Rocket

Rocket Model 204

Host Language Interface Reference Manual

Version 7 Release 5.0

September 2014
204–75–HLIREF-01

Notices

Edition

Publication date: September 2014
Book number: 204–75–HLIREF-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

3 HLI Job Requirements

7 Field Formatting Options for HLI Calls

About this Manual

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, Assembler, Pascal, and C. This manual describes the HLI jobs and functions that are used with the Host Language Interface.

This manual is a companion to the *Model 204 Host Language Interface Programming Guide*.

Audience

This manual serves as the primary reference source for the application programmer using the Model 204 Host Language Interface facility.

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."

Model 204 documentation set

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/).

Documentation conventions

This manual 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.
`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.
E`DIT`	Partial bolding indicates a usable abbreviation, such as E for EDIT in this example.

1

Introduction to the HLI Facility

Overview

The Host Language Interface facility of Model 204 serves the following principal purposes in a data processing installation:

- Makes the database available to host language programs and programmers, which enables existing systems and organizations to take advantage of the information resources managed by Model 204.
- Improves and accelerates the process of host language system maintenance by providing high-level facilities for database access and update.

Advantages of the HLI facility

Utilizes the unique advantages of Model 204

Many unique advantages of using Model 204 are developed from the logical concepts used in structuring Model 204 databases and from the physical techniques used in organizing and accessing them. Using the Model 204 HLI facility allows you to access the Model 204 database utilizing those underlying structures and methods.

Minimizes the introduction of data dependencies

Function call parameters specify information such as the name of the file to be opened, the criteria by which records are selected, the names

of fields to be retrieved from a record, and the content of data to be stored in an updated record.

The HLI call parameters provide a high-level logical view of the data and minimize the introduction of physical data dependencies into application programs.

Provides flexibility in design of databases and applications

The entire Model 204 system makes the design of databases and applications as flexible as possible and eases the dynamic growth of both databases and applications after implementation.

For example, record composition can vary from record to record within a file, and within the same record over time. Field length can vary in similar fashion. You can define new fields at any time, usually without reloading the databases and without altering application programs.

Ensures database integrity

Use of the Host Language Interface shields you from certain operational parameters and problems that can vary from run to run.

For example, Model 204 maintains data buffering, control of concurrent access to data, protection of data from unauthorized use, and protection and recovery of the database in the event of system failures and some application failures.

To enable external recovery systems and procedures to be coordinated with those built into Model 204, the Host Language Interface also provides functions for the synchronization of checkpointing from application programs.

HLI capabilities

Concurrent processing

With the HLI facility, a host language program using Model 204 can concurrently use other data accessing facilities and can run under the control of teleprocessing systems such as CICS.

Host language programs can also run under the IBM Conversational Monitor System (CMS) within the Virtual Machine Facility (VMF).

The Host Language Interface facility provides a bridge between Model 204 and other systems and between Model 204 databases and other data.

Batch mode operation

Host language programs that use Model 204 operate in batch mode.

Model 204 Host Language Interface programs can share a copy of Model 204 with other HLI application programs and online users in IFAM2, or they can use a private copy in IFAM1 and IFAM4.

Host language programs can run in the same or different job and address space as Model 204 itself. Using the IBM inter-user communication vehicle (IUCV), programs can run in a different virtual machine than the one that hosts Model 204.

31-bit addressing

Model 204 Host Language Interface programs can run with 31-bit addressing.

Subroutine calls to Model 204

Host language programs communicate with Model 204 through the subroutine calls, that is, the HLI function calls, described in this manual.

Each call specifies an operation to be performed by Model 204. A complete transaction is ordinarily accomplished through a sequence of calls, as illustrated on page 4.

Model 204 configurations

Model 204 host language processing

The following Model 204 configurations support host language processing using the HLI facility:

- IFAM1
- IFAM2
- IFAM4

Model 204's Inverted File Access Method (IFAM)

IFAM is an acronym for Inverted File Access Method, which is Model 204's database I/O access mechanism.

IFAM1, IFAM2, and IFAM4 denote different configurations of the Model 204 environment that provide IFAM type access to the Model 204 database from an application program written in a host language such as COBOL, FORTRAN, PL/1, or Assembler, using the Host Language Interface. The host language programmer must design an application to run in one of these environments.

Refer to Chapter 2 for more information about the IFAM1, IFAM2, and IFAM4 environments.

Thread connections to Model 204

Model 204 thread

A host language program starts a thread which provides a connection to Model 204. A thread is a logical connection between Model 204 and the host language application program. The host language program must start at least one thread in order to access the Model 204 database using the HLI facility. A thread corresponds to an IODEV definition in the Model 204 ONLINE.

IFSTRT and IFDIAL threads

There are three types of threads that are available using the HLI facility:

- Single cursor IFSTRT thread
- Multiple cursor IFSTRT thread
- IFDIAL thread

The IFSTRT and IFDIAL threads each utilize a different set of communications protocols which support different types of functionality. In addition, the single and multiple cursor IFSTRT threads allow you to access the Model 204 database in different ways. The host language programmer must code an application corresponding to the type of threads that are started.

Single and multiple cursor IFSTRT threads

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

The differences between single cursor and multiple cursor IFSTRT threads is described in greater detail in “Multiple cursor and single cursor IFSTRT threads” on page 11 and in the *Rocket Model 204 Host Language Interface Programming Guide*.

Note: For host language applications that use IFSTRT threads, we suggest that you use a multiple cursor IFSTRT thread.

