user ID.

IFDSET locks a record set in EXC mode

The IFDSET (DELETE SET) function temporarily locks the set of records to be deleted in EXC mode before deletion occurs. The locking does not succeed, if any other user or thread has access to any of the records either in SHR or EXC mode.

Note: Once a record has been deleted, the EXC lock on the record is released, because the record no longer exists in the file.

SHR lock on the current record

The following retrieval functions lock the current record in SHR mode:

HLI function	Equivalent SOUL command
IFFRN	FOR RECORD NUMBER
IFOCC, IFCTO	COUNT OCCURRENCES
IFGET	GET
IFMORE	MORE

EXC lock on the current record

Two of the HLI retrieval calls, IFGETX and IFMOREX, and all the file updating calls enqueue in EXC mode and lock the current record with a SRE (single record enqueue) lock. Model 204 applies the SRE lock regardless of whether the record set is being held in SHR or EXC mode, or is unlocked.

Model 204 enqueues with the SRE lock on records from transaction back out (TBO) and non-TBO files alike. For TBO files with the LPU option set, Model 204 applies an additional EXC lock on updated records.

Single record enqueue (SRE) locks

The following retrieval calls lock the current record in EXC mode with a SRE lock:

- IFGETX (GET EXCLUSIVE)
- IFMOREX (MORE EXCLUSIVE)

The following updating calls lock the current record in EXC mode with a SRE lock before modifying the record:

- IFDALL (DELETE ALL)
- IFDREC (DELETE RECORD)
- IFDVAL (DELETE VALUE)
- IFPUT (PUT)

- IFUPDT (UPDATE)

The following calls, if successful, lock the newly created record in EXC mode with an SRE lock:

- IFBREC (BEGIN RECORD)
- IFSTOR (STORE RECORD)

On a single cursor IFSTRT thread, Model 204 holds the SRE lock on the current record until the HLI thread enqueues on another record.

On a multiple cursor IFSTRT thread, Model 204 holds the SRE lock on a cursor's current record until the HLI program issues a call to IFRELA or IFCMTR to release locks, or until the cursor position is modified by one of the following calls.

Call	Function
IFFTCH	Moves the cursor to the next logical record in the set.
IFFRN	Points to specified a record number.
IFSTOR	Changes the cursor position to the newly created record.
IFCCUR	Closes the cursor.

Note: When using these functions inside a processing loop in the host language program, the current record remains locked until it passes through the loop unless it has been released (as described above) or deleted. If the record has been deleted, the EXC lock on the record is released, because the record no longer exists in the file.

Lock pending updates (LPU) locks

Model 204 provides a special file facility for TBO files called lock pending updates (LPU), which prevents updated records in one transaction from being used by other applications until the transaction ends.

When processing records from a TBO file with the LPU option set, the first updating call locks the current record in EXC mode with an additional LPU lock and adds the record to a set of updated locked records, called the pending update pool.

The following updating calls lock the current record in a pending update pool with the additional EXC lock:

- IFDALL
- IFDVAL
- IFDREC
- IFPUT

- IFSTOR
- IFUPDT

Model 204 locks the record in the pending update pool with the additional EXC lock until the transaction is committed. The record is locked until the end of the transaction when the entire pending update pool is released. See page 114 for information about using IFCMMT to release LPU locks.

Note: Lock pending updates is an option of the FOPT parameter, which is enabled or disabled on a file by file basis. Using files that have the LPU option set ensures logical file consistency.

See page 104 for a description of IFFNDX locking behavior. Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/FOPT_parameter and http://m204wiki.rocketsoftware.com/index.php/File_integrity_and_recovery) for more information about the FOPT parameter.

Record locking: sample processing loops

The following coding sequence shows HLI processing against a TBO file (the LPU option is set). The program creates a found set with a SHR lock and updates records.

Call	Locking behavior
1. IFFIND or IFFAC	Enqueues SHR lock on found record set
2. Loop:	
IFFTCH	Dequeues previous SRE lock (if any)
IFUPDT	Enqueues current record with SRE lock (EXC-1)
	Enqueues current record with LPU lock (EXC-2)
End loop	
3. IFCMMT	Releases all SRE and LPU locks (EXC-1, EXC-2 locks)
4. IFRELR	Releases SHR lock on record set

The following coding sequence shows HLI processing against a TBO file (the LPU option is set). The program creates a found set with an EXC lock and updates records.

Call	Locking behavior
1. IFFNDX	Enqueues EXC lock on found record set
2. Loop:	
IFFTCH	Dequeues previous SRE lock (if any)
IFUPDT	Enqueues current record with SRE lock (EXC-1)
	Enqueues current record with LPU lock (EXC-2)
End loop	

Call	Locking behavior
3. IFCMMT	Releases all SRE and LPU locks (EXC-1, EXC-2 locks)
4. IFRELR	Releases EXC lock on record set

Releasing record locks

The following functions allow you to explicitly remove certain types of record locks:

HLI function	Equivalent SOUL command
IFRELA	RELEASE ALL RECORDS
IFRELR	RELEASE RECORDS
IFCMMT	COMMIT
IFCMTR	COMMIT RELEASE

The types of record locks released by each function is described in the following sections.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a detailed description of the HLI functions and their syntax. Refer to Chapter 15 for more information about managing transactions.

IFRELA releases all locks

IFRELA (RELEASE ALL RECORDS) empties all record sets and terminates all locks held by the thread. IFRELA also performs the following actions:

- Releases all SHR and EXC locks placed on records
- Clears all lists and the results of all IFSORT functions
- Sets the current record number to -1

Note: Use IFRELA with caution. After IFRELA is processed in a host language program, no records are available for processing. Issue a call to IFRELA at the end of a logical phase.

IFRELR releases a record set lock

IFRELR (RELEASE RECORDS) empties the records in a single found set and terminates the lock on those records. You can use IFRELR to release the SHR lock placed on records by IFFIND or IFFAC and the EXC lock placed on a record set by IFFNDX.

You can also use IFRELR to release IFSORT records in the scratch file.

When IFRELR refers to an IFSORT sorted set, the CCATEMP space, which was occupied by the temporary sorted record copies, is released.

Note: Sorted records released in this manner are no longer available to the host language program. However, the original found set or list from which the sorted set was built is not affected.

IFCMMT releases LPU locks

If the host language program is processing against a TBO file, IFCMMT (COMMIT) releases the LPU exclusive lock on updated records. IFCMMT also releases a single record enqueue (SRE) on the current record if one exists, for example, as a result of a call to IFUPDT or IFPUT. IFCMMT ends the current transaction and dequeues checkpoints.

When updating records, you can minimize record locking conflicts by issuing frequent calls to IFCMMT.

A call to IFCMMT inside a loop that processes each record ends the current update unit each time the loop is processed. This results in several short update units instead of one long update unit and is especially useful for minimizing conflicts on an LPU file where all updated records are enqueued in EXC mode until the update unit ends. Record set locks obtained by IFFIND, IFFAC, and IFFNDX are not affected by IFCMMT.

Example of using IFCMMT

The following COBOL example shows the call to IFCMMT issued inside a record processing loop. The program finds and prints a count of all records whose last name is NELSON. The MONTH field of each record in the found set is changed to November (value is NOV).

In this example, the following conditions apply:

- IFSTRT starts a multiple cursor thread that allows updating
- IFOPEN uses a password that allows updating privileges
- IFUPDT enqueues the current record in EXC mode
- IFCMMT releases the LPU lock on the current record

```
WORKING-STORAGE SECTION.
01  COUNTERS.
    05  0FIND-TOT   PIC 9(5) VALUE ZERO.
    05  UPDATE-TOT  PIC 9(5) VALUE ZERO.
01  CALL-ARGS.
    05  RETCODE     PIC 9(5) COMP SYNC.
    05  DIR         PIC 9 COMP SYNC VALUE "1".
01  FIND-CRITERIA.
    05  FDSPEC      PIC X (13) VALUE "LNAME=NELSON;".
```

```
    05  COUNT       PIC 9(5) COMP SYNC.
    05  FDLNAME     PIC X(8) VALUE "FDLNAME;".
    05  END-CALL    PIC X(4) VALUE "END;".
01  CURSOR-PARMS.
    05  CURSPEC     PIC X(11) VALUE "IN FDLNAME;".
    05  CURNAME     PIC X(8) VALUE "CRLNAME;".
01  DATA-SPEC       PIC X(74) VALUE
    "EDIT(FNAME,LNAME,ADDRESS,YEAR,MONTH,DAY)
    (A(10),A(20),A(40),A(2),A(3),A(2));".
01  WORK-REC.
    05  FIRST-NAME  PIC X(10).
    05  LAST-NAME   PIC X(20).
    05  ADDRESS     PIC X(40).
    05  DATE-YEAR   PIC X(2).
    05  DATE-MONTH  PIC X(3).
    05  DATE-DAY    PIC X(2).
01  WORK-VAL.
    05  MONTH1      PIC X(3) VALUE "JAN".
•
•
•
    05  MNTH11      PIC X(3) VALUE "NOV".
•
•
•
PROCEDURE DIVISION.
•
•
•
CALL "IFSTRT" ...
CALL "IFOPEN" ...
•
•
•

* FIND AND COUNT RECORDS WHOSE LAST NAME IS NELSON
*
CALL "IFFAC" USING RETCODE,FDSPEC,COUNT,FDLNAME,END-CALL.
MOVE COUNT TO FIND-TOT
DISPLAY "TOTAL RECORDS FOUND IS " FIND-TOT
IF RETCODE IS NOT EQUAL TO ZERO THEN
  GO TO END-RTN
*
ELSE
* UPDATE NELSON RECORDS
*
  CALL "IFOCUR" USING RETCODE,CURSPEC,CURNAME.
  PERFORM UPDATE-RTN UNTIL RETCODE IS NOT EQUAL TO ZERO OR
      FIND-TOT IS EQUAL TO ZERO
  IF RETCODE IS NOT EQUAL TO ZERO THEN
```

```
    GO TO END-RTN
  ELSE
*
* PRINT THE UPDATE TOTAL
*
    CALL "IFCCUR" USING RETCODE,CURNAME
    DISPLAY "TOTAL RECORDS UPDATED IS " UPDATE-TOT.
*
GO TO END-RTN.
*
* SUBROUTINE TO CHANGE MONTH TO NOV AND UPDATE FILE
  COMMITS THE UPDATE UNIT
*
UPDATE-RTN.
  CALL "IFFTCH" USING RETCODE,WORK-REC,DIR,CURNAME,DATA-SPEC.
  MOVE MNTH11 TO DATE-MONTH.
  CALL "IFUPDT" USING RETCODE,WORK-REC,CURNAME,DATA-SPEC.
  CALL "IFCMMT" USING RETCODE.
  ADD 1 TO UPDATE-TOT.
  SUBTRACT 1 FROM FIND-TOT.
*
* SUBROUTINE TO END THE TRANSACTION
*
END-RTN.
  IF RETCODE IS EQUAL TO ZERO THEN
    NEXT SENTENCE
  ELSE
    PERFORM ERROR-RTN.
  CALL "IFCLOSE"...
  CALL "IFFNSH"...
  •
  •
  •
```

IFCMTR releases all locks and ends a transaction

The IFCMTR (COMMIT RELEASE) function performs all the operations of both IFCMMT and IFRELA.

Note: After IFCMTR is processed, there is no current record. To avoid confusing results, Technical Support recommends that you issue a call to IFCMTR at the end of a logical processing step.

Locking functions

Table 9-1 summarizes the following enqueuing and dequeuing actions for the HLI functions that lock:

Files or groups
Record sets

Individual records
Arbitrary resources

Codes used in the table

Table 9-1 use the following codes:

Code	Meaning
Thread:	
mc	Multiple cursor IFSTRT thread
st	Single cursor IFSTRT thread
Action:	
ENQ	Enqueues resource
DEQ	Dequeues resource
	Note: Some HLI functions enqueue or dequeue only, while others first dequeue and then enqueue. Those functions that perform both operations in this manner only when used on a single cursor IFSTRT thread. On a multiple cursor IFSTRT thread, except for IFFRN, an HLI function either enqueues or dequeues.
Mode:	
SHR	Shares lock on resource
EXC	Exclusive lock on resource (for individual records, SRE or LPU lock
Times:	
n	User-specified number of times to try enqueuing
0	No wait
	Note: Except for IFOPENX and IFENQ, which is wait time in seconds, Times is the number of times to attempt enqueuing on the resource before returning a completion code (RETCODE) of 3 to the HLI application. If times is greater than zero, Model 204 waits for three seconds (or 10 seconds for IFOPEN) before the next attempt.
	The total number of times to try equates to a total wait time in seconds, where:
	IFOPEN total wait time is the total number of tries multiplied by 10 seconds.
	For all other calls that specify a wait time except IFOPEN: total wait time is the total number of tries multiplied by 3 seconds.

Table 9-1. HLI locking functions

HLI function	Thread	Action	Mode	Times	Resource
Locking files or groups					
IFCLOSE	mc, st	DEQ	—	0	All previously enqueued: • Records • Record sets and lists • Files
IFDELF	mc, st	ENQ	EXC	0	Specified file
IFDFLD	mc, st	ENQ	EXC	0	Specified file
IFNFLD	mc, st	ENQ	EXC	0	Specified file
IFRFLD	mc, st	ENQ	EXC	0	Specified file
IFOPEN	st	1. DEQ	—	0	All previously enqueued: • Records • Record sets and lists
	mc, st	2. ENQ	SHR	*2*	Specified file or group **Note:** The wait time is 10 seconds each time.
IFOPENX	st	1. DEQ	—	0	All previously enqueued: • Records • Record sets and lists
	mc, st	2. ENQ	EXC	*n*	Specified file or files ***Note:*** *n* is a user-specified wait time in seconds.
Locking record sets					
IFBOUT	mc, st	DEQ	—	0	• LPU locks • SRE exclusive locks
IFCMMT	mc, st	DEQ	—	0	• LPU locks • SRE exclusive locks
IFCMTR	mc, st	DEQ	—	0	• Records held in EXC lock, previously updated • Record sets and lists, previously enqueued (EXC or SHR)
IFDEQL	st	DEQ	—	0	Record set on a specified list, previously enqueued
IFDSET	mc, st	ENQ	EXC	0	Record set
IFENQL	st	ENQ	SHR/EXC	*n*	Record set on a specified list

Table 9-1. HLI locking functions (Continued)

HLI function	Thread	Action	Mode	Times	Resource
IFFAC IFFACE	mc, st	ENQ	SHR	0	New record set
IFFIND IFFINDE	st	1. DEQ	—	0	Previously enqueued: • Record • Record set
	mc, st	2. ENQ	SHR	0	New record set
IFFNDX IFFNDXE	st	DEQ	—	0	Previously enqueued: • Record • Record set
IFRELA	mc	DEQ	—	0	All record sets and lists previously enqueued (SHR or EXC)
IFRELR	mc	DEQ	—	0	Specified record set or list, previously enqueued
IFSKEY IFSKYE	st	DEQ	—	0	Previously enqueued: • Record • Record set
IFSORT IFSRTE	st	DEQ	—	0	Previously enqueued: • Record • Record set
Locking individual records					
IFBREC	st	1. DEQ	—	0	Record previously enqueued
		2. ENQ	EXC	0	New record
IFCCUR	mc	DEQ	—	0	Record previously enqueued
IFCTO IFCTOE	st	1. DEQ	—	0	Record previously enqueued
		2. ENQ	SHR	0	Current record
IFDALL	st	1. DEQ	—	0	Record previously enqueued
	mc, st	2. ENQ	EXC	10	Current record (SRE lock)
		3. ENQ	EXC	0	Current record in LPU pending update pool
IFDREC	st	1. DEQ	—	0	Record previously enqueued
	mc, st	2. ENQ	EXC	10	Current record (SRE lock)
		3. ENQ	EXC	0	Current record in LPU pending update pool
IFDVAL	st	1. DEQ	—	0	Record previously enqueued
	mc, st	2. ENQ	EXC	10	Current record (SRE lock)
		3. ENQ	EXC	0	Current record in LPU pending update pool

Table 9-1. HLI locking functions (Continued)

HLI function	Thread	Action	Mode	Times	Resource
IFFRN IFFRNE	mc	1. DEQ 2. ENQ	— SHR	0 0	Record previously enqueued Current record
IFFTCH IFFTCHE	mc	DEQ	—	0	Record previously enqueued
IFGET IFGETE	st	1. DEQ 2. ENQ	— SHR	0 0	Record previously enqueued Next logical record
IFGETX IFGETXE	st	1. DEQ 2. ENQ	— EXC	0 *n*	Record previously enqueued Next logical record
IFMORE IFMOREE	st	1. DEQ 2. ENQ	— SHR	0 0	Record previously enqueued Current record
IFMOREX IFMORXE	st	1. DEQ 2. ENQ	— EXC	0 *n*	Record previously enqueued Current record
IFOCC IFOCCE	mc	ENQ	SHR	0	Current record
IFPUT IFPUTE	st	1. DEQ 2. ENQ 3. ENQ	— EXC EXC	0 10 0	Record previously enqueued Current record (SRE lock) Current record in LPU pending update pool
IFSTOR	mc	1. DEQ 2. ENQ 3. ENQ	— EXC EXC	0 0 0	Record previously enqueued New record (SRE lock) New record in LPU pending update pool
IFUPDT IFUPDTE	mc	1. DEQ 2. ENQ	EXC EXC	10 0	Current record (SRE lock) Current record in LPU pending update pool
Locking arbitrary resources					
IFDEQ	mc, st	DEQ	—	0	Resource previously enqueued by IFENQ
IFENQ	mc, st	1. DEQ 2. ENQ	— SHR/EXC	0 *n*	User-specified source ***Note:*** *n* is a user-specified wait time in seconds.