Using an IFSTRT thread

Using an IFSTRT thread allows the host language program to specify operations to be performed by Model 204 against the database.

The following set of calls exemplify the Host Language Interface functionality that is available with a multiple cursor IFSTRT thread:

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:
     a. Fetch the order name (IFFTCH).
     b. 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).

Using an IFDIAL thread

Using an IFDIAL thread allows the host language program to transfer data to and from Model 204 using line-by-line terminal emulation mode.

The following set of calls exemplify the Host Language Interface functionality that is available with an IFDIAL thread:

1. Start a Host Language Interface thread (IFDIAL).
2. Send a line input to Model 204 (IFWRITE).
3. Get a line of output from Model 204 (IFREAD).
4. Send an attention interrupt signal (IFATTN).
5. End the thread (IFHNGUP).

For more information

Refer to Chapter 2 for more information about using the IFSTRT and IFDIAL threads in HLI jobs. Refer to Chapter 5 for information about the calls that are available using the two different types of threads.

For examples of application program code using the HLI calls, refer to the appendixes at the back of this manual. For information describing how to code applications using IFSTRT and IFDIAL threads, refer to the *Rocket Model 204 Host Language Interface Programming Guide*.

Part I
HLI Jobs

Part I describes in detail the information that is required to set up and execute a job to use the Model 204 Host Language Interface. Use the information to identify which type of HLI job to run and to structure your HLI job with all of the necessary components.

2

HLI Job Design Factors

Overview

This chapter gives the application programmer an overview of the IFAM1, IFAM2, and IFAM4 Model 204 environments that support HLI processing.

This chapter presents the factors that determine the type of job to run using the Model 204 Host Language Interface facility. Each type of job provides certain processing capabilities within a particular system environment.

For more information

Refer to Chapter 3 for details about setting up and running HLI jobs. Refer to the *Rocket Model 204 Host Language Interface Programming Guide* for more information about coding applications.

Processing in different Model 204 environments

Table 2-1 summarizes host language application program processing in the different Model 204 environments.

Table 2-1. Summary of HLI processing environments

Item	IFAM1	IFAM2	IFAM4
Operating system	z/OS, VSE, CMS	z/OS, VSE, CMS	z/OS
Teleprocessing monitors	—	CICS INTERCOMM	—

Table 2-1. Summary of HLI processing environments

Item	IFAM1	IFAM2	IFAM4
Model 204 copy	Private	Shared	Private
Threads	Single	Multiple	Multiple
Protocol	IFSTRT IFDIAL	IFSTRT IFDIAL	IFSTRT
Enqueuing, with other application programs	File	Record, with those sharing Model 204	File
Recovery	RESTART (rollback only)	RESTART	RESTART
Performance	—	Cross-region overhead	Scheduler overhead

Refer to the *Rocket Model 204 Host Language Interface Programming Guide* for more information about enqueuing and recovery in the HLI processing environment.

Call protocols

IFSTRT and IFDIAL protocols

The Model 204 Host Language Interface facility supports two types of call protocols: IFSTRT and IFDIAL.

Each of the protocols is available using a different type of thread connection to Model 204. The protocols operate differently and provide different types of host language functionality.

To use IFSTRT protocols, you start an IFSTRT thread (by coding an IFSTRT or IFTSTRTN call in your HLI application). To use IFDIAL protocols, you start an IFDIAL thread (by coding an IFDIAL or IFDIALN call). The functionality that is available using a particular thread is determined by the type of protocol that it uses.

See Chapter 6 for descriptions of the IFSTRT and IFDIAL calls. For more information about coding applications using the two protocols, refer to the *Rocket Model 204 Host Language Interface Programming Guide*.

IFDIAL thread

An IFDIAL thread provides a line-at-a-time terminal type interface between Model 204 and a host language program that is running in batch. An IFDIAL connection allows an HLI application to issue commands (such as LOGWHO, MONITOR, and LOGCTO) and run SOUL requests, and to receive responses from Model 204.

IFSTRT thread

An IFSTRT thread provides a user interface between Model 204 and a program that uses the Host Language Interface (which runs in batch).

IFSTRT protocols allow an application that is written in a host language to issue calls to Model 204 that perform functions against the database which are similar to Model 204 commands and SOUL statements.

Each IFSTRT connection supports either multiple cursor or single cursor IFSTRT functionality.

Multiple cursor and single cursor IFSTRT threads

An IFSTRT or IFSTRTN call starts either a multiple cursor or single cursor thread by setting a thread type parameter. A multiple cursor type thread supports only multiple cursor HLI functionality, and a single cursor type thread supports only single cursor HLI functionality.

A multiple cursor IFSTRT thread supports host language program access to multiple files and to multiple record sets, using a single thread. By contrast, each single cursor IFSTRT thread allows only single file, single record set database access.

With a multiple cursor IFSTRT thread, the file, set, or record that is specified in the HLI call is the one that is current for processing. In contrast, the current item to be processed on a single cursor IFSTRT thread is always relative to the last file opened, the last set created, and the record last referenced.

Certain IFSTRT calls are supported for use only on a multiple cursor or a single cursor IFSTRT thread, while other IFSTRT calls are supported for use on both multiple cursor and single cursor IFSTRT threads. The application programmer must code only the calls, specifications, and corresponding program logic that are valid for use with the particular type of IFSTRT thread that is started.