10

Record Locking Conflicts

Overview

This chapter describes for the HLI programmer a typical locking conflict and provides an example of how to handle conflicts in a host language application. Refer to the guidelines for avoiding conflicts when you are coding your host language program.

For more information

Refer to Chapter 9 for a description of enqueuing and information about the locking behavior of individual HLI calls.

When a record locking conflict occurs

If Model 204 cannot lock a record, a locking conflict occurs. Record locking conflicts can occur when multiple users try concurrently to access the same records and attempt overlapping updating operations.

A conflict arises when one or more users are reading a file (in SHR mode) and another user attempts to update the file (enqueuing in EXC mode) or when two or more users attempt to perform file maintenance (each requiring EXC mode access) on the same records retrieved from a file.

For update operations from a host language application, locking guarantees an HLI thread exclusive control of a resource until the thread completes update processing on the resource. Conflicting requests are automatically delayed until exclusive control is released.

Model 204 locks at different levels

Model 204 resolves internal enqueuing conflicts by locking at the following levels:

- Files or groups
- Sets of records
- Single records
- Arbitrary resources

For more information about record locking, refer to Chapter 9.

Example of record locking conflict

In this example, a SOUL request and an IFAM2 host language application attempt to access the same Model 204 record at the same time. Because the IFAM2 application requires an exclusive (EXC) lock, this situation causes a record locking conflict.

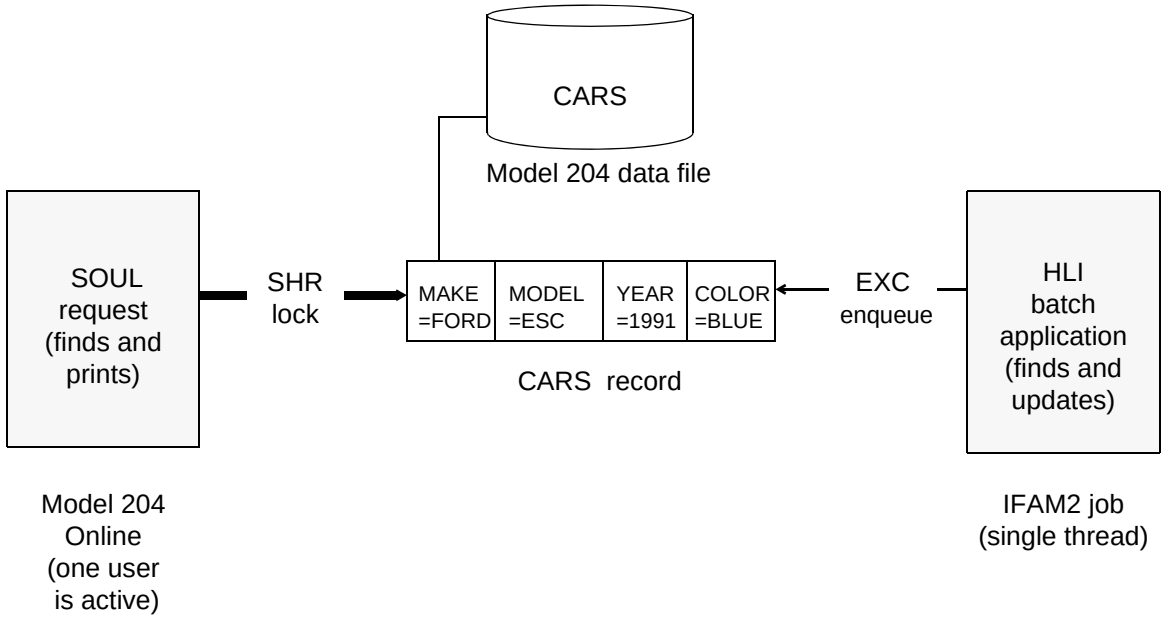
IFAM2 application requires an EXEC lock

Events occur in the following order:

1. The SOUL request begins processing and finds all the records in the CARS file that were made in 1991 and starts to print a report. (The procedure enqueues the found records in SHR mode.)
2. Then the host language application begins processing and finds all the FORD records in the CARS file, including some of the 1991 records selected by the SOUL request. The application then attempts an updating function that requires an EXC lock as it processes each record. However, the program cannot gain EXC access to any of the 1991 records.

Figure 10-1 illustrates the record locking conflict involving the SOUL procedure and the host language application.

Figure 10-1. Example of record locking conflict



SOUL request opens CARS with read-only privileges

The SOUL request begins processing first and opens the CARS file with read-only privileges (password is READS), selects the FORD records, and prints the information found in each record.

The SOUL request includes the following statements:

```
OPEN CARS
READS
BEGIN
1 FD MAKE=FORD
 YEAR=1991
2 FR 1
PRINT ALL INFORMATION
END
CLOSE CARS
```

Note: The records in the found set remain locked in SHR mode while they are being printed by the for loop.

IFAM2 application attempts to update CARS

The HLI IFAM2 batch application begins processing next and starts a multiple cursor IFSTRT thread, which allows update processing. The program opens the CARS file with update privileges (password is UPDATES) and selects the FORD records (using IFFAC).

The host language application starts a processing loop that performs the following functions:

- Fetches (IFFTCH) each record in the found set.
- Calls IFUPDT to update the record, changing the COLOR field to TAN.
- Writes an output record to a user-specified report file.
- Displays record counts.

However, as soon as the host language program attempts to perform the updating function on the first 1991 record in the found set, it enqueues on the record in EXC mode, as shown in Figure 10-1 on page 123. Model 204 returns a completion code (RETCODE) of 3, which indicates that the required EXC lock on the record could not be obtained.

The excerpts from a COBOL host language program, in the section “Sample host language error processing” on page 125, updates the FORD records.

Resolution of the locking conflict

The host language application shown in Figure 10-1 cannot access the 1991 records (which are locked in SHR mode by the SOUL procedure) until the SOUL request finishes processing or explicitly releases the records.

When the SHR lock has been released, the host language application can access the record in EXC mode and update records with IFUPDT.

Handling record locking conflicts

Specifying an action when a record locking conflict occurs

Model 204 allows the user to specify the action to be taken if, after an initial record locking attempt and subsequent wait, an effort to lock a set of records is still unsuccessful.

In SOUL, two special forms of ON units, the ON RECORD LOCKING CONFLICT and ON FIND CONFLICT, can be invoked to specify the action to be taken for a record locking conflict. A SOUL PAUSE statement can be used to cause the request to wait a specified time and then to retry the statement that caused the evaluation of the ON unit.

In your host language program, you can use the IFERLC call to help determine the cause of the record locking conflict and to specify an action to be taken.

Sample host language error processing

The following COBOL program excerpts show error processing in a host language program using an ERROR-RTN subroutine.

The program tests the Model 204 completion return code (RETCODE) after each HLI call. In this example, if the return code is not equal to zero, the program transfers control to ERROR-RTN and stops processing.

For a return code of 3, the program displays a record locking conflict message. For all nonzero return codes, the program displays an error message with the name of the HLI function call and the return code value. The error routine also calls IFGERR and displays the text of the Model 204 error message.

```
DATA DIVISION.
FILE SECTION.
FD  UPDATE-REPORT
    LABEL RECORDS ARE STANDARD
    BLOCK CONTAINS 0 RECORDS
    DATA RECORD IS CAR-REC-OUT.
01  CAR-REC-OUT.
    05  CAR-MAKE-OUT        PIC X(5).
    05  FILLER              PIC X(10).
    05  CAR-MODEL-OUT       PIC X(5).
    05  FILLER              PIC X(10).
    05  CAR-YEAR-OUT        PIC X(2).
    05  FILLER              PIC X(10).
    05  CAR-COLOR-OUT       PIC X(3).
WORKING-STORAGE SECTION.

01  M204-CALL-ARGS.
    05  RETCODE             PIC 9(5) COMP SYNC.
    05  DIR                 PIC 9(5) COMP SYNC VALUE "1".
01  M204-ERROR.
    05  M204-CALL-ERROR     PIC X(8) VALUE SPACES.
    05  M204-RETCODE-ERROR  PIC 9(5) VALUE ZERO.
    05  M204-MSG-ERROR      PIC X(80) VALUE SPACES.
01  WK-VARS.
    05  WK-COLOR            PIC X(3) VALUE "TAN".
    05  FIND-TOT            PIC 9(5) VALUE ZERO.
    05  UPDATE-TOT          PIC 9(5) VALUE ZERO.
01  START-THREAD.
    05  LANG-IND            PIC 9(5) COMP SYNC.
    05  LOGIN               PIC X(12) VALUE "CARSUSER;UPDATES;".
    05  THRD-TYPE           PIC 9(5) VALUE "2" COMP SYNC.
    05  THRD-NO             PIC 9(5) COMP SYNC.
01  FILE-INFO.
    05  FILE-NAME           PIC X(10) VALUE "CARS;".
    05  PASSWORD            PIC X(6) VALUE "UPDATES;".
01  FIND-SPEC.
    05  FDSPEC              PIC X(10) VALUE "MAKE=FORD;".
```

```
    05  COUNT               PIC 9(5) COMP SYNC.
    05  FDNAME              PIC X(7) VALUE "FDNAME;".
    05  END-CALL            PIC X(4) VALUE "END;".
01  CURSOR-PARMS.
    05  CURSPEC             PIC X(10) VALUE "IN FDNAME;".
    05  CURNAME             PIC X(7) VALUE "CRNAME;".
01  WORK-REC.
    05  MAKE                PIC X(5) VALUE SPACES.
    05  MODEL               PIC X(5) VALUE SPACES.
    05  YEAR                PIC X(2) VALUE SPACES.
    05  COLOR               PIC X(3) VALUE SPACES.
01  CALL-ARGS.
    05  RETCODE             PIC 9(5) COMP SYNC.
    05  RECNUM              PIC X(11).
    05  USERNUM             PIC X(5).
    05  FILENAME            PIC X(8).
01  DATA-SPEC               PIC X(49) VALUE
    "EDIT(MAKE,MODEL,YEAR,COLOR)(A(5),A(5),A(2),A(3));".
•
•
•
PROCEDURE DIVISION.
•
•
•
* START A MULTIPLE CURSOR THREAD
*
CALL "IFSTRT" USING RETCODE,START-THREAD.
IF RETCODE IS NOT EQUAL TO ZERO THEN
  MOVE "IFSTRT  " TO M204-CALL-ERROR
  GO TO ERROR-RTN.
*
* OPEN CARS WITH UPDATE PRIVILEGES
*
CALL "IFOPEN" USING RETCODE,FILE-INFO.
IF RETCODE IS NOT EQUAL TO ZERO THEN
  MOVE "IFOPEN  " TO M204-CALL-ERROR
  GO TO ERROR-RTN.
*
* FIND AND COUNT FORD RECORDS
*
CALL "IFFAC" USING RETCODE,FDSPEC,COUNT,FDNAME,END-CALL.
IF RETCODE IS NOT EQUAL TO ZERO THEN
  MOVE "IFFAC   " TO M204-CALL-ERROR
  GO TO ERROR-RTN
*
ELSE
*UPDATE AND WRITE FORD RECORDS
*
  MOVE COUNT TO FIND-TOT
```

```
  DISPLAY "TOTAL RECORDS FOUND IS " FIND-TOT
  CALL "IFOCUR" USING RETCODE,CURSPEC,CURNAME
  MOVE "IFOCUR  " TO M204-CALL-ERROR
  PERFORM UPDATE-AND-WRITE UNTIL
    RETCODE IS NOT EQUAL TO ZERO OR
    FIND-TOT IS EQUAL TO ZERO.

IF RETCODE IS NOT EQUAL TO ZERO THEN
  GO TO ERROR-RTN
ELSE
*
* PRINT UPDATE TOTAL
*
  DISPLAY "TOTAL RECORD UPDATED IS " UPDATE-TOT
  CALL "IFCCUR" USING RETCODE,CURNAME
  IF RETCODE IS NOT EQUAL TO ZERO THEN
  MOVE "IFCCUR  " TO M204-CALL-ERROR
  GO TO ERROR-RTN.
*
GO TO END-RTN.

*
*
* SUBROUTINE TO PROCESS FORD RECORDS
*
UPDATE-AND-WRITE.
  MOVE "IFFTCH  " TO M204-CALL-ERROR.
  CALL "IFFTCH" USING RETCODE,WORK-REC,DIR,CURNAME,DATA-SPEC.
  PERFORM UPDATE-COLOR.
  PERFORM WRITE-REC.
  MOVE SPACES TO WORK-REC.
  SUBTRACT 1 FROM FIND-TOT.
*
* SUBROUTINE TO CHANGE COLOR TO TAN AND UPDATE FILE
*
UPDATE-COLOR.
  MOVE WK-COLOR TO COLOR.
  MOVE "IFUPDT  " TO M204-CALL-ERROR.
  CALL "IFUPDT" USING RETCODE,WORK-REC,CURNAME,DATA-SPEC.
  IF RETCODE IS EQUAL TO 3 THEN GO TO ERROR-RTN.
  MOVE "IFCMMT  " TO M204-CALL-ERROR.
  CALL "IFCMMT" USING RETCODE.
  ADD 1 TO UPDATE-TOT.
*
* SUBROUTINE TO WRITE A REPORT RECORD
*
WRITE-REC.
  MOVE MAKE TO CAR-MAKE-OUT.
  MOVE MODEL TO CAR-MODEL-OUT.
  MOVE YEAR TO CAR-YEAR-OUT.
```

```
  MOVE COLOR TO CAR-COLOR-OUT.
  WRITE CAR-REC-OUT.

*
* SUBROUTINE TO DISPLAY HLI CALL ERRORS
*
ERROR-RTN.
  IF RETCODE IS EQUAL TO 3 THEN
  DISPLAY "RECORD LOCKING CONFLICT".
  CALL "IFERLC" USING RETCODE, RECNUM, USERNUM, FILENAME
  "RECORD NUMBER = " RECNUM
    "USER NUMBER   = " USERNUM
    "IN FILE         " FILENAME.
  MOVE RETCODE TO M204-RETCODE-ERROR.
  DISPLAY "CRITICAL ERROR - UNSUCCESSFUL HLI FUNCTION CALL: "
     M204-CALL-ERROR ", WITH A RETCODE OF: "
     M204-RETCODE-ERROR.
  CALL "IFGERR" USING RETCODE, M204-MSG-ERROR.
  DISPLAY "MODEL 204 ERROR MESSAGE: " M204-MSG-ERROR.
  GO TO END-RTN.
*
* SUBROUTINE TO END THE TRANSACTION
*
END-RTN.
  CALL "IFCLOSE" USING RETCODE.
  CALL "IFFNSH" USING RETCODE.
  •
  •
  •
  STOP RUN.
```

Controlling record locking conflicts

Releasing records

Use the following guidelines for releasing records:

- Release records as soon as you no longer need them. Use IFRELR to release records, including those on a list. (To free pages in CCATEMP, use IFCLST when you no longer need the records in a list.)
- Place the records you need on a list, release the records, and process from the list in situations where no updates are taking place or where updates are known not to affect the data in question.

Processing update units

Use the following guidelines for processing update units:

- Use IFCMMT at the end of logical updates and keep logical updates short.