See “Multiple cursor and single cursor IFSTRT threads” on page 11 for an overview of the calls that are valid for use with a multiple cursor IFSTRT thread. Refer to the *Rocket Model 204 Host Language Interface Programming Guide* for more information about multiple cursor IFSTRT functionality.

Multiple cursor IFSTRT thread

For an application that requires access to multiple files concurrently, Rocket recommends that you use a multiple cursor IFSTRT thread and that you do not run a multithreaded job (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, however, that if you are performing update processing using one thread, checkpointing is easier on a multiple cursor IFSTRT thread.

Recovery considerations for single and multicursor IFAM2 threads

Single cursor IFAM update threads differ from SOUL and multicursor IFAM threads in that the single cursor threads start an update unit during their IFSTRTN call and end the update unit when IFFNSH is issued. This means that with single cursor IFAM threads active, all transaction checkpoints fail—meaning: time-out. For these threads, you must use IFCHKPT calls to end single cursor update units, wait for a transaction checkpoint attempt, and then start new single cursor update units.

Multicursor IFAM, like SOUL, starts an update unit at the first file update and ends the update unit at update commit (IFCMMT ccall in IFAM). Therefore, multicursor IFAM threads do not require IFCHKPT calls to let a transaction checkpoint proceed, although they may issue IFCHKPT calls to initiate transaction checkpoint attempts or query transaction checkpoint status.

You can use sub-transaction checkpoints to recover files in active update by either single or multicursor IFAM threads; they require no form of IFCHKPT call. When IFAM threads on a sub-transaction enabled Online issue IFCHKPT calls, they are requesting transaction checkpoints. If no IFCHKPT calls are made, IFAM updates are recoverable with sub-transaction checkpoints.

System configurations

IFAM1

The Host Language Interface facility is supported in the IFAM1 configuration running under an z/OS, VSE, or CMS operating system.

IFAM1 supports a single program (that is, a single user), in a single region (that is, the IFAM1 job runs in a region that is separate from the Model 204 Online region using a private copy of Model 204), using a singly-threaded connection (that is, using one thread, either IFSTRT or IFDIAL).

IFAM1 provides the most efficient system access which is useful for testing program modules and applications.

IFAM2

The Host Language Interface facility is supported in the IFAM2 configuration running under an z/OS, VSE, or CMS operating system. IFAM2 is supported in CICS.

IFAM2 supports one or more programs (that is, multiple users sharing the same copy of Model 204), running in separate regions or machines or partitions (that is, the user program runs in a separate region from Model 204 requiring communications between the regions), using a multithreaded connection (that is, using one or more threads, and except for IFDIAL under CMS, can be both IFSTRT and IFDIAL threads).

Note: A batch utility program, BATCH2, is provided by Rocket for use in the IFAM2 environment. For information about the BATCH2 utility, refer to the Rocket Model 204 documentation wiki:

http://m204wiki.rocketsoftware.com/index.php/Program_communications_facilities#BATCH2_facility

IFAM4

The Host Language Interface facility is supported in the IFAM4 configuration running under the z/OS operating system.

IFAM4 supports a single program (that is, a single user), running in a single region (that is, the user program and Model 204 run in the same region), using a multithreaded connection (that is, using one or more IFSTRT threads).

IFAM4 provides the most core-efficient way to run a multithreaded HLI application and is useful in certain test and batch production situations.

Setting up an HLI job

You must set up your HLI job for the particular Model 204 configuration (that is, IFAM1, IFAM2, or IFAM4) in which it will run. To connect to Model 204, the job references a particular link module or set of link modules.

For detailed information about setting up jobs, refer to Chapter 3.

Call protocols and Model 204 configurations

Each of the HLI protocols may be used in several different Model 204 configurations as summarized below.

Protocol	Model 204 configuration
IFSTRT	IFAM1, IFAM2, IFAM4
IFDIAL	IFAM1, IFAM2

For more information

The diagrams of HLI application processing in the IFAM1, IFAM2, and IFAM4 configurations on the following pages show the basic components of each configuration.

For more information about the Model 204 configurations, see the Rocket Model 204 documentation wiki:

http://m204wiki.rocketsoftware.com/index.php/Model_204_configurations_and_operating_environments

IFAM1 configurations

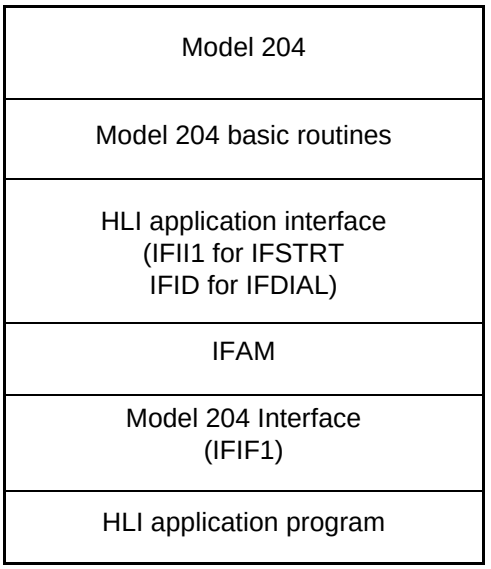
In IFAM1, the HLI batch application program and Model 204 run together in the same region, which is separate from the Online region. There are two basic configurations of the IFAM1 processing environment, one for z/OS and VSE, and one for CMS.

IFAM1 under z/OS and VSE

Figure 2-1 shows the logical view of the IFAM1 processing environment for an HLI application running under z/OS or VSE. Figure 2-1 shows the application program linking to Model 204 with the IFIF1 interface (IFIF1OS for z/OS or IFIF1DOS for VSE).

Note that the Model 204 system manager must build the IFAM1 link with the IFID module included for a user to allow an HLI program to access Model 204 using an IFDIAL thread.

Figure 2-1. IFAM1 under z/OS and VSE



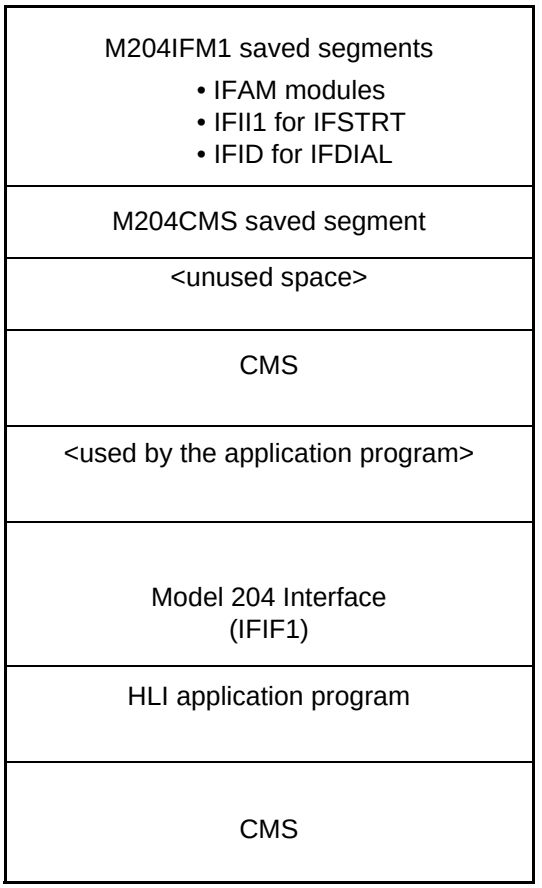
IFAM1 under CMS

Figure 2-2 shows the logical view of the IFAM1 processing environment for an HLI application running under CMS.

The HLI application program is loaded to include a library (not shown) that contains the IFIF1 object module (IFIF1CMS) which must be linked to the

program as shown below. Figure 2-2 shows the saved segments in a common storage area separate from the user's CMS application.

Figure 2-2. IFAM1 under CMS



IFAM2 configurations

In IFAM2, the HLI application program runs as a job in a separate region from Model 204, sharing a copy of Model 204 with other users who access the database from host language and SOUL applications.

IFAM2 requires communications between regions, using the Model 204 Cross-Region Access Method (CRAM).

There are two basic configurations of the IFAM2 processing environment, one for z/OS and VSE, and one for CMS.

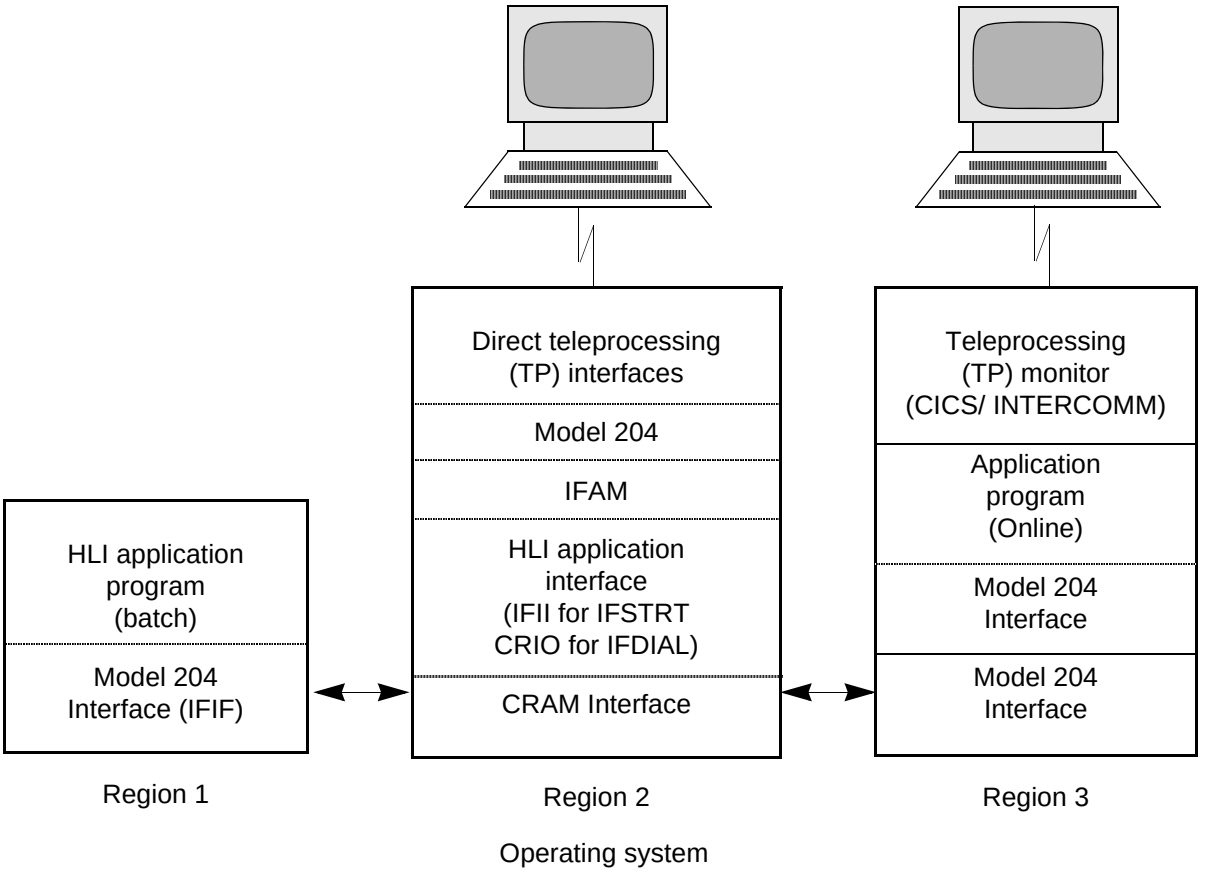
Logical view of IFAM2 under z/OS and VSE