- If using HLI from an Online monitor, such as CICS, try to keep update units within the same terminal I/O point.

Changes to the database

Use the following guidelines for applications that make changes to the database:

- In an application that reads and updates, segregate updating functions from read-only functions.
- Perform the following functions during off-peak hours: IFRFLD, IFNFLD, and IFDFLD.
- Defer index updates whenever possible. Because exclusive locks are not held as long, deferring index updates speeds updating. Only a part of the work is being done while the lock is held. For more information about index updates, refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/File_Load_utility).

11

Model 204 Security

Overview

This chapter briefly describes Model 204 security for application programmers who are using the Host Language Interface facility.

For more information

For more information about Model 204 security, refer to the *Rocket Model 204 Security Interfaces Manual*.

See also the Rocket Model 204 documentation wiki:

http://m204wiki.rocketsoftware.com/index.php/Storing_security_information_(CCASTAT)

http://m204wiki.rocketsoftware.com/index.php/Establishing_and_maintaining_security

Using Model 204 security

Model 204 provides a variety of security features that prevent unauthorized use of IDs, files, groups, records, and fields. When a particular security feature is operational, the corresponding access restrictions apply.

Login security

Login security requires you to enter a password when logging in. Only a valid password can gain access to the system. After successfully logging in, you are granted particular privileges.

When login security is in effect, specify login information using the following HLI calls:

- For an IFSTRT thread in IFAM1, use the IFLOG call to provide login information, as necessary, where the user authorization is to be validated by a security interface.
- In IFAM2 and IFAM4, specify the login parameter in the IFSTRT call to supply the user ID and password that permit entry to the system.
- For an IFDIAL thread, supply the login information, as necessary, using the IFWRITE call.

File security

File security requires you to specify a legal password in the IFOPEN call. After you successfully open the file, Model 204 grants you particular file privileges, a user class number, and field security levels.

Group security

Group security requires you to specify a legal password in the IFOPEN for the file group.

Record security

Record security limits access to records by allowing you to retrieve and update only records that you have stored in the file or that users can share.

Field-level security

Field-level security protects fields in a file. Field access levels are assigned when you open a file or group. Specify the security level associated with a field in the IFDFLD call.

Terminal security

Terminal security allows particular login user IDs and particular files and file groups to be accessed only from specific Model 204 threads.

Part III
Job-Related HLI Processing Requirements

This part gives details about the components of HLI jobs. It expands on information about HLI jobs presented in the *Rocket Model 204 Host Language Interface Reference Manual*.

12

Tables

Overview

This chapter describes Model 204 tables, which comprise the user work area, for application programmers who are using the Host Language Interface facility. Use the information in this chapter to avoid or correct table full conditions that occur during program execution.

Refer to the descriptions of table entries for multiple cursor IFSTRT thread calls if you are using multiple cursor functionality in your HLI application for the first time.

For more information

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Large_request_considerations) for more information about user work areas.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of individual HLI calls.

User work area

Model 204 allocates an internal work area for each IFSTRT and IFDIAL thread to store the information necessary to evaluate a call or to run a request.

Each work area is composed of a set of tables. The internal work areas for IFSTRT and IFDIAL threads include the following Model 204 tables:

- FTBL, for file groups
- NTBL, for names of cursors, lists, %variables, compilations
- QTBL, for quadruples, that is, statements in internal form

- STBL, for character strings
- TTBL, for temporary work pages
- VTBL, for compiler variables

In IFAM2 and IFAM4, the CCASERVR data set stores a user's work area, including these tables, when the user is swapped out of memory. Refer to Chapter 6 for information about CCASERVR.

The next two sections give information about specifying the size of these tables and avoiding table full conditions. The sections that follow describe each of the user work tables for HLI processing.

Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Large_request_considerations) for more information about user work areas.

Managing table sizes

Specifying user table size

You can specify the size of a user work area table by setting the corresponding user table parameter in any user parameter line in an IFAM1 or IFAM4 job. In an IFAM2 or IFAM4 job, you can reset a user table parameter by using the IFUTBL call.

The following user table parameters correspond to the user tables:

Parameter	Table	Specifies...	Default value	Maximum
LFTBL	FTBL	Length in bytes	1000 bytes	65,528 bytes
LNTBL	NTBL	Number of entries	50 entries	5,460 entries
LQTBL	QTBL	Number of entries	400 entries	262,143 entries
LSTBL	STBL	Length in bytes	600 bytes	16 megabytes
LTTBL	TTBL	Number of entries	50 entries	8,190 entries
LVTBL	VTBL	Number of entries	50 entries	524,287 entries

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for information about IFAM1, IFAM2, and IFAM4 jobs and for a description of the IFUTBL call.

Note: In IFAM1, IFAM2, and IFAM4, the Model 204 server area must be large enough to accommodate the aggregate work area table size for HLI threads in the job.

If you override the default server area that is allocated by specifying the SERVSIZE User 0 parameter, calculate a value using the appropriate SERVSIZE formula that includes the sum of the user table sizes.

Refer to the Rocket Model 204 documentation wiki for more information about server areas, SERVSIZE, and user work area table parameters:

http://m204wiki.rocketsoftware.com/index.php/Defining_the_runtime_environment_(CCAIN)#Server_areas

and for more information about UTABLE parameters:

http://m204wiki.rocketsoftware.com/index.php/Overview_of_Model_204_parameters

Avoiding table full conditions

Model 204 keeps user work area table entries for precompiled specifications and %variables for use by later calls. On a multiple cursor IFSTRT thread, Model 204 also stores compilations and retains multiple record sets.

On each thread, Model 204 clears table entries for noncompiled HLI calls after processing each call. It is unlikely that noncompiled calls would generate enough table entries to cause table full conditions.

However, you might encounter a table full condition for a noncompiled call when the table is already full of entries for calls issued by the thread. If a table is full, Model 204 returns a completion code of 7 to the HLI program.

To avoid table full conditions, when a compilation is no longer needed, delete it with a call to IFFLUSH. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of the IFFLUSH call.

File group table (FTBL)

Model 204 stores data structures related to file groups, as opposed to single files, in FTBL. There are two types of FTBL entries. Model 204 allocates an entry under each of the following conditions:

- Each time a group is opened.

  This type of entry has a fixed-size portion of 56 bytes, plus 2 bytes per file in the group definition. Model 204 releases this entry when the group is closed.

- For collecting field name codes and properties, each time a new field name is encountered in a HLI call.

  The entry size is variable, consisting of 9 fixed bytes, plus a number of bytes equal to the length of the field name plus 11 bytes for each file in the group. Model 204 does not delete this entry until the group is closed or until IFFNSH is executed.

In addition to the space required by these two types of entries, Model 204 allocates a fixed amount of space in FTBL equal to 2 bytes times the value of the NGROUP runtime parameter.

Names table (NTBL)

Model 204 creates one entry for each cursor, list name, %variable, or compilation name. On a single cursor IFSTRT thread, the first IFFDV call also generates an entry, and Model 204 saves entries until a new file or group is opened or until the thread is deleted.

NTBL entries are 12 bytes long.

Note: You can explicitly delete compilation names and %variables with the IFFLUSH call.

Internal statements/quad table (QTBL)

Quadruples are the internal version of many HLI calls. The Host Language Interface evaluates some calls directly. Others, like IFFIND, IFCOUNT, and list manipulations, build quadruples (quads) so that the SOUL evaluator routines can be used.

QTBL entries range from 4 to 40 bytes in length. The following sections describe QTBL entries.

QTBL requirements for search functions

The following HLI calls, which perform find functions, use space in QTBL:

- Each IFCTO generates 8 bytes of control information. Additionally, each field name takes 16 bytes.
- IFFIND generates 20 bytes of control information.

  For information about additional QTBL bytes used by IFFIND, refer to the Rocket Model 204 wiki documentation description of storage requirements for FIND (http://m204wiki.rocketsoftware.com/index.php/Large_request_considerations).
- IFFAC uses the same amount as IFFIND plus an additional 40 bytes.
- IFFDV generates 20 bytes of control information. Additional bytes are generated depending on the type of field used in IFFDV, as follows:
  - For an FRV field, an additional 76 bytes are generated for single files and 96 bytes for a group.
  - For an ORDERED field, an additional 32 bytes are generated.

    For information about additional QTBL bytes used by IFFDV, refer to the Rocket Model 204 documentation wiki description of QTBL storage requirements for FIND ALL VALUES (http://m204wiki.rocketsoftware.com/index.php/Large_request_considerations).

QTBL requirements for retrieval and update functions

The following HLI calls, which perform get and store functions, use space in QTBL:

- Each IFGET compilation generates 12 bytes of control information. Additionally, each field name takes 16 bytes.

  Additional bytes in QTBL are used by IFFTCH, IFUPDT, IFGET, and IFPUT as necessary based on the following storage requirements:

  – If a field name list is specified, each field name or field name variable in the list takes 8 bytes.

  – If an EDIT specification exists, each EDIT item takes approximately 3 bytes.

  – Iteration factors take 4 bytes each.

  – If a secondary field name list is specified with EDIT, the list uses 4 extra bytes plus the space taken for the fields and extra format items.

- An IFGETV compilation requires 12 bytes. Additionally, if an EDIT specification exists, each EDIT item takes approximately 3 bytes.

- IFSTOR takes the same amount as IFGET, plus an additional 20 bytes.

QTBL requirements for %variables

HLI calls, which assign %variables in specifications, use space in QTBL. %Variable assignment specifications use the same amount of space as IFGET specifications. Model 204 deletes a %variable specification as soon as the assignment is completed.

QTBL requirements for sort functions

The following HLI calls, which perform sort functions, use space in QTBL:

- Each IFSORT compilation uses 72 bytes.

- Each IFSRTV compilation uses 76 bytes.

QTBL requirements for cursor functions

The following HLI calls, which perform cursor functions, use space in QTBL:

- Each IFOCUR compilation uses 20 bytes.

- IFFRN uses 8 bytes.

Character string table (STBL)

Model 204 stores all character strings in STBL. Each stored string is preceded by a field that is 1 byte in length.

STBL entries are as follows:

- IFDVAL and IFFILE use STBL to store the value string from the input parameter.
- Any values specified for IFFIND are stored in STBL.
- The EDIT form of IFPUT or of a %variable assignment uses STBL for each value, reusing the space from previous values.
- Each %variable uses enough space to hold its current value, plus 1 byte. If the length of the value changes, the amount of STBL space changes accordingly.
- When an IFOCUR that specifies an IN ORDER BY [ordered field] clause is executed, Model 204 allocates 256 bytes of STBL space. Model 204 frees the STBL space when the cursor is closed.

  Additionally, if LIKE is specified on the IN ORDER clause, Model 204 allocates storage for the pattern terms. This amount depends on the length and format of the pattern string and cannot exceed 255 bytes.

Temporary work table (TTBL)

IFFIND uses TTBL entries to keep track of temporary storage. The number of TTBL entries required depends on the complexity of the selection criteria. Model 204 deletes TTBL entries as soon as the IFFIND has been executed.

TTBL entries are 4 bytes each.

Compiler variable table (VTBL)

Entries in VTBL are variable, with most ranging from 8 to 20 bytes.

Note: In addition to the requirements described in the following sections, the Host Language Interface uses some additional VTBL entries for temporary work space.

VTBL requirements for search functions

The following HLI calls, which perform search functions, use space in VTBL:

- IFCOUNT allocates one 8-byte entry.
- On a single cursor IFSTRT thread, each IFFIND allocates one basic entry, either 8 bytes for a single file or (8 + 8*_number-of-files_) bytes for a group. On a multiple cursor IFSTRT thread, each compiled IFFIND generates one basic entry for a single file or group.

  In addition to the basic entry, IFFIND allocates additional bytes in VTBL as listed below. Model 204 releases all but the basic entry after evaluating the IFFIND call.

  – IFFIND allocates at least two 20-byte entries for scratch purposes, and

more for complex Boolean criteria. Also, one entry is allocated for each field name = value pair referenced.

  - The length of a field name = value entry is at least 20 bytes and is greater for large files.
  - IFFIND generates one 8-byte entry in VTBL if any direct search condition is specified in the IFFIND, and allocates one 28-byte entry for each such direct search condition.

  For information about additional VTBL bytes used by IFFIND, refer to the Rocket Model 204 wiki documentation description of VTBL storage requirements for FIND (http://m204wiki.rocketsoftware.com/index.php/Large_request_considerations).

- On a single cursor IFSTRT thread, each IFFDV allocates one basic entry, either 20 bytes for a single file or (12 + 8 + 8**number-of-files*) bytes for a group. On a multiple cursor IFSTRT thread, each compiled IFFDV generates one basic entry for a single file or group.

  In addition to the basic entry, IFFDV allocates additional bytes in VTBL as listed below. Model 204 releases all but the basic entry after evaluating the IFFDV call.

  - IFFDV allocates two 20-byte entries for scratch purposes, and another 64 bytes for a group.
  - If an IFFDV call containing the FROM/TO option is used with an FRV field, IFFDV allocates one 44-byte entry. With ORDERED fields, IFFDV allocates two 44-byte entries.

VTBL requirements for retrieval functions

IFGET allocates no VTBL entry unless it specifies the IN ORDER clause, which uses 168 bytes.

VTBL requirements for sort functions

The following HLI calls, which perform sort functions, use space in VTBL:

- Each IFSORT compilation uses 76 bytes in VTBL.
- Each IFSRTV compilation uses 80 bytes in VTBL.
- Each IFFRN uses 32 bytes.
- Each IFSTOR uses 32 bytes.

VTBL requirements for cursor functions

The following HLI calls, which perform search functions, use space in VTBL:

- IFOCUR usage depends on the type of record set that the cursor compiles against and on the ordering criteria specified, as follows:
  - If the record set is unordered and no IN ORDER clause is specified, IFOCUR uses 36 bytes.
  - If the record set is unordered and sorted file ordering is specified, IFOCUR uses 100 bytes.
  - If the record set is unordered and Btree ordering is specified, IFOCUR uses 252 bytes.
  - If the record set is a sorted set, IFOCUR uses 40 bytes.

VTBL requirements for lists and %variables

HLI calls that specify lists and %variables use space in VTBL as follows:

- Each list allocates an entry, either 8 bytes for a single file or (8+8**number-of-files*) bytes for a group.
- Each %variable allocates one 28-byte entry.

13

Model 204 Data Sets in HLI Jobs

Overview

This chapter describes the Model 204 data sets used in an IFAM1 or IFAM4 HLI job for application programmers who are using the Host Language Interface facility.

For more information

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description of IFAM1 and IFAM4 jobs.

Model 204 data sets for IFAM1 and IFAM4 jobs

Required data sets

The following standard Model 204 data sets are required in IFAM1 and IFAM4 job runs. Model 204 requires each of these data sets for a particular function.

Model 204 data set	Function
CCAPRINT	Prints the contents of CCAIN
CCASERVR	Holds user work area for server swapping (IFAM4 only)
CCASNAP	Stores a snap dump used for error diagnostics
CCATEMP	Writes a scratch file used for temporary storage

Note: For IFAM2, these data sets are included in the Model 204 Online job.

Data sets required for a particular Model 204 facility

Except for CCAAUDIT which is optional, Model 204 requires that you specify the following data sets when using the particular facility:

Data set	Function	Required for...
CCAAUDIT	Audit log	Optional for recovery
CCAGRP	Group definitions	Permanent file groups
CCAJRN	Journal log	Recovery
CHKPOINT	Checkpoint log	Recovery
CCASTAT	Pointer to table	Security features

Specifying a Model 204 data set

To specify a particular Model 204 data set, include a job control statement that references the data set in the job setup. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for detailed information about IFAM1 and IFAM4 jobs.

The following sections describe each of the Model 204 data sets used in HLI jobs.

CCAPRINT file

CCAPRINT defines a single sequential output data set that contains a summary of the user parameter lines and output generated by User 0 (from CCAIN in IFAM4).

The CCAPRINT file is required for IFAM1 and IFAM4 jobs.

The following guidelines apply when using CCAPRINT:

- In IFAM1 and IFAM4 under z/OS, specify SYSOUT=A.
- In IFAM1 under VSE, specify SYSLST.
- In IFAM1 under CMS, specify PRINTER .

CCATEMP file

CCATEMP is a scratch file, either temporary or permanent, whose pages are used as work areas during a Model 204 run. The CCATEMP data set is initialized with each Model 204 run. CCATEMP must not be shared with other Model 204 jobs.

The CCATEMP file is required for IFAM1 and IFAM4 jobs.

Multiple uses for CCATEMP

Model 204 uses the scratch file for HLI jobs primarily to hold the results of IFFIND calls and record set lists.

In addition, Model 204 uses CCATEMP in the following ways:

- Copies of sorted records

  Copies of records sorted via IFSORT occupy pages in CCATEMP.

  For a single cursor IFSTRT thread, Model 204 releases the pages in CCATEMP as the records are processed by IFGET or when IFFIND is called. For a multiple cursor IFSTRT thread, Model 204 holds the pages in CCATEMP until the sorted set is released.

- File group references

  File groups also use the scratch file. For IFAM1 and IFAM4 jobs that use file groups, Technical Support recommends that you allocate a scratch file of 20 pages for each thread. (Note that in IFAM1, a single thread is permitted.)

CCATEMP size requirements

When using CCATEMP, the unit type must be compatible with the installation site's page size, as summarized below.

Page size	Allowable CCATEMP units
6184	3330, 3340, 3350, 3375, 3380, 3390, or 2305-2

The size of the scratch file is related to the complexity of the retrievals executed during the run.

Note also that copies of sorted records in CCATEMP are at least as large as the original records in Table B. Coded values are expanded to character strings. You must allocate enough CCATEMP space to hold the records to be sorted.

Note: If you encounter the CCATEMP FULL error message in an IFAM1 or IFAM4 job run, increase the CCATEMP space allocation.

Using secondary CCATEMP data sets

To increase the efficiency of the scratch file for very heavy usage, you can partition CCATEMP into as many as 10 secondary data sets (CCATMP0 through CCATMP9).

You can optionally provide secondary scratch data sets for a Model 204 run using file names CCATMP0 through CCATMP9. Model 204 logically appends all the space in each available data set to CCATEMP.

Spreading the data sets over many channels and balancing the channel load improves performance. Also, partitioning makes it possible to incorporate one or more high-speed storage media for CCATEMP.

Note: If you specify CCATMP data sets in a job run, Model 204 attempts to open the secondary data sets in numerical order and stops at the first missing data set.

CCASERVR file

All runs that use server swapping (the parameter NSERVS is less than NUSERS) require one of more server data sets, which can be either temporary or permanent files.

CCASERVR is required in IFAM4 (and is not valid for use in IFAM1).

The server data sets are used for temporary storage of a user's work area when that user is swapped out of memory. Refer to Chapter 12 for information about the user work area.

CCASNAP file

Model 204 can trap program checks and file integrity problems at appropriate times and print snap dumps of selected portions of storage to the CCASNAP data set.

These dumps are invaluable to Technical Support for locating and correcting errors.

The CCASNAP file is required for IFAM1 and IFAM4 jobs.

The following guidelines apply when using CCASNAP:

- In IFAM1 and IFAM4 under z/OS, either the SYSUDUMP or SYSMDUMP data set might also be required to diagnose certain errors. Both are normally specified as SYSOUT data sets.
- In IFAM1 under VM, a VMDUMP may be required in addition to CCASNAP.
- In IFAM1 under VSE, CCASNAP goes to SYSLST.

CCASTAT file

CCASTAT points to a previously created password table data set.

CCASTAT is required for any IFAM1 or IFAM4 run that uses the Model 204 security features.

CCAGRP file

CCAGRP is a previously created data set that stores definitions of permanent groups.

CCAGRP is required for any IFAM1 or IFAM4 run that accesses permanent file groups.

Note: To access the definitions in the CCAGRP file, set option 2 of the SYSOPT parameter.

CCAJRNL, CCAJLOG, and CCAAUDIT files

The Model 204 journals— CCAJRNL and the optional CCAJLOG—and audit trail (CCAAUDIT) files provide a log of information about a Model 204 run. A single execution of Model 204 can log information in a journal(s), an audit trail, or both.

The CCAJRNL, CCAJLOG, and CCAAUDIT files are optional for IFAM1 and IFAM4 jobs. However, CCAJRNL is required for the recovery facility and functions differently for recovery in IFAM1 and IFAM4.

See Chapter 16 for more information about CCAJRNL, CCAJLOG, and CCAAUDIT used in HLI jobs.

CHKPOINT file

CHKPOINT is a sequential file that contains copies of Model 204 file pages before updates are applied and marker records that record the date and time when no updating activity is occurring on the system.

The CHKPOINT file is optional for IFAM1 and IFAM4 jobs. However, CHKPOINT is required for checkpointing, which is used with the recovery facility.

See Chapter 16 for more information about CHKPOINT used in HLI jobs.

14

IFAM2 CICS Processing

Overview

This chapter describes special requirements of CICS programs for application programmers who are using the Host Language Interface facility.

For more information

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for information about HLI jobs running in IFAM2 under CICS.

CICS program link-editing requirements

The version of IFENTPS that is link-edited with your program must be assembled specifically for either macro- or command-level program code. To do this, set the conditional assembly flag &IFCALLR in the CICFG copy member.

IFENTPS requires that register 13 contain the following address.

For...	Register 13 must contain...
Macro-level module	18-word register save area
Command-level module	DFHEISTG storage area

CICS application program work areas

Before issuing any IFAM2 call, ensure that you can address the following CICS areas:

- Common System Area (CSA)
- Task Control Area (TCA)

- Transaction Work Area (TWA)

Transaction work area (TWA)

The IFAM2 interface requires 88 bytes of CICS TWA area. When preparing an application program, ensure that the 88 bytes of the Transaction Work Area (TWA) are reserved for the CICS/Model 204 interface. The Model 204 TWA area may be displaced within the TWA area if application programs require the TWA area.

The area that Model 204 uses may be displaced within the TWA by setting the &IFTWADP in the CICFG copy member. The &IFTWADP value in the CICFG copy member specifies the number of bytes to displace the Model 204 TWA area. The value must be expressed in multiples of four because the Model 204 TWA area fullword-aligns.

Note: Both IFENTPS and IFPS must be assembled with the same CICFG copy member values. The TWA area is used to pass parameters between the two and, thus, must refer to the same area.

Your installation might include several compatible versions of IFENTPS and IFPS for applications that use different TWA areas. The CICS load module name that IFENTPS is linked to and its name in the CICS load library can be specified in &IFAM2LM within the CICFG copy member. During installation, take care that IFENTPS and IFPS are generated in compatible sets.

For example, consider a version of IFENTPS that has its TWA area displaced by 20 bytes and needs to link to a version of IFPS that also expects its TWA area to be displaced by 20 bytes. If the reference to the TWA area is the same, both a macro-level and a command-level version of IFENTPS can refer to the same copy of IFPS.

COBOL example of addressing the CICS areas

The following excerpt from a COBOL program illustrates how you can address the CICS areas (CSA, TCA, and TWA) for the interface from command-level CICS. You can also use this interface with macro-level CICS.

```
IDENTIFICATION DIVISION.
•
•
•
DATA DIVISION.
WORKING-STORAGE SECTION.
•
•
•
LINKAGE-SECTION.
01  DYNAMIC-STORAGE-POINTERS.
    05  FILLER                  PIC S9(8)  COMP.
```

```
    05  CSA-POINTER             PIC S9(8)  COMP.
    05  TCA-POINTER             PIC S9(8)  COMP.
    05  TWA-POINTER             PIC S9(8)  COMP.
        COPY  DFHCSADS.
        COPY  DFHTCADS.
01  TWA-AREA.
    05  M204-INTERFACE-PORTION  PIC X(88).
    05  YOUR-TWA-PORTION        PIC X(?).
PROCEDURE DIVISION.
    EXEC CICS ADDRESS CSA(CSA-POINTER) END-EXEC.
    SERVICE RELOAD DFHCSADS.
    MOVE CSACDTA TO TCA-POINTER.
    SERVICE RELOAD DFHTCADS.
    EXEC CICS ADDRESS TWA(TWA-POINTER) END-EXEC.
     CALL 'IFCSA' USING DFHCSADS.
```

COBOL2 example of addressing the CICS areas

The following excerpt from a COBOL2 program illustrates how you can address the CICS areas (CSA and TWA) for the interface from command-level CICS.

```
IDENTIFICATION DIVISION.
•
•
•
DATA DIVISION.
WORKING-STORAGE SECTION.
*
*
*
01WK-ADR.
03CSA-POINTERPOINTER.
03TWA-POINTERPOINTER.
*
*
*
LINKAGE-SECTION.
COPY DFHCSADS.
*
PROCEDURE DIVISION.
*
*
 EXEC CICS ADDRESS CSA(WKPTR1) END-EXEC.
 EXEC CICS ADDRESS TWA(WKPTR2) END-EXEC.
SET ADDRESS OF DFHCSADS TO CSA-POINTER.
SET ADDRESS OF TWA-AREA TO TWA-POINTER.
 CALL 'IFCSA' USING DFHCSADS.
```

Temporary storage queue

A temporary storage queue is created for each Host Language Interface transaction when the IFSTRT or IFDIAL call is issued. No other user program can use this queue. The name of the queue is:

CCA*Ixxxx*

where:

- *xxxx* is equal to EIBTRMID

CICS abend handling

The Model 204 IFAM2 interface provides automatic abend handling that operates along with your application program abend handler, if any.

Whenever an abend occurs, open CRAM channels must be cleaned up and CICS resources must be released. If the IFAM2 abend handler is not in effect, the user application abend handler must perform these tasks. If the IFAM2 abend handler is in effect, the IFAM2 abend handler supersedes any user abend handler and performs these tasks automatically.

This section provides an overview of IFAM2 abend handling, followed by examples of suggested abend handling.

How to deactivate IFAM2 abend handling

To deactivate the automatic IFAM2 abend handling, set the &IFABEND parameter in the CICFG copy member to NO.

Protecting against abend exposure

Because CICS cancels abend handlers for pseudo conversational waits, an exposure gap in abend protection occurs after each pseudo conversational wait. During the time an application restarts after a pseudo conversational wait and before its next IFAM2 call, the transaction can abend without invoking the IFAM2 abend handler.

To prevent such an exposure, the application program must establish an abend handler immediately upon entering the program. This abend handler must call IFFNSH to close all CRAM channels and release all CICS resources, and must contain any application-specific logic. The program examples in the next section show coding that protects against this abend exposure.

If the application does not require its own abend handler, it can use the IFAM2 call, IFABXIT, which establishes the IFAM2 abend handler. IFABXIT is specific to the CICS environment and is for the convenience of the programmer.

Call IFABXIT as soon as possible after entering the program but only after the initial IFCSA call, which still must be the first call of any IFAM2 application

session. The IFABXIT call protects the application against hung CRAM channels in the event of an abend prior to the first functional IFAM2 call.

How IFAM2 abend handling operates

The Model 204 IFAM2 abend handler is set by the first IFAM2 CICS call (other than IFCSA) and remains active until an IFHNGUP or IFFNSH call occurs, the transaction ends, or a pseudo conversational wait is entered. After a pseudo conversational wait, the abend handler is reset on the next IFAM2 call.

When it is invoked, the IFAM2 abend handler supersedes any user abend handler, operating differently in the macro- and command-level environments:

- Macro-level coded applications cannot use the PUSH and POP commands. Applications that need their abend handler to be in effect after an IFAM2 call must reset their abend handler after returning from each IFAM2 call.
- For command-level applications, the IFAM2 interface issues a PUSH HANDLE command to save the existing application's abend handler before establishing the IFAM2 abend handler.

  In the event of an abend, the IFAM2 interface releases all CRAM channels and CICS resources. It issues a POP HANDLE command to restore the application's abend handler and then issues an abend call. The abend call drives the application's abend handler.

CICS abend handling: macro-level program

Table 14-1 shows recommended IFAM2 abend handling for a macro-level program.

This example assumes that the application's abend handler calls IFFNSH to clean up open IFAM2 CRAM channels and CICS resources.

Table 14-1. CICS Macro-level program

Step	Program flow	Abend handler in effect
1.	Transaction start	No abend handler active
2.	IFCSA call	No abend handler active
3.	Next IFAM2 call	IFAM2 abend handler active
4.	Pseudo conversational wait	IFAM2 abend handler canceled
5.	Transaction restart	No abend handler active, exposure from CRAM channel recovery
6.	DFHPC TYPE=ABEND (or IFABXIT)	User abend handler active (IFAM2 abend handler active)
7.	IFAM2 call	IFAM2 abend handler active

Table 14-1. CICS Macro-level program

Step	Program flow	Abend handler in effect
8.	DFHPC TYPE=ABEND	User abend handler reset

The following points describe each step in the program flow:

1. The transaction program starts. Because no Model 204 CRAM channels are open and CICS resources are not allocated, there is no exposure.
2. An IFCSA call is issued. Prior to Model 204 Version 3.1, IFCSA had to be the first call of any IFAM2 application session. For Model 204 Version 3.1 and later, the IFCSA call is not required. For upward compatibility, the IFCSA call is allowed, but performs no function.
3. The first IFAM2 call after IFCSA is issued. During this call, the IFAM2 interface, IFENTPS, establishes the Model 204 abend handler. This remains in effect until the program ends or enters a pseudo conversational wait.
4. The transaction enters a pseudo conversational wait. CICS cancels the abend handler.
5. The transaction program resumes. No abend handler is active until the application program issues the first IFAM2 call. If an abend occurs, the CRAM channels are not closed, resulting in hung CRAM channels. Any CICS resources, such as storage and queue entries, are not released.
6. The transaction establishes its own abend handler as the first CICS call upon entry. Because the user abend handler is coded with an IFFNSH call, an abend within the user program calls the IFAM2 interface to close existing CRAM channels and release CICS resources.

   Note: An alternative is to call IFABXIT, establishing the IFAM2 abend handler before any functional IFAM2 call.
7. An IFAM2 call is issued. The IFAM2 abend handler goes into effect.
8. The transaction reestablishes its own abend handler. Macro-level applications requiring their own abend handlers must reset their own abend handlers after each IFAM2 call.

CICS abend handling: command-level program

Table 14-2 shows recommended IFAM2 abend handling for a command-level program.

This example assumes that the application's abend handler calls IFFNSH to clean up open IFAM2 CRAM channels and CICS resources.

Table 14-2. CICS command-level program

Step	Program flow	Abend handler in effect
1.	Transaction start	No abend handler active
2.	IFCSA call	No abend handler active
3.	Next IFAM2 call	IFAM2 abend handler active
4.	Pseudo conversational wait	IFAM2 abend handler canceled
5.	Transaction restart	No abend handler active. Exposure from CRAM channel recovery.
6.	HANDLE ABEND (OR IFABXIT)	User abend handler active (IFAM2 abend handler active)
7.	IFAM2 call	IFAM2 abend handler active

The following points describe each step in the program flow:

1. The transaction program starts.
2. An IFCSA call is issued. Prior to Model 204 Version 3.1, IFCSA had to be the first call of any IFAM2 application session. For Model 204 Version 3.1 and later, the IFCSA call is not required. For upward compatibility, the IFCSA call is allowed, but performs no function.
3. The first IFAM2 call after IFCSA is issued. During this call, the IFAM2 interface, IFENTPS, establishes the Model 204 abend handler. This remains in effect until the program ends or enters a pseudo conversational wait.
4. The transaction enters a pseudo conversational wait. CICS cancels the current abend handler.
5. The transaction restarts.
6. The transaction establishes its abend handler as the first CICS call upon entry. because the user abend handler is coded with an IFFNSH call, an abend within the user program calls the IFAM2 interface to close existing CRAM channels and release CICS resources.

   Note: An alternative is to call IFABXIT, establishing the IFAM2 abend handler before any functional IFAM2 call.
7. An IFAM2 call is issued. The IFAM2 call sets its own abend handler, which remains in effect until altered.

Part IV
HLI Transaction Processing and Recovery

This part describes transaction processing and recovery for HLI users. It gives information that is useful for planning a recovery strategy and necessary for using the Model 204 recovery facilities for HLI jobs.

15

HLI Transactions

Overview

This chapter describes Model 204 transaction processing for the application programmer who is using the Host Language Interface facility. It details the action of HLI calls that perform update operations against the database and gives information about backing out an incomplete transaction for a TBO file.

Read the sections that pertain to multiple cursor IFSTRT threads if you are using multiple cursor functionality in your host language program for the first time.

For more information

Refer to Chapter 1 for more information about using threads in HLI applications. Refer to Chapter 9 for information about using HLI calls, IFCMMT and IFCMTR, to end transactions.