Figure 2-3 shows the logical view of the IFAM2 processing environment for an HLI application running under z/OS or VSE.

Figure 2-3 shows the following components:

- A host language program is running in Region 1 as a batch job. The HLI application communicates to Model 204 through CRAM. The HLI application program links to Model 204 with the IFIF object module.
- Model 204 resides in Region 2 and runs with the interfaces that enable CRAM communications and teleprocessing.
- An Online application is running in Region 3 under a teleprocessing monitor. The Online application also communicates to Model 204 through CRAM.

Figure 2-3. IFAM2 under z/OS and VSE



IFAM2 under CMS

Figure 2-4 shows the logical view of the IFAM2 processing environment for an HLI application running under CMS.

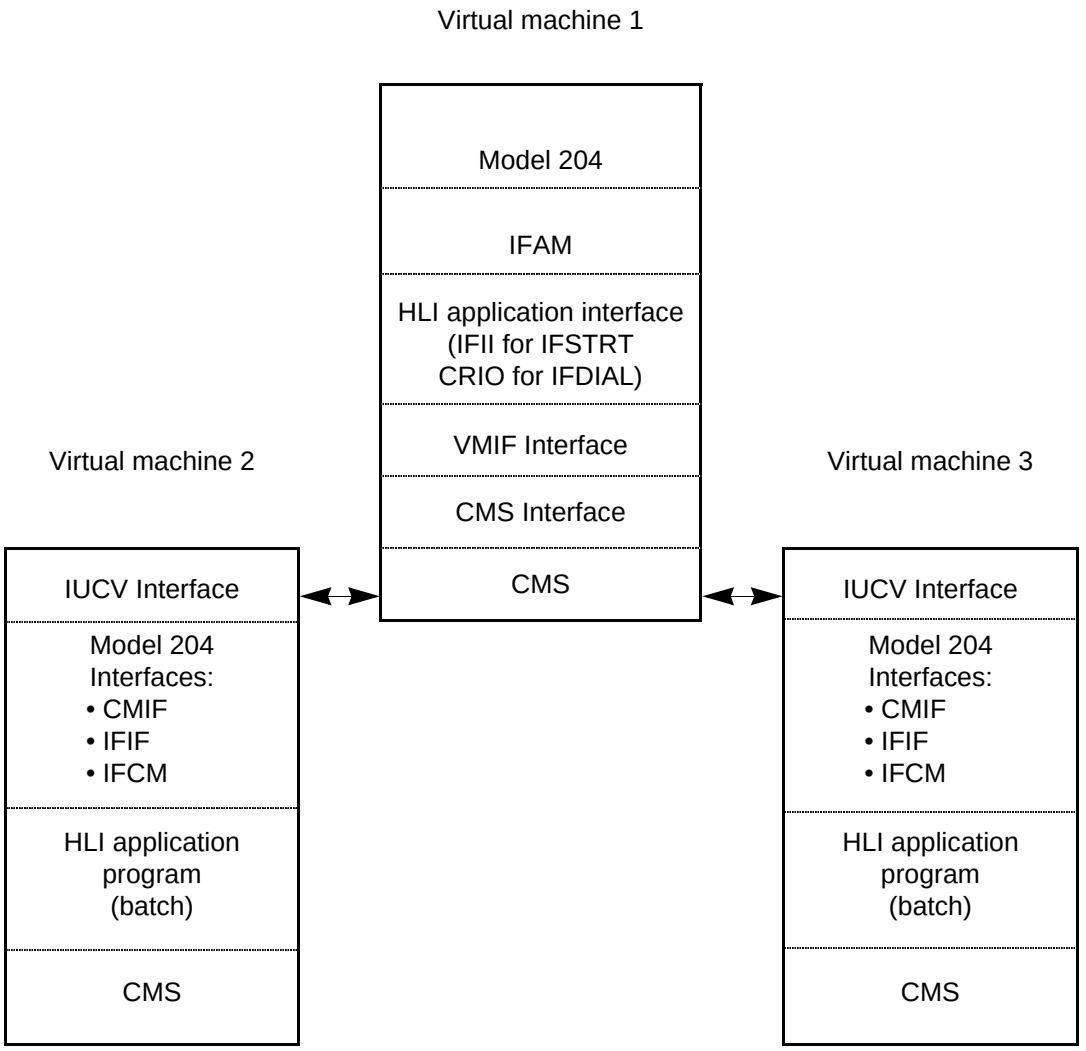
The IBM inter-user communication vehicle (IUCV) allows a host language program running under CMS to communicate with a Model 204 system running in a separate virtual machine under CMS.

The HLI application program is loaded to include a library (not shown) that contains the following object modules (shown in Figure 2-2) which must be linked to the program: CMIF, IFIF, IFCM, and IUCV.