Refer to Chapter 16 for information about recovery for HLI jobs. Refer to the Rocket Model 204 documentation wiki for more information about recovery and checkpointing:
http://m204wiki.rocketsoftware.com/index.php/File_integrity_and_recovery

http://m204wiki.rocketsoftware.com/index.php/Checkpoints:_Storing_before-images_of_changed_pages

Transaction processing

In general terms, a transaction is a sequence of operations against the database to access data. The order of the operations is defined by the user in the application program. In data processing terms, there are two kinds of transactions, based on the type of data access:

- Read-only

- Updating

An updating transaction has particular significance when processing against the Model 204 database.

Update unit

An updating transaction corresponds to an HLI call or series of calls that perform(s) an update operation, that is, an operation that modifies the database. A Model 204 updating transaction is called an update unit. There are two types of update units:

- Backoutable (that is, update units can be backed out), which includes:
  - Data manipulation updates to transaction back out (TBO) files.
- Nonbackoutable (that is, update units cannot be backed out), including:
  - Data manipulation updates to non-TBO files
  - Procedure updates
  - Data definition updates, for example, using IFDFLD (DEFINE FIELD)

Transaction is an update unit

In the Model 204 environment, the following terminology applies:

- The term *transaction* refers to a backoutable (TBO file) update unit.
- An *updating call* is an HLI call that performs an update operation on a TBO file.

See “Transaction back out facility” on page 171 for information about backing out transactions for TBO files.

Update unit boundaries

Update units for a multiple cursor IFSTRT thread

On a multiple cursor IFSTRT thread, a Model 204 transaction becomes active, that is, a backoutable update unit is started, when the first updating call is executed against the database. An updating call performs an update operation, that is, an operation that modifies the database.

The update unit continues until it is explicitly ended by an IFCMMT, IFCMTR, or IFBOUT call. The IFCMMT call ends the current transaction. A new update unit does not begin until the next updating call is issued. Until the new update unit begins, no transaction is active on the thread.

If the host language program using a multiple cursor IFSTRT thread does not perform any update processing, no update units are allocated and no transaction is activated for the thread.

Model 204 manages update units for a multiple cursor IFSTRT thread similarly to SOUL.

Update units for a single cursor IFSTRT thread

The boundaries of update units for single cursor IFSTRT threads are different from those for multiple cursor IFSTRT threads and SOUL. The thread that starts an update unit must be the thread that ends the update unit. When using IFDTHRD, your must assure in your coding that a change in threads does not occur within the context of a single update unit. Always call IFCMT or IFCMTR before calling IFDTHRD to assure that update unit processing starts and completes on the same thread.

For single cursor IFSTRT threads, an update unit is started by the first IFSTRT call issued by the host language program that starts an update thread (THRD_IND is 0 in IFAM1; THRD_IND is 1 in IFAM2 / IFAM4).

In general, a new update unit starts as soon as the previous update unit ends and a transaction remains active until host language processing is finished. The only HLI functions that end the current update unit and do not immediately start a new update unit are IFCHKPT and IFFNSH.

If an HLI job contains more than one IFSTRT thread with update privileges, all those threads participate in the same transaction. And if the transaction is backed out, all the updates for all the IFSTRT threads with update privileges are backed out.

Note: Read-only IFSTRT threads are not included in update units. Only single cursor IFSTRT threads with update privileges participate in Model 204 transactions.

When an update unit ends

When an update (backoutable or nonbackoutable) ends, Model 204 performs the following actions:

- Writes the current journal buffer out to the journal (CCAJRNL)
- Releases the exclusive lock on updated records in the pending updates pool for any files with the LPU option set
- If the update unit is backoutable, frees the back out log and constraints log for the transaction (which means that the transaction can no longer be backed out)
- Commits the update unit to the database (or databases) that were modified

When a transaction backs out

When a transaction backs out, all the updates performed by that transaction are undone and affected files are returned to the same state as before the transaction began.

After the back out is complete, the transaction ends, and Model 204 performs the same actions as for the end of an update unit. (See above.)

Refer to Chapter 16 for more information about backing out a transaction.

Update units: designing your application

Placing terminal I/O points outside update units

If a response is required from a terminal either to complete or back out a transaction, a set of records that is exclusively locked for the update unit might be unintentionally locked for a long time.

Avoid placing terminal I/O points between the start of a transaction or backoutable update unit and the transaction's end or back out. Place terminal I/O points outside of transactions.

Note: When using a single cursor IFSTRT thread, update units inhibit checkpoints and thus increase the amount of work to be done for recovery. Refer to Chapter 16 for more information about checkpointing in your host language application.

Unit of work for recovery

Update units have particular significance for Model 204 recovery. Recovery uses the starting and ending points of update units to return files to logically and physically consistent states. An update unit is a unit of work for recovery.

To plan for recovery when you are coding your HLI application program, you must know where update units begin and end.

HLI updating calls and update units

HLI calls that end the current update unit

On a multiple cursor IFSTRT thread, each of the following HLI calls ends the current update unit.

HLI call	Equivalent SOUL command
IFBOUT	BACK OUT
IFCMMT	COMMIT
IFCMTR	COMMIT RELEASE

HLI call	Equivalent SOUL command
IFCLOSE	CLOSE
IFDTHRD, if the thread has a file open	
IFFNSH	

Note that, in group file context, IFCLOSE ends the transaction only if it results in the closing of a file and all files opened outside of the group have been closed. IFCLOSE does not end the transaction if all files in the group are opened singly.

On a single cursor IFSTRT thread with update privileges, all the calls listed above (except for IFCMTR) end the current update unit and start a new update unit, and CLOSE ALL is the only option for the IFCLOSE call.

HLI calls that start an undesignated update unit

On a single cursor IFSTRT thread with update privileges, each of the following HLI calls performs different actions depending on whether or not an update unit is active.

HLI call	Equivalent SOUL command
IFDELF	DELETE FIELD
IFDFLD	DEFINE FIELD
IFINIT	INITIALIZE
IFNFLD	RENAME FIELD
IFRFLD	REDEFINE FIELD
IFRPRM	RESET
IFSPRM	SET If used to reset a file parameter

If no update unit is active

If no update unit is active when one of the previous HLI calls is issued, Model 204 performs the following actions:

- Starts a nonbackoutable update unit
- Processes the update
- Ends the nonbackoutable update unit
- Starts a new update unit

If an update unit is active

If an update unit is active when one of the previous HLI calls is issued, Model 204 performs the following actions:

- Ends the active update unit
- Starts a nonbackoutable update unit
- Processes the update
- Ends the nonbackoutable update unit
- Starts a new update unit

HLI calls that start a backoutable update unit

On a single cursor IFSTRT thread with update privileges, each of the following HLI calls performs the following actions:

- Ends a nonbackoutable update unit if one is active
- Starts a backoutable update unit or continues a previously started backoutable update unit

HLI call	Equivalent SOUL command
IFBREC	STORE RECORD
IFDALL	DELETE ALL
IFDREC	DELETE RECORD
IFDVAL	DELETE fieldname=value
IFDSET	DELETE RECORDS IN
IFFILE	FILE RECORDS
IFPUT	

HLI threads and transactions

Model 204 manages transactions for the threads that are started by a host language application. Model 204 manages transactions differently for different types of threads, depending in which environment (IFAM1, IFAM2, IFAM4) the application is running.

Logical relationship of threads and transactions

The logical relationship between HLI threads and Model 204 transactions are shown in simplified form in the diagrams in the following sections.

The diagrams do not show the detailed elements of the interface, which are particular to each environment. Also, the diagrams provide an overview of HLI

transactions and do not represent the preferred HLI job, which is recommended by Technical Support.

The figures use the following symbols:

Symbol	Meaning
MC	Multiple cursor IFSTRT thread
ST	Single cursor IFSTRT thread
DI	IFDIAL thread
T-n	Model 204 transaction

Refer to Chapter 1 for more information about using threads in HLI applications.

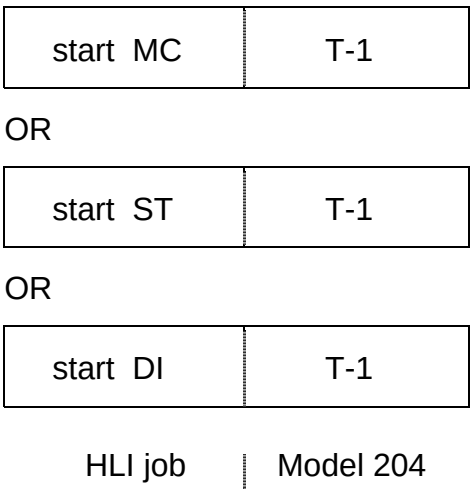
IFAM1 transaction

Single thread

In IFAM1, you can start a single thread in your HLI application program. The updating calls that are issued on the thread are processed in the transaction.

Figure 15-1 shows the logical relationship between an HLI thread in an IFAM1 application and a Model 204 transaction.

Figure 15-1. An IFAM1 Transaction



IFAM2 transactions

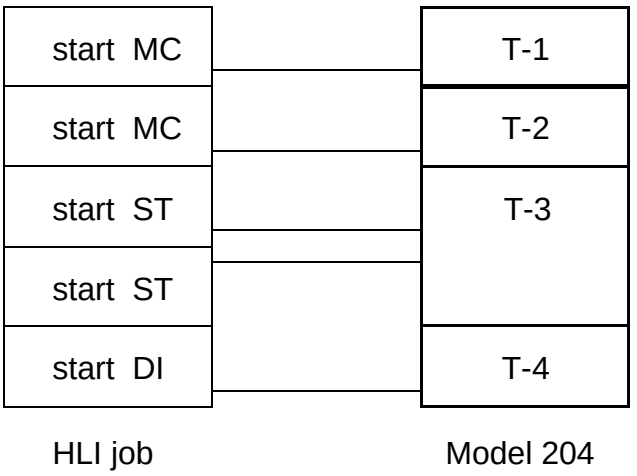
One or more threads

In IFAM2, you can start one or more threads in your HLI application program.

Each multiple cursor IFSTRT thread and IFDIAL thread processes as a single transaction. In a multithreaded application, all single cursor IFSTRT threads process as a single transaction, as shown in Figure 15-2 and Figure 15-3.

Figure 15-2 shows the logical relationship between HLI threads in an IFAM2 application under z/OS or z/VSE and Model 204 transactions.

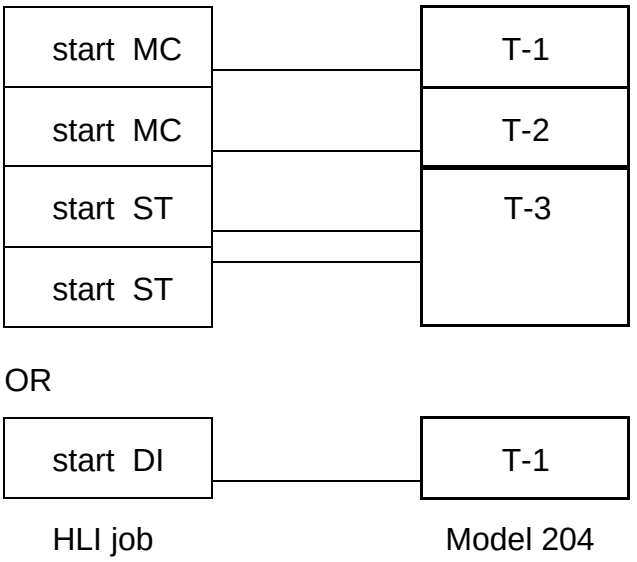
Figure 15-2. IFAM2 Transactions in z/OS or z/VSE



Note: Technical Support does not recommend that you mix multiple cursor and single cursor IFSTRT threads in a HLI application.

Figure 15-3 shows the logical relationship between HLI threads in an IFAM2 application under VM/CMS and Model 204 transactions. Under VM, you can start a single thread in your IFAM2 job if you are using IFDIAL.

Figure 15-3. IFAM2 Transactions in VM



Note: Technical Support does not recommend that you mix multiple cursor and single cursor IFSTRT threads in a HLI application.

IFAM4 transactions

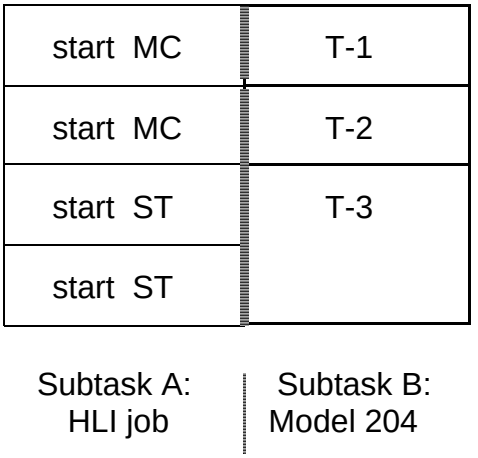
One or more threads

In IFAM4, you can start one or more IFSTRT threads in your HLI application program.

Each multiple cursor IFSTRT thread processes as a single transaction. In a multithreaded application, all single cursor IFSTRT threads process together as a single transaction.

Figure 15-4 shows the logical relationship between HLI threads in an IFAM4 application and Model 204 transactions.

Figure 15-4. IFAM4 Transactions



Note: Technical Support does not recommend that you mix multiple cursor and single cursor IFSTRT threads in a HLI application.

Multithreaded IFAM2 and IFAM4 transactions

In IFAM2 and IFAM4, a host language job can start more than one thread. When you start more than one single cursor IFSTRT thread in your HLI application, Model 204 treats all of those threads as a single transaction.

For example, an IFBOUT executed on one of the single cursor IFSTRT threads backs out the updates for all of the single cursor IFSTRT threads in the application program. Likewise, an IFCMMT executed on one of the single cursor IFSTRT threads commits the updates for all of the single cursor IFSTRT threads in the program.

Multiple single cursor IFSTRT threads function as a separate transaction from an IFDIAL thread or any multiple cursor IFSTRT threads that are started by the application.

Note: Technical Support does not recommend that you mix multiple cursor and single cursor IFSTRT threads in a HLI application. An HLI job that mixes single cursor and multiple cursor IFSTRT threads processes multiple transactions separately against the database. This might produce undesired results because a job normally performs a logical unit of work.

Refer to Chapter 1 for more information about multithreaded host language applications.

Multithreaded transaction with read-only IFSTRT threads

Single cursor IFSTRT threads with read-only privileges are not included in update units. This can cause undesired results in a multithreaded application. Such threads can access data from uncommitted nonbackoutable update units, which can lead to logical inconsistencies if updates are made based on this data.

If your HLI job uses multiple single cursor IFSTRT threads and performs any updating operations, start all single cursor IFSTRT threads with update privileges so that all of the threads participate in transaction processing.

To use a single cursor IFSTRT thread with updating privileges for retrieval operations, open files on the thread with read-only passwords. This prevents problems that result from accessing uncommitted updates when you use single cursor IFSTRT threads with read-only privileges.

Using IFCMMT in a multithreaded transaction

A call that ends a multithreaded transaction (IFCMMT, IFCMTR, IFBOUT) releases only the record and resource locks associated with the thread on which it is issued.

Whenever one of these calls is issued in a multithreaded application, IFCMMT must be issued on all the remaining threads that participate in the transaction. This ensures that all associated record and resource locks are released.

Note: If IFCMMT is not issued on all remaining threads in the multithreaded transaction, it is possible for the HLI application to generate locking conflicts with itself.

Committing transactions for lock pending updates files

Lock Pending Updates (LPU) locking mechanism

A TBO file, or any file with the LPU option enabled, exclusively locks all records updated within a transaction. When the transaction completes (when the update unit is committed) or is backed out, the exclusive lock on the entire set of updated records in the pending update pool is released.

The locking mechanism prevents updated records in one update unit from being used by other applications until Model 204 establishes whether the update is committed. An update can be backed out without affecting the logical validity of any other update.

Minimizing enqueuing conflicts

A TBO file, or any file with the LPU option enabled, generates an exclusive lock on updated records until the updates are committed. The LPU locking behavior may result in many additional enqueuing conflicts.

You can minimize the occurrence of enqueuing conflicts on those files by frequently issuing a call to IFCMMT. If you issue IFCMMT frequently, that is, after each updating call, each record update is immediately committed and each LPU lock is released.

For example, the following COBOL excerpt shows a frequent call to IFCMMT:

```
GET-NEXT RECORD.
CALL "IFGET" USING...
IF RETCODE IS EQUAL TO 2
GO TO NO-MORE-RECORDS.
*
* PERFORM RECORD PROCESSING ROUTINES
*
•
•
•
CALL "IFPUT" USING...
CALL "IFCMMT" USING...
GO TO GET-NEXT-RECORD.
*
NO-MORE-RECORDS.
```

In this example, the LPU exclusive lock is obtained on each record as it is updated with IFPUT, then it is released with a call to IFCMMT.

Alternative for minimizing enqueuing conflicts

When you issue frequent calls to IFCMMT, as described in the preceding section, although record locking conflicts are reduced, a particular application might actually take longer to run because of the overhead associated with commit processing.

An alternative to immediate updating is to issue IFCMMT less frequently, so that LPU locks on updated records in the pending update pool are held until the entire found set is processed.

For example, the COBOL excerpt below shows an infrequent call to IFCMMT:

```
GET-NEXT RECORD.
CALL "IFGET" USING...
IF RETCODE IS EQUAL TO 2
GO TO NO-MORE-RECORDS.
*
* PERFORM RECORD PROCESSING ROUTINES
*
•
•
•

CALL "IFPUT" USING...
```

```
GO TO GET-NEXT-RECORD.
*
NO-MORE-RECORDS.
CALL "IFCMMT" USING...
```

In this example, the LPU exclusive lock is obtained on each record as it is updated with IFPUT. The LPU locks are held on the updated records in the pending update pool until the entire found set is processed, then they are released with a call to IFCMMT.

Note that IFDREC, IFDSET, and IFFILE might result in logical data inconsistencies. See page 175 for more information about logical inconsistencies.

Refer to Chapter 9 for more information about using HLI calls to end transactions.

Transaction back out facility

The Model 204 transaction back out facility provides a mechanism to undo the effects of incomplete updates to TBO files. A back out is a logical inversion of an update, which restores a file to its original state before the current update unit started.

The back out mechanism logically inverts the effects of an update to a TBO file by issuing compensating updates. The back out of a transaction can be initiated only on active transactions. Completed transactions cannot be backed out.

Invoking the back out mechanism ends the current transaction. A back out operation itself cannot be backed out. See “Using the transaction back out facility” on page 172 for more information about backing out a transaction.

See page 162 for a description of update units. For complete information about the transaction back out facility, refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/Transaction_back_out).

Requirements for a back out

Model 204 requires that the following conditions be met for a back out to be successful:

- All files in the transaction must be TBO files, so that transaction back out logging is active.

  A back out log stores compensating updates and a constraints log prevents reuse of freed file resources until the transaction ends. See page 173 for a description of these logs.

- The updated records must be locked exclusively so that no other update units are affected.

Note: TBO is the default file type. Lock pending updates is an option of the FOPT parameter that is enabled or disabled on a file by file basis.

See page 174 for information about backing out LPU files. Refer to the Rocket Model 204 documentation wiki (http://m204wiki.rocketsoftware.com/index.php/FOPT_parameter and http://m204wiki.rocketsoftware.com/index.php/File_integrity_and_recovery) for more information about the FOPT parameter.

Using the transaction back out facility

Model 204 automatically backs out an incomplete update for a TBO file under any of the following conditions:

- Model 204 cancels a HLI call or SOUL request
- Model 204 detects a file problem, such as a table full condition
- Model 204 restarts a user who has a transaction in progress (hard restart)

Upon canceling a HLI call in an active transaction, Model 204 returns a completion code (RETCODE) of 10. Model 204 automatically backs out the active transaction and returns control to the HLI application program.

You can use the IFGERR call to retrieve the text of the latest cancellation and counting error messages generated by Model 204. Note that IFGERR returns a null value if no such message exists.

Using IFBOUT to back out updates

A host language application can initiate a back out to undo an incomplete update to a TBO file. To activate the back out mechanism in your HLI program, issue the IFBOUT call. Note that IFBOUT is valid for use only with TBO files.

When the IFBOUT call is successfully executed, Model 204 performs the following actions:

- Backs out the current transaction
- Ends the transaction
- Releases the LPU exclusive lock on updated records in the pending updates pool

Note: Upon executing IFBOUT, Model 204 does not release found sets. A call to IFBOUT does not change the current record.

Using transaction back out logs

Back out log

Model 204 builds a log of compensating updates, called a back out log, for each active transaction. When a transaction ends or is backed out, Model 204 discards the log for that particular transaction.

The back out log contains the necessary information to do a transaction back out on each active transaction in the Model 204 run. For example, if an IFDREC (DELETE RECORD) call is processed, all the information necessary to rebuild the entire deleted record must be available to Model 204 in case the transaction containing the IFDREC operation is backed out.

Model 204 stores the back out log in CCATEMP. At least one CCATEMP page is always used for transaction back out logging.

Constraints log

Model 204 builds a log of freed resources, called a constraints log, for each active transaction. When a transaction ends or is backed out, Model 204 discards the constraints log entries for that particular transaction.

Model 204 keeps the constraints log to guarantee the availability of resources freed by an active transaction. For example, the IFDREC, IFDVAL, IFDALL, and IFPUT calls free table space and other file resources. It is important that such freed file resources are not reused by other active transactions until the transaction that freed the resources ends.

Model 204 stores the constraint information in CCATEMP, in specially formatted pages.

Issue frequent calls to IFCMMT

To help keep both the back out log and the constraints log a manageable size, make transactions as brief as possible and issue IFCMMT as often as is feasible.

Refer to Chapter 9 for information about using IFCMMT.

Back out logging and CCATEMP space

Model 204 stores the back out and constraints logs in CCATEMP.

Transactions that are allowed to continue over many file updates increase the size of back out and constraint logs and can degrade the performance slightly for users who are doing back out logging.

Large back out and constraint logs also increase the possibility of CCATEMP filling. If transaction logging or constraint records fill up CCATEMP, Model 204

backs out the active transaction and cancels the user request, but does not restart the user.

Lessening CCATEMP space requirements

You can lessen the CCATEMP space required for logging and constraint records if you keep transactions short in your host language application, so that they contain a few file updates, by issuing frequent calls to IFCMMT.

Note: Theoretically, the only upper limit on the number of pages that can be used is the CCATEMP size. However, other uses of CCATEMP space limit the back out and constraint log pages that are available to the transaction back out facility. You might need a larger CCATEMP if you use TBO files.

Refer to the Rocket Model 204 documentation wiki for more information about the size calculation of CCATEMP:

http://m204wiki.rocketsoftware.com/index.php/Using_the_system_scratch_file_(CCATEMP)

Transaction back out for LPU files

HLI calls that do not lock records for LPU files

Three updating calls function differently from normal LPU file processing. When processing against files that have the LPU option enabled, the following HLI calls do not lock records:

HLI call	Equivalent SOUL command
IFDREC	DELETE RECORD
IFDSET	DELETE SET
IFFILE	FILE

The records that are updated with any of these calls are not added to the pending update pool and are not locked for the remainder of the update unit. Deleted records are not locked because they no longer exist in the file, and thus cannot be accessed by other update units.

Note: These calls are the only exceptions to the exclusive locking behavior of LPU files on updated records.

Logical inconsistencies can occur when a transaction involving IFDREC, IFDSET, and IFFILE is backed out.

Logical inconsistencies with deleted records

When you are using TBO files with the LPU option enabled, logical inconsistencies can occur when a transaction involving IFDREC or IFDSET to delete records is backed out.

To prevent logical inconsistencies when deleting records, issue frequent calls to IFCMMT.

The following example shows why there is a need to keep transactions brief when deleting records, using IFCMMT to reduce the likelihood of logical inconsistencies.

In this example, events occur in the following order:

1. User 1 begins processing, deletes (IFDSET) the set of records with NAME = SMITH, and continues to update other records.

   Note: The deleted set of records is not locked (IFDSET does not lock records), and User 1's transaction is not complete.

2. User 2 begins processing and finds the set of records with either NAME = SMITH or NAME = SAUNDERS.

3. There is no enqueuing conflict because the deleted records do not exist.

4. User 2 adds a field to his found set (which includes only SAUNDERS records), prints a report, and ends his transaction.

5. Subsequently, User 1's application comes to a point where it calls for the transaction in progress to be backed out. Because the deletion of the SMITH records is in the backed out transaction, the SMITH records reappear on the file.

   However, the original SMITH records do not have the field added by User 2. There is a logical inconsistency in the file.

If, in the first step, User 1 had issued an IFCMMT immediately after deleting the SMITH records, no inconsistency would have occurred unless the back out mechanism had been automatically activated during that short piece of the user request.

Logical inconsistencies using IFFILE

When you are using TBO files with the LPU option enabled, logical inconsistencies can occur when a transaction involving IFFILE to update records is backed out.

To prevent logical inconsistencies when updating records with IFFILE, use IFFNDX to lock records exclusively and hold the lock until after the update unit that contains the IFFILE call ends.

16

Recovery and Checkpoints

Overview

This chapter describes Model 204 recovery and checkpoints, which are used for recovery, for application programmers who are using the Host Language Interface facility.

Refer to the descriptions of particular HLI calls and their use in transaction processing, and code your HLI application to minimize the amount of work required for recovery.

Read the information about checkpoints on a multiple cursor IFSTRT thread if you are using multiple cursor functionality in your HLI application for the first time.

For more information

Follow the guidelines in this chapter for using the Model 204 recovery facilities for HLI jobs. Note that you might need to ask your Model 204 system administrator for additional assistance when running recovery for HLI jobs.

Refer to Chapter 15 for information about managing HLI transactions.

Refer to the Rocket Model 204 documentation wiki for more information about recovery and checkpointing:
http://m204wiki.rocketsoftware.com/index.php/File_integrity_and_recovery

http://m204wiki.rocketsoftware.com/index.php/Checkpoints:_Storing_before-images_of_changed_pages

Model 204 recovery facilities

IFAM1 Roll Back recovery

In IFAM1, only Roll Back recovery is available. You cannot do roll forward or media recovery in IFAM1.

For roll back recovery, you must include the CHKPOINT file in the IFAM1 job. To run the recovery step, run a BATCH204 job with the CHKPOINT file from the IFAM1 job as input and issue the RESTART command.

On a single cursor IFSTRT thread, Model 204 automatically marks the start and end points on the CHKPOINT file. You can specify additional checkpointing to be performed throughout the IFAM1 job run by specifying either or both the CPTIME and CPSORT user zero parameters.

On a multiple cursor IFSTRT thread, you can specify checkpointing to be performed throughout the IFAM1 job by specifying the CPTIME user zero parameter.

Note that you cannot issue the IFCHKPT call in your IFAM1 application.

See “Enabling the checkpoint facility” on page 180 for more information about IFAM1 checkpointing.

Recovery Logging

The Model 204 journals—CCAJRNL and the optional CCAJLOG—and audit trail (CCAAUDIT) files provide a log of information about a Model 204 run. A single execution of Model 204 can log information in a journal(s), in an audit trail, or in both files.

These logs for HLI use are described in more detail in the following sections. For complete information about the journals and audit trail, refer to the Rocket Model 204 documentation wiki:

http://m204wiki.rocketsoftware.com/index.php/Tracking_system_activity_(CCAJRNL,_CCAAUDIT,_CCAJLOG)

Journals

The journals are sequential files that maintains run information and can be used to analyze the functional operation of Model 204. Except for IFAM1, CCAJRNL alone contains the following information, which is provided during execution of a Model 204 job:

- User input
- System messages
- Roll forward entries

If both CCAJRNL and CCAJLOG are defined, CCAJRNL collect the recovery records and CCAJLOG collects the messages and statistics.

In IFAM1, the journal does not contain roll forward information.

Note: The CCAJRNL is required if system RESTART recovery or media recovery is being used. The CCAJRNL is produced in a nonprintable format that is used during system recovery and media recovery.

To use the journal in IFAM1, include a CCAJRNL DD, DDBL, or FILEDEF statement in the job setup. In IFAM4, include a CCAJRNL DD statement.

You can use the Audit204 utility to print the journals, to format billing information, and to analyze some types of system statistics. The Audit204 utility accepts CCAJRNL, if you define only one journal. If you define both journals, the Audit204 utility accepts only CCAJLOG.

Audit trail

The audit trail is a formatted form of the journal. It contains all the information that is kept in the journal except the roll forward information (that is, the user input and system messages).

The audit trail can be printed directly by a Model 204 run without requiring a separate job step. It is invaluable for debugging application programs and for analyzing system performance.

Technical Support recommends that you use an audit trail under the following conditions:

- For HLI batch jobs, to avoid running an extra job step to print the journal
- In jobs that require a printed log

To use the audit trail in IFAM1, include a CCAAUDIT DD, DDBL, or FILEDEF statement in the job setup. In IFAM4, include a CCAAUDIT DD statement.

Checkpoints

The Model 204 checkpoint facility consists of pseudo subtasks, which work together to take checkpoints, and the CHKPOINT file.

The CHKPOINT is a sequential file that contains copies of file pages before updates are applied (called *before-images* or *preimages*) and marker records (called *checkpoints*) that record the date and time when the system is *quiescent*, that is, when no updating activity is occurring.

Each checkpoint taken during a Model 204 run marks a time when no updates are in progress. When a checkpoint is taken, Model 204 writes a record containing a date and time stamp to CHKPOINT.

Using the checkpoint facility in conjunction with the recovery facilities, a valid copy of the Model 204 database can be recovered after a system failure. See

"Model 204 recovery facilities" on page 178 for a description of roll back, roll forward, and media recovery.

Refer to the Rocket Model 204 documentation wiki for more information about recovery and checkpointing:
http://m204wiki.rocketsoftware.com/index.php/File_integrity_and_recovery

http://m204wiki.rocketsoftware.com/index.php/Checkpoints:_Storing_before-images_of_changed_pages

Enabling the checkpoint facility

In IFAM1 and IFAM4, to enable checkpointing, include the following in the HLI job setup:

- A job control CHKPOINT DD, DLBL, or FILEDEF statement (which defines the CHKPOINT file)
- The RCVOPT (recovery options) system parameter, set to 1 or to 9 if roll forward logging is also used, either on the User 0 parameter line in the CCAIN file or in the JCL EXEC parameter (which indicates that checkpoints are to be taken)

In IFAM2, the CHKPOINT file and RCVOPT parameter are specified in the Model 204 online run. Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for information about HLI jobs.

Refer to the Rocket Model 204 documentation wiki for detailed information about the CHKPOINT file and RCVOPT parameter:

http://m204wiki.rocketsoftware.com/index.php/Checkpoints:_Storing_before-images_of_changed_pages

Four different checkpointing mechanisms

Model 204 provides the HLI user with four different mechanisms by which checkpointing can be activated, either automatically or explicitly at the user's request.

When used, the following User 0 parameters cause Model 204 to automatically initiate an attempt to take a checkpoint:

- CPTIME, which performs a checkpoint attempt at timed intervals
- CPSORT, for which a checkpoint attempt is initiated by HLI calls to IFSTRT and IFFNSH on a single cursor IFSTRT thread

The following functions provide the Model 204 user with a means to explicitly initiate checkpointing:

- HLI call to IFCHKPT in IFAM2 or IFAM4
- CHECKPOINT command, which can be issued only on an IFDIAL thread

Checkpointing specific to HLI processing for CPTIME, CPSORT, and IFCHKPT is described in more detail on the following pages. Refer to the *Rocket Model 204 Command Reference Manual* for information about the CHECKPOINT command.

Automatic checkpointing: CPTIME

When timed checkpointing is activated, Model 204 attempts to take a checkpoint at regular timed intervals. This process is controlled by the following User 0 parameters:

- CPTIME, which is the number of minutes between attempts to take a checkpoint
- CPTQ, which is the number of seconds to wait for IFSTRT threads to quiesce before timing out a checkpoint
- CPTO, which is the number of seconds to wait for IFDIAL threads, or SOUL threads in IFAM2, to quiesce before timing out a checkpoint

When the CPTIME interval expires

When the CPTIME interval expires, Model 204 disallows new update units. Model 204 performs other actions depending on the status of update units and the CPTQ and CPTO checkpoint parameters.

Model 204 performs the following actions:

- If the system is quiescent, that is, no updates are currently in progress, Model 204 first takes a checkpoint and then allows new updates to begin.
- If any IFSTRT threads have updates in progress, Model 204 starts the CPTQ timer.

  Whenever an update unit ends, Model 204 checks for any other IFSTRT updates in progress and repeats the process until no more IFSTRT threads are updating or until CPTQ expires.
- If any IFDIAL threads, or SOUL threads in the IFAM2 environment, have update units in progress, and if the CPTQ interval has not expired, Model 204 starts the CPTO timer.