Figure 2-4 shows the following components:

- Model 204 runs under the CMS interface in virtual machine 1 with the VMIF interface that enables IUCV communications.
- A host language program is running in virtual machine 2 as a batch job. The HLI application communicates to Model 204 through IUCV.
- Another host language application is running in virtual machine 3 as a batch job, and communicates to Model 204 through IUCV.

Figure 2-4. IFAM2 under CMS



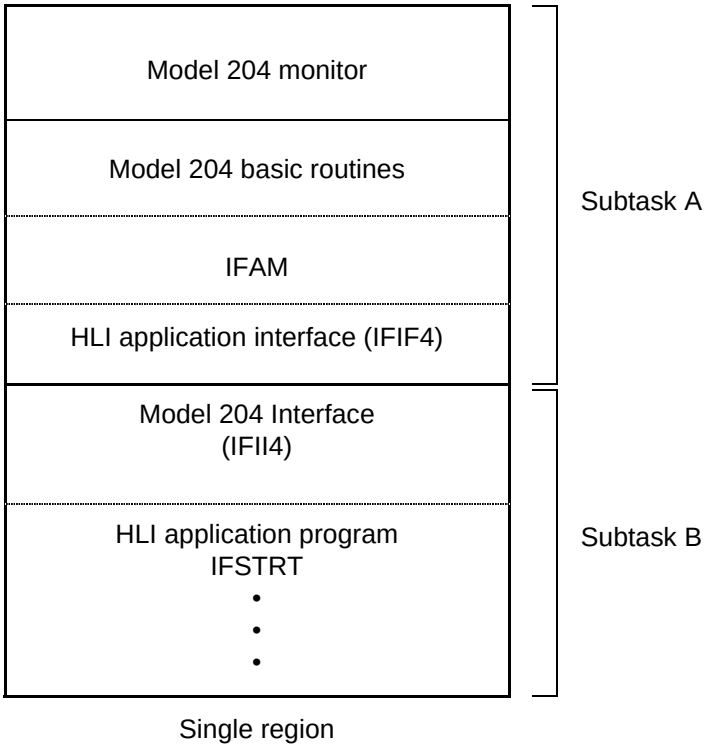
IFAM4 configuration

There is one basic configuration of the IFAM4 processing environment, for z/OS. The HLI application program and Model 204 run in the same region as separate subtasks of a single batch job.

Logical view of IFAM4 under z/OS

Figure 2-5 below shows the logical view of the IFAM4 processing environment for an HLI application running under z/OS.

Figure 2-5. IFAM4 under z/OS



Converting HLI applications for different environments

Running in different Model 204 environments

In many cases, applications written to operate with the Host Language Interface in one Model 204 environment require little or no change to operate in another environment.

Before you can run a host language program with any version of the Host Language Interface, you must link-edit it with one or more Model 204 modules. In general, you can convert a host language application program which runs in one environment to run in another by linking the appropriate interface module.

Use the guidelines in the following sections to convert HLI applications to run in the different Model 204 environments that support HLI processing.

Applications using different call protocols

Note the following restrictions on converting applications that use different call protocols or different types of threads:

- You cannot convert an application program using IFSTRT protocols to IFDIAL, and vice versa.
- For applications using IFSTRT threads, a user may choose to convert applications from single cursor IFSTRT processing to multiple cursor IFSTRT processing. Note, however, that this conversion requires modifying existing code in both the calls and the logic.

Applications using IFSTRT threads, from IFAM2 to IFAM4

Converting an IFAM2 application program that uses IFSTRT threads to run in IFAM4 does not require recompilation of the IFAM2 program. Only the interface module changes.

To convert an IFAM2 application program to run in IFAM4, modify the link-edit job step to reference the IFIF4 link module (instead of IFIF) and run IFAM4 to execute the application program.

Note that IFAM4 applications cannot use IFDIAL. You cannot convert an IFAM2 application program that is using an IFDIAL connection to run in IFAM4. See “Applications that require coding changes” on the next page.

Applications from IFAM4 to IFAM2

With simple link-edit modifications, IFAM4 applications can be converted to run in IFAM2. To convert an IFAM4 application program to run in IFAM2, modify the link-edit job step to reference the IFIF link module (instead of IFIF4) and execute the application program.

Converting from IFAM4 to IFAM2 is useful for allowing multiple applications or copies to share a single copy of Model 204.

Applications that require coding changes

Certain of the HLI calls that are supported in one environment may not be supported in another. For example, the same calls that are valid for use in IFAM2 can be used in IFAM4, and vice versa, with the following exceptions:

- IFSTRTN, which can be used only in IFAM2
- IFABXIT and IFCSA, which are valid only for IFAM2 under CICS

There are differences in using certain calls in the different environments. For example, there are two different forms of IFSTRT (using different parameter lists); one is for IFAM1 and the other is for IFAM2 and IFAM4.

In general, you must modify an application program to convert the following types of HLI applications (using IFSTRT threads):

- From IFAM1 to IFAM2 or IFAM4
- From IFAM2 or IFAM4 to IFAM1

Use the following guidelines when converting to or from IFAM1:

- You can use IFCALL (with function numbers) only in IFAM2 or IFAM4.
- Note that IFGERR functions somewhat differently in IFAM2 and IFAM4 than in IFAM1.
- The following calls are valid for use only in IFAM1: IFSETUP (which can only be used with an IFDIAL connection) and IFLOG.
- In general, you cannot use calls that perform thread management or checkpointing functions in IFAM1, including: IFDTHRD, IFSTHRD, and IFCHKPT.