  Whenever an update unit ends, Model 204 checks for any other IFDIAL updates (or SOUL thread updates in the IFAM2 environment) in progress and repeats the process until no more IFDIAL threads, or SOUL threads in the IFAM2 environment, are updating or until CPTO expires.

Specifying a CPTIME value

In IFAM2, CPTIME is set in the Model 204 Online run. In IFAM1 and IFAM4, you can specify the CPTIME User 0 parameter in the CCAIN input file. If you do not

specify a value, CPTIME defaults to a value of 0, and timed checkpoints are disallowed for the HLI/Model 204 run.

A user having system manager privileges can reset the CPTIME parameter only if CPTIME is set to a nonzero value on the User 0 parameter line.

CPTIME processing steps

See the following processing flow charts for a detailed description of the steps involved in CPTIME checkpointing:

- “Checkpoint processing steps: CPTIME main flow” on page 185
- “Checkpoint processing steps: CPTQ timer” on page 186
- “Checkpoint processing steps: CPTO timer” on page 187
- “Checkpoint processing steps: CPTIME time-out” on page 188

Refer to the Rocket Model 204 documentation wiki for more information about the CPTIME parameter:

http://m204wiki.rocketsoftware.com/index.php/Checkpoints:_Storing_before-images_of_changed_pages

Automatic checkpointing: CPSORT

CPSORT checkpointing attempts to take a checkpoint upon the execution of the initial IFSTRT call (only for a single cursor IFSTRT thread) in an HLI job and upon execution of an IFFNSH call on a single cursor IFSTRT thread for every IFAM2 job (which uses single cursor IFSTRT threads).

Note: CPSORT can be used on a single cursor IFSTRT thread, and is not available for use on a multiple cursor IFSTRT thread.

CPSORT checkpointing is controlled by the following User 0 parameters:

Parameter	Description
CPSORT (**C**heckpoint **S**ign-**O**n **R**etry)	Number of times that Model 204 attempts to take a checkpoint at the beginning (the initial IFSTRT call) and end (an IFFNSH call) of HLI jobs.
CPTQ	Number of seconds to wait for IFSTRT threads to quiesce before timing out a checkpoint
CPTO	Number of seconds to wait for IFDIAL threads, or SOUL threads in IFAM2, to quiesce before timing out a checkpoint

CPSORT operates separately from CPTIME and has no effect on the timing of CPTIME checkpoints.

Specifying a CPSORT value

In IFAM2, CPSORT is set in the Model 204 Online run. The CPSORT parameter can be reset by a user having system manager privileges.

Note: CPSORT defaults to a value of 1. You can specify a higher value. However, a high CPSORT value can inhibit new update units for long periods of time and affect overall system throughput.

If you specify a value of 0 for CPSORT, Model 204 does not attempt to take a checkpoint at the beginning and end of HLI jobs.

CPSORT is useful for roll back recovery in the IFAM2 multiuser environment, because it marks the beginning and end of each HLI job.

CPSORT processing steps

See the following processing flow charts for a detailed description of the steps involved in CPSORT checkpointing:

- “Checkpoint processing steps: CPSORT main flow” on page 189
- “Checkpoint processing steps: CPTQ timer” on page 186
- “Checkpoint processing steps: CPTO timer” on page 187
- “Checkpoint processing steps: CPSORT main flow” on page 189

Refer to the Rocket Model 204 documentation wiki for more information about the CPSORT parameter:

http://m204wiki.rocketsoftware.com/index.php/Checkpoints:_Storing_before-images_of_changed_pages

IFCHKPT checkpointing

The IFCHKPT call provides a mechanism for initiating attempts to take checkpoints from within an HLI application.

You can use IFCHKPT in IFAM2 and IFAM4 applications. You cannot issue a call to IFCHKPT in an IFAM1 application.

Differences in checkpointing procedure

There are differences in the procedure that is used for checkpointing depending on whether IFCHKPT is issued on a multiple cursor IFSTRT thread or on single cursor IFSTRT threads.

For example, in a multithreaded IFSTRT application, each single cursor IFSTRT thread that is updating must indicate to Model 204 that it is quiescing in preparation for an attempt to take a checkpoint. A single cursor IFSTRT thread performing update processing prevents checkpoints from occurring

unless the thread specifically requests a checkpoint by issuing an IFCHKPT call.

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for a description IFCHKPT and detailed information about using the IFCHKPT call on different types of IFSTRT threads.

IFCHKPT processing steps

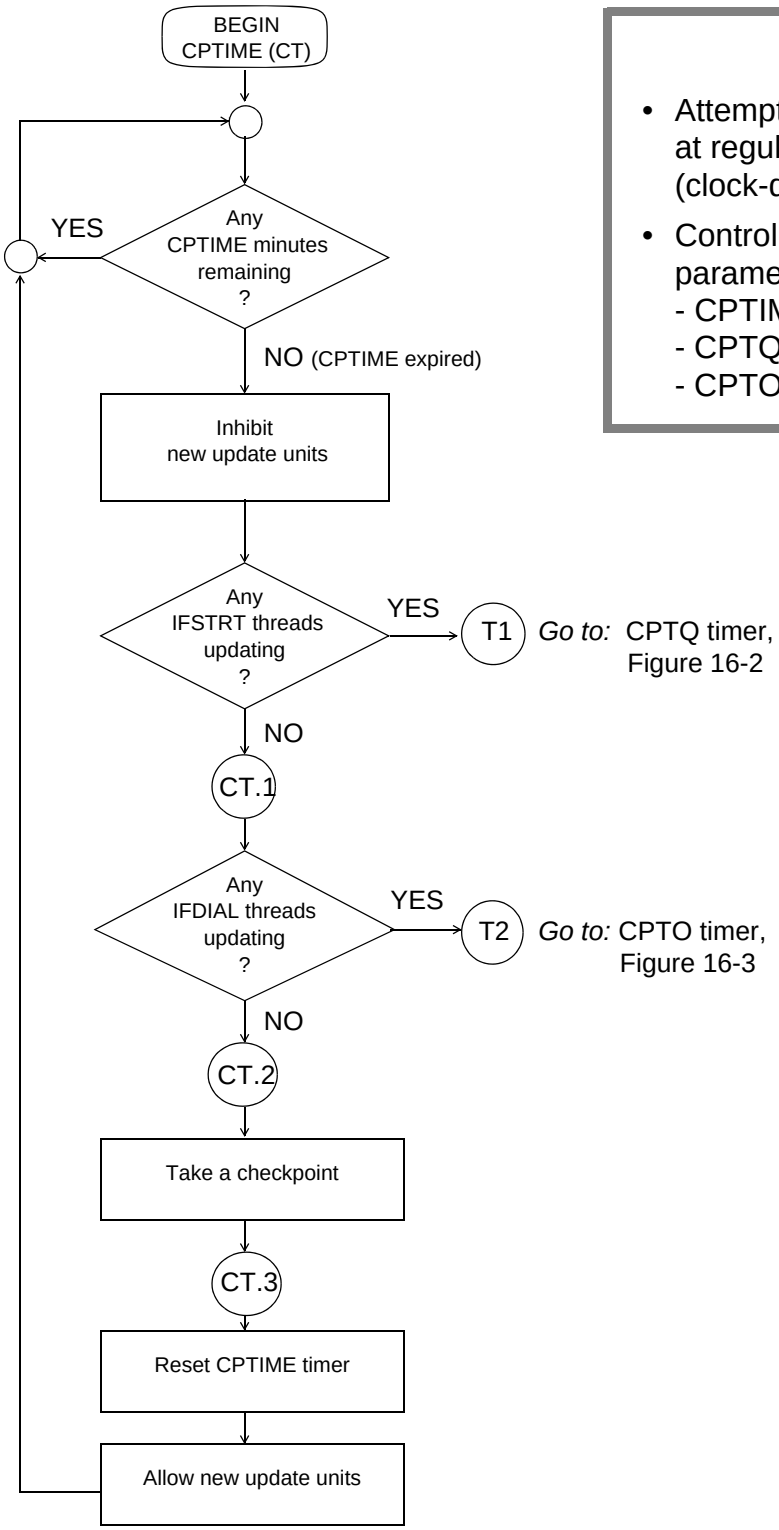
See the following processing flow charts for a detailed description of the steps involved in IFCHKPT checkpointing:

- “Checkpoint Processing steps: IFCHKPT main flow” on page 191
- “Checkpoint processing steps: CPTQ timer” on page 186
- “Checkpoint processing steps: CPTO timer” on page 187
- “Checkpoint processing steps: IFCHKPT time-out” on page 192

Refer to the *Rocket Model 204 Host Language Interface Reference Manual* for more information about IFCHKPT.

Checkpoint processing steps: CPTIME main flow

Figure 16-1. CPTIME checkpointing: main processing flow

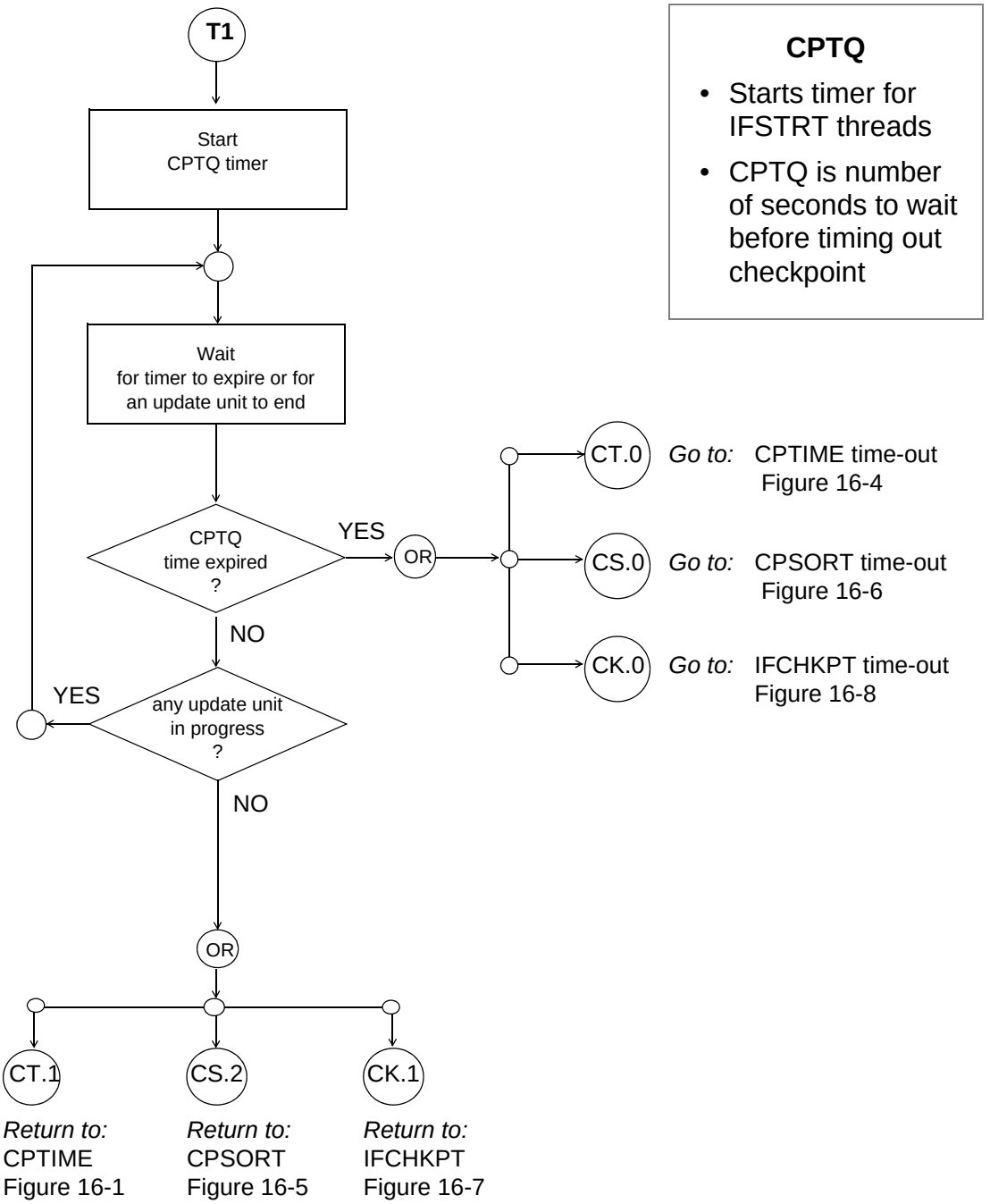


CPTIME

- Attempts to take a checkpoint at regular intervals (clock-driven operation)
- Controlled by user zero parameters:
  - CPTIME
  - CPTQ
  - CPTO