Refer to Chapter 6 for information about individual calls and their usage.

3

HLI Job Requirements

Overview

This chapter details the job requirements for the application programmer using the Host Language Interface facility in the IFAM1, IFAM2, and IFAM4 Model 204 environments.

Contents of this chapter

There are three broad categories of batch jobs that you can run corresponding to the IFAM1, IFAM2, and IFAM4 Model 204 environments that support HLI processing. This chapter presents job information in the following order:

- IFAM1 jobs
  - Under z/OS, VSE, CMS
- IFAM2 jobs
  - Under z/OS, VSE, CMS
  - Using the CICS interface

  Note: The INTERCOMM interface (supporting the use of Teletype and 3270 terminals in line-at-a-time mode) is no longer supported as of Model 204 version 7.5.
- IFAM4 jobs (under z/OS only)

This chapter describes each type of job and includes examples of job setup and execution in the different operating environments.

For more information

See Chapter 6 for a detailed description of HLI calls and call parameters. Refer to the appendices at the back of this manual for examples of applications using HLI calls.

IFAM1 jobs

IFAM1 applications can run under z/OS, VSE, or CMS operating systems. An IFAM1 job involves a two-step procedure: step one is to link-edit the application program and step two is to run it.

Examples of the IFAM1 job steps are provided in the sections which follow. Sample setups are shown for each of the operating systems. The following topics are covered for each system:

- Link-editing the application program
- Running the application program

See page 31 for a description of the control statements that are used with an IFAM1 job. See page 32 for more information about applications in the IFAM1 environment.

IFAM1 dynamic loading

Rocket recommends that the IFAM1 application load the Model 204 portion of the application at runtime.

Invoke IFAM1 dynamic loading by link-editing the IFAM1 application program with the appropriate module:

- IFIF1OS for z/OS
- IFIF1DOS for VSE
- IFIF1 for CMS

See the examples of link-editing with IFIF1OS for z/OS on page 25 and with IFIF1DOS for VSE on page 28. Note that the IFIF1 module is automatically invoked at runtime for CMS.

Note: In order to use the dynamic loading facility, the Model 204 system manager must initially build the IFAM1 load module for your installation site.

Contact your Model 204 system manager to get the name of the Model 204 load library that is used to run IFAM1 programs at your site. For more information about creating the IFAM1 module, refer to the Rocket Model 204 installation instructions for your operating system.

Advantages of dynamic loading

The IFAM1 dynamic loading facility reduces application maintenance and storage by allowing the Model 204 IFAM1 interface to dynamically load the IFAM1 code at runtime.

Dynamic loading provides the application programmer with the easiest and most efficient method for running IFAM1 applications and is the preferred method recommended by Rocket.

Note that the direct link-edit method which statically link-edits IFAM1 applications with Model 204 routines can be used by the application programmer to run IFAM1 jobs. However, using this method requires that:

- All IFAM1 application programs must be link-edited again each time any Model 204 routines change.
- Each IFAM1 application must carry its own copy of all the Model 204 routines, thereby increasing the application's load module size and load library disk space requirements.

Examples of directly link-editing IFAM1 applications are not provided in this document.

IFAM1 jobs: Compiling under Enterprise PL/I for z/OS

When compiling a PL/I application under the Enterprise PL/I for z/OS compiler, the following compiler parameter is required:

```
DEFAULT(LINKAGE(SYSTEM))
```

This causes the parameter list to be built in the same way that it was built by the old compilers (including turning on the high-order bit of the address of the last parameter).

For example:

```
//PLICMPL EXEC PGM=IBMZPLI,PARM='OBJECT,OPTIONS,
//             DEFAULT(LINKAGE(SYSTEM))',REGION=512K,...
```

If this compiler option is not specified, subsequent executions of the application will fail with 0C4 abends.

IFAM1 jobs: Link-editing under z/OS

Link with IFIF1OS

To link-edit your IFAM1 job running under z/OS, link the IFAM1 application with the Model 204 module IFIF1OS using the INCLUDE CCA statement in SYSLIN DD.

Figure 3-1 shows an example of the JCL that may be used to link-edit an IFAM1 application to build the application load module. In this example, the program name is IFAMTEST.

Figure 3-1. Example of z/OS JCL to Link-Edit Application (IFAM1)

```
//LINKAPPL JOB,LINKAPPL,MSGLEVEL=(1,1),CLASS=T,MSGCLASS=C
//LINK EXEC PGM=IEWL,PARM='LIST,MAP,LET,NCAL,SIZE=(250K,150K)'
//SYSPRINT DD SYSOUT=C
//SYSUT1 DD UNIT=WORK,SPACE=(TRK,(40,20))
//USEROBJ DD DSN=LOCAL.M204.OBJLIB,DISP=SHR
//OBJLIB DD DSN=M204.V220.OBJLIB,DISP=SHR
//SYSLMOD DD DSN=LOCAL.M204.IFAM1.APPLIC,DISP=OLD
//SYSLIN DD *
INCLUDE USEROBJ(IFAMTEST)
INCLUDE OBJLIB(IFIF1OS)
ENTRY IFAMTEST
NAME IFAMTEST(R)
/*
```

Note: Before the individual Host Language Interface application programmer can link an IFAM1 application, the Model 204 system manager must build the IFAM1 load module containing the IFII1 member. Contact your Model 204 system manager to get the name of the Model 204 object and load libraries that are used to link-edit and run IFAM1 programs at your site.