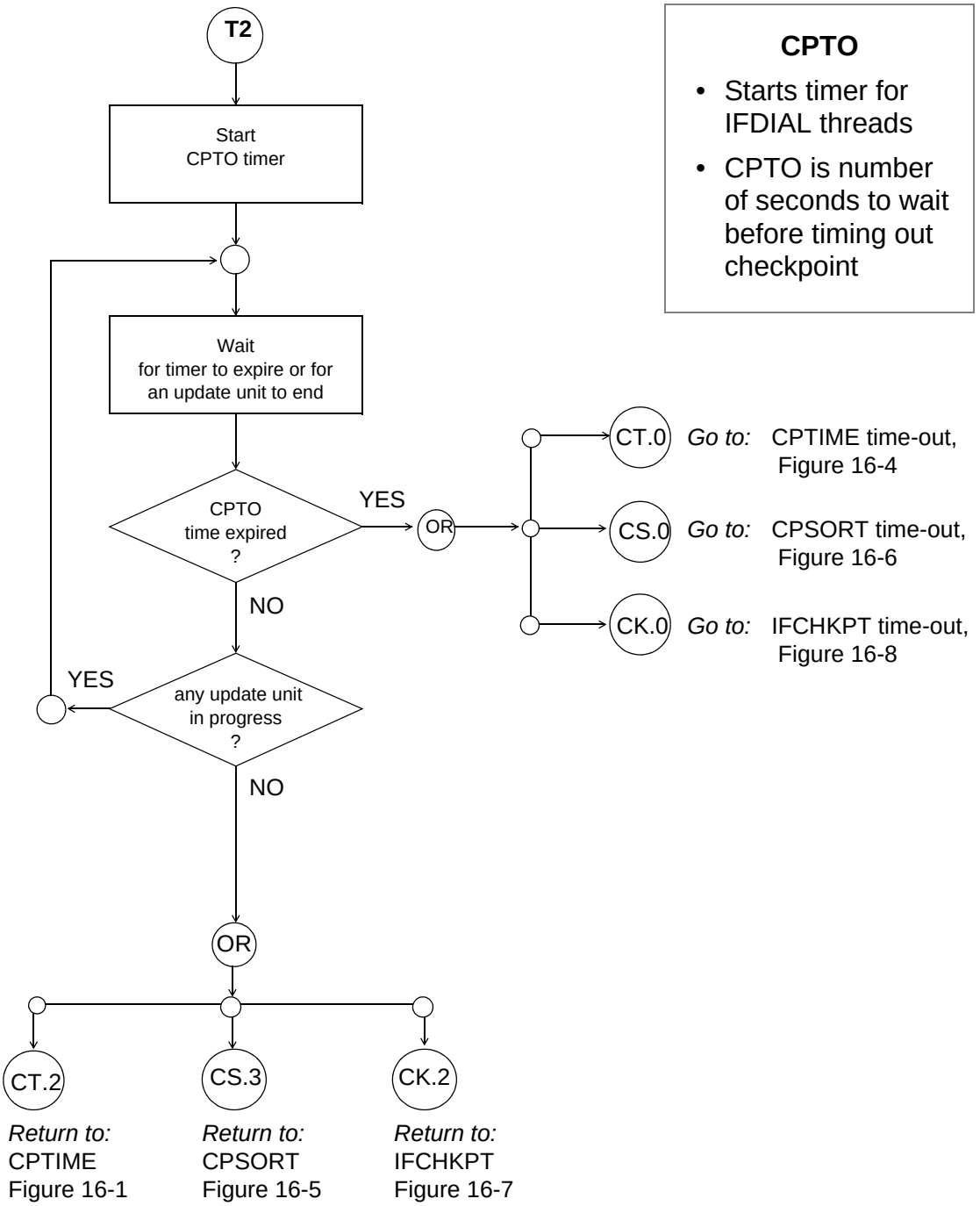
Checkpoint processing steps: CPTQ timer

Figure 16-2. CPTQ checkpointing timer processing flow



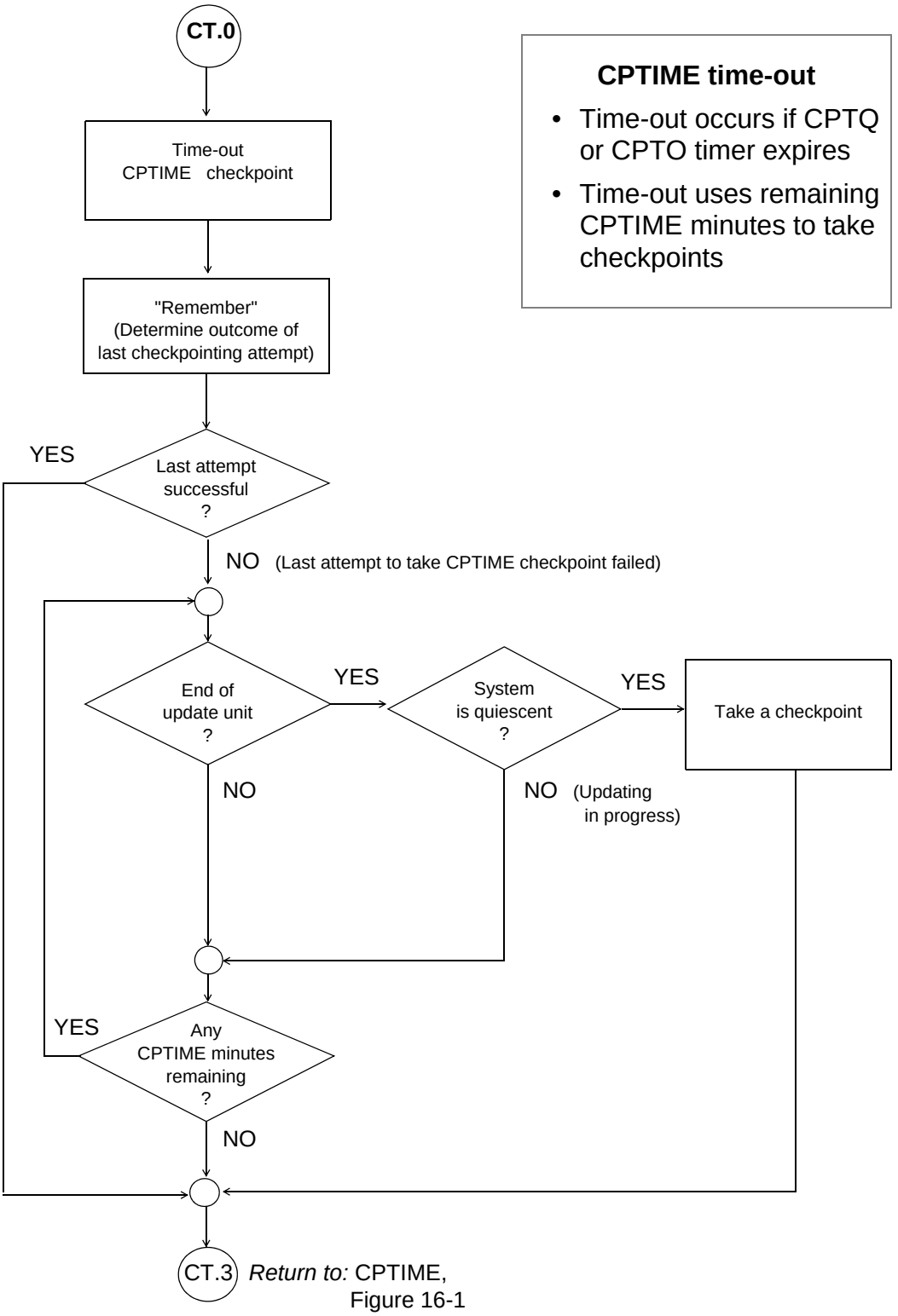
Checkpoint processing steps: CPTO timer

Figure 16-3. CPTO checkpointing timer processing flow



Checkpoint processing steps: CPTIME time-out

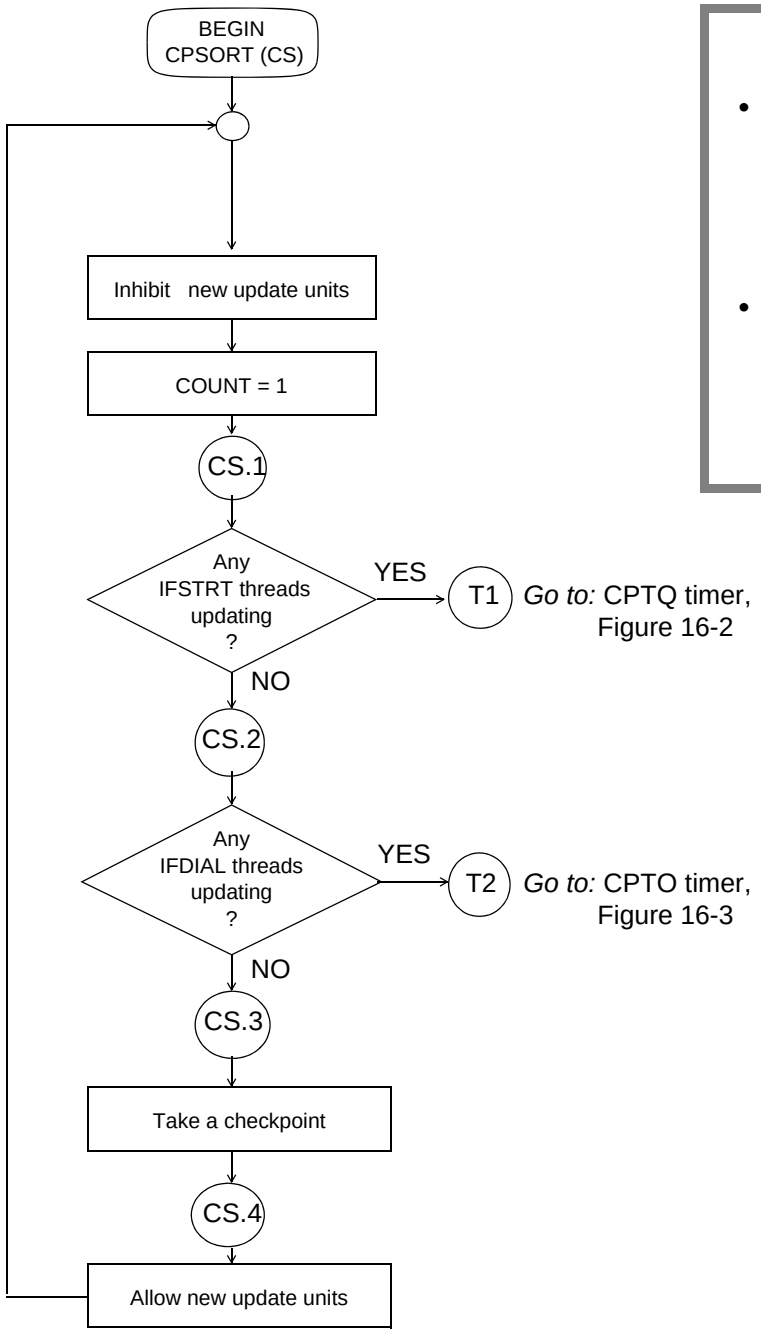
Figure 16-4. CPTIME checkpointing: time-out processing flow



Checkpoint processing steps: CPSORT main flow

Figure 16-5 shows the main flow of CPSORT checkpointing. The CPSORT parameter must be set to a value that is not equal to 0 for CPSORT processing to be enabled; a call to IFSTRT or IFFNSH, as described on page 182, initiates the CPSORT process shown below.

Figure 16-5. CPSORT checkpointing: main processing flow

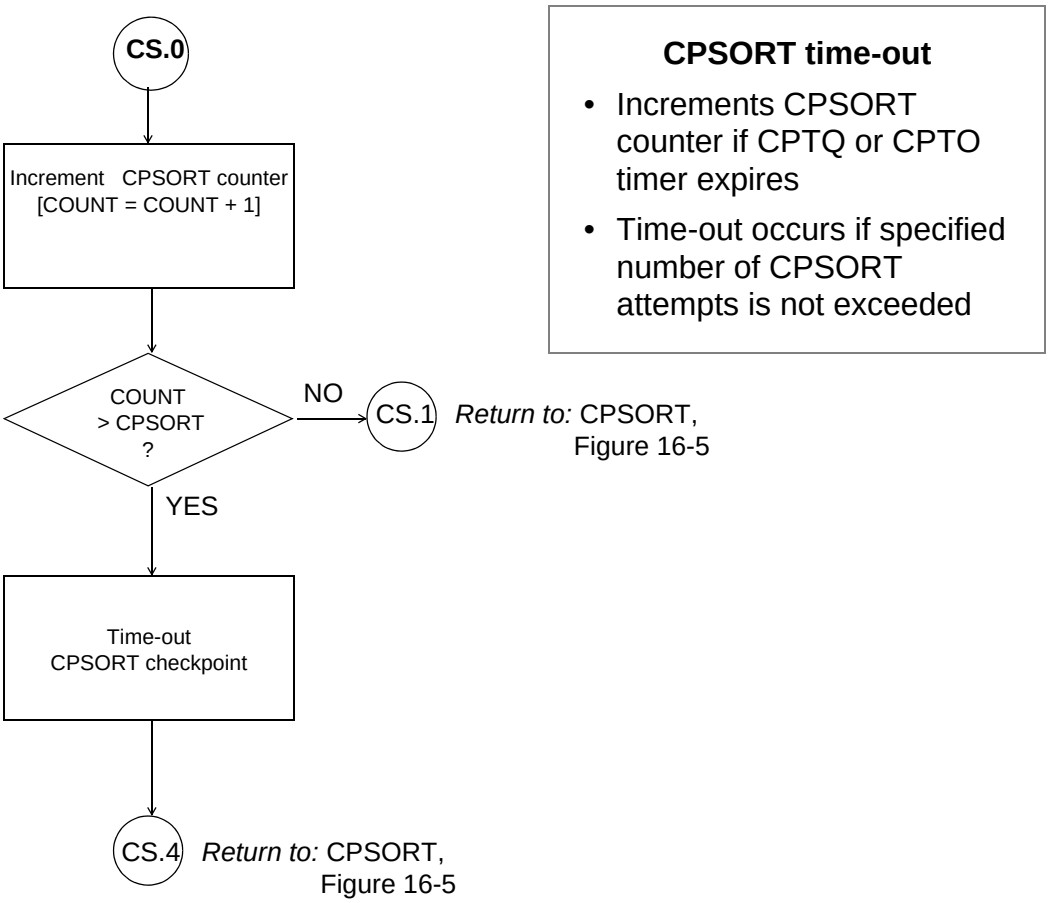


CPSORT

- Attempts to take a checkpoint upon execution of:
  - initial IFSTRT in an HLI job
  - an IFFNSH in any HLI job (HLI calls drive operation)
- Controlled by user zero parameters:
  - CPSORT
  - CPTQ
  - CPTO

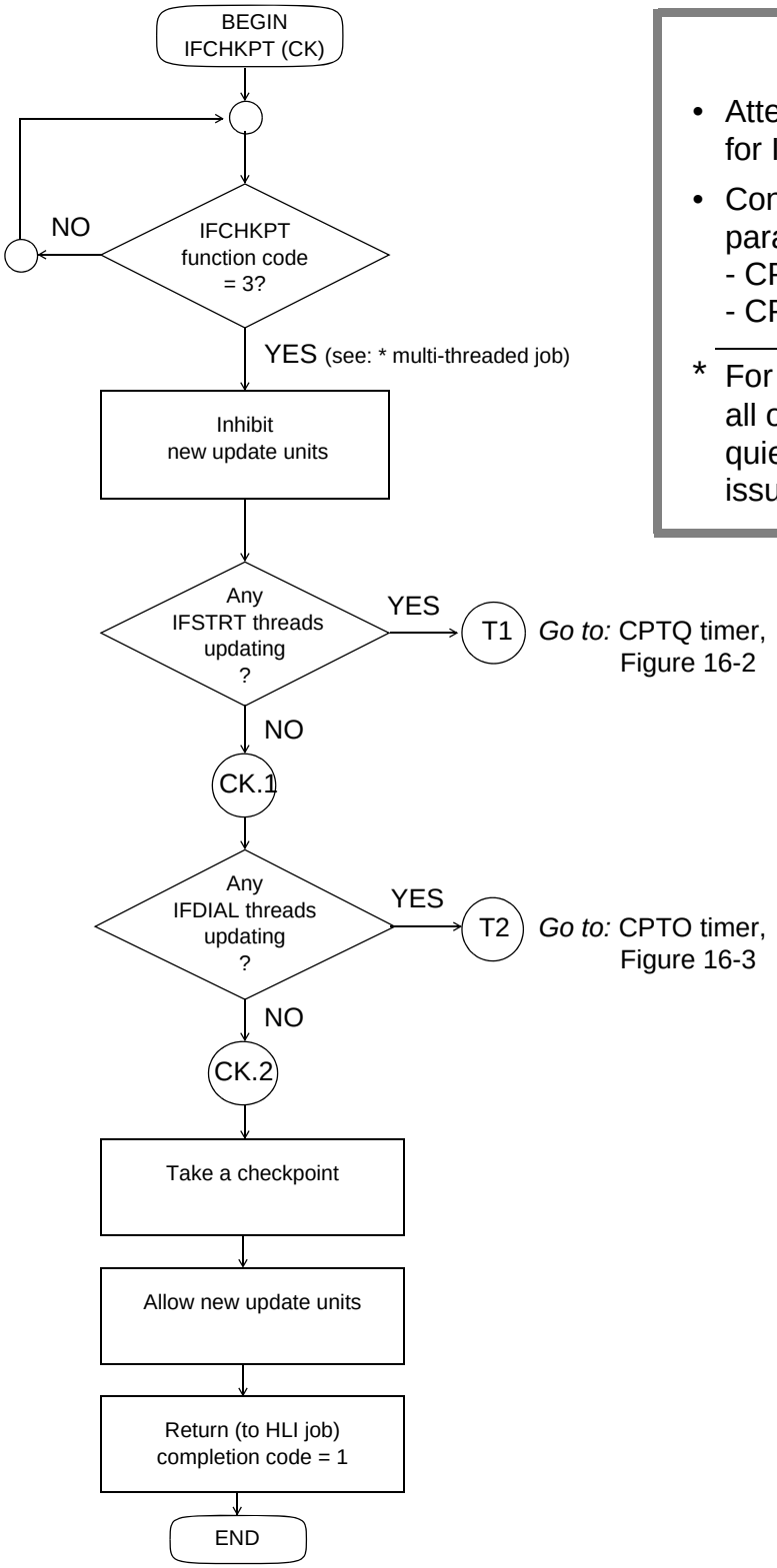
Checkpoint processing steps: CPSORT time-out

Figure 16-6. CPSORT checkpointing: time-out processing flow



Checkpoint Processing steps: IFCHKPT main flow

Figure 16-7. IFCHKPT checkpointing: main processing flow



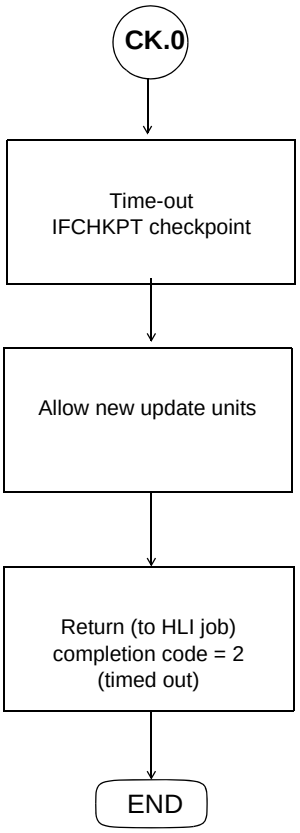
IFCHKPT

- Attempts to take a checkpoint for IFCHKPT function code of 3
- Controlled by user zero parameters:
  - CPTQ
  - CPTO

* For a multi-threaded HLI job, all other threads must first be quiesced by IFCHKPT call issued with function code of 1

Checkpoint processing steps: IFCHKPT time-out

Figure 16-8. IFCHKPT checkpointing: time-out processing flow



IFCHKPT time-out

- Time-out occurs if CPTQ or CPTO timer expires
- Returns completion code of 2 to HLI job

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W