IFAM1 jobs: Running under z/OS

Execute with dynamic loading

Once you have link-edited the application program, run the IFAM1 application in z/OS.

Figure 3-2 shows a sample excerpt of the JCL that may be used to run an IFAM1 application which was link-edited as shown in Figure 3-1. In this example, the program name is IFAMTEST.

Figure 3-2. Example of z/OS JCL to Run Application (IFAM1)

```
//RUNIFAM1 JOB,'RUNIFAM1',MSGLEVEL=(1,1),MSGCLASS=C,CLASS=T
//*
//IFAM1EXEC PGM=IFAMTEST
//*
//STEPLIB DD DSN=LOCAL.M204.IFAM1.APPLIC,DISP=SHR
// DD DSN=M204.V220.LOADLIB,DISP=SHR
//CCAAUDITDD SYSOUT=C
//CCAPRINTDD SYSOUT=C
//CCASNAP DD SYSOUT=C
//SYSUDUMPDD SYSOUT=C
//CCATEMPDD DISP=NEW,UNIT=SYSDA,SPACE=(TRK,20)
//CCASTATDD DSN=M204.CCASTAT,DISP=OLD
•
```

```
•...application program DD statements
•
```

Using the STEPLIB statement

The STEPLIB file points to the load module libraries where the application program and the Model 204 IFAM1 interface program reside.

This statement must specify the load module library or libraries that contain IFAM1 and the Host Language Interface user's application program.

Note: One of the following requirements must be met for IFAM1 applications:

- M204XSVC must be installed as an SVC and that SVC number must be passed to Model 204 in the IFSTRT or IFLOG call as the value of the XMEMSVC parameter. XMEMOPT must also be set to 2 or 4, or
- The load library where the IFAM1 application resides (and all run-time concatenated libraries) must be APF authorized.

Unless one of these requirements is met, IFAM1 applications will terminate with:

```
MODEL 204 IS NOT AUTHORIZED FOR THIS CPUID: 0000000000000000
```

Use the DSN of the Model 204 load library that is used to run IFAM1 programs at your site. If the Model 204 load module library and application load library are in separate libraries, use concatenated data set or file definition statements.

Using the EXEC statement

You can specify the following parameters in the EXEC statement:

- PGM — Required; Indicates the name of the application program being run. Note that application programs cannot be called IFAM1.
- REGION — Optional; Indicates the size of the memory area to be allocated for IFAM1 and the application. Note that REGION may be specified depending on your site's Model 204 configuration. Refer to the Rocket Model 204 documentation wiki for information about setting the REGION parameter:

  http://m204wiki.rocketsoftware.com/index.php/Defining_the_runtime_environment_(CCAIN)#Runtime_environment_specifications
- TIME — Optional; Indicates how much time the application program and Model 204 together can use before being cancelled by the operating system. TIME depends on the requirements of the application.
- PARM — Optional; Indicates any application runtime parameters to be set.

See page 31 for more information about the job control statements that are used with an IFAM1 job.

IFAM1 jobs: Link-editing under VSE

Link with IFIF1DOS

To link-edit your IFAM1 job running under VSE, link the IFAM1 application with the Model 204 module IFIF1DOS using the INCLUDE statement in OPTION CATAL.

Figure 3-3 shows an example of the JCL that may be used to link-edit an IFAM1 application to build the application load module. In this example, the program name is IFAMTEST.

Figure 3-3. Example of VSE JCL to Link-Edit Application (IFAM1)

```
// JOB LINKAPPL FOR IFAM1TEST
// DLBL PRIVLIB,'PRIV.USER.LIBRARY'
// EXTENT,SYSnnn,volser,balance of extent information
// LIBDEF CL,TO=PVTCL
// DLBL M204LIB,'M204.PROD.LIBRARY'
// EXTENT,SYSnnn,volser,balance of extent information
// LIBDEF OBJ.SEARCH=(M204LIB.V220,PRIVLIB.XXXX)
// LIBDEF PHASE.CATALOG=PRIVLIB.xxxx
// OPTION CATAL
  PHASE IFAMTEST,*
  INCLUDE usermodule
  INCLUDE IFIF1DOS
/*
// EXEC LNKEDT
/&
```

Note: Before the individual Host Language Interface application programmer can link an IFAM1 application, the Model 204 system manager must build the IFAM1 load module containing the IFII1 member. IFAM1 is provided as a phase in the distribution sublibrary and may need to be relinked by the Model 204 system manager.

Contact your Model 204 system manager to get the name of the Model 204 object and load sublibraries that are used to link-edit and run IFAM1 programs at your site.

IFAM1 jobs: Running under VSE

Execute with dynamic loading

Once you have link-edited the application program, run the IFAM1 application in VSE.

Figure 3-4 shows a sample excerpt of the JCL that may be used to run an IFAM1 application which was link-edited as shown in Figure 3-3. The EXEC

statement specifies the name of the application program to be executed. In this example, the program name is IFAMTEST.

Figure 3-4. Example of VSE JCL to run application (IFAM1)

```
// JOB DOS IFAM1 DYNAMIC LINKEDIT TEST
// DLBL PRIVLIB,'PRIV.USER.LIBRARY'
// EXTENT,SYSnnn,volser,balance of extent information
// LIBDEF CL,TO=PVTCL
// DLBL M204LIB,'M204.PROD.LIBRARY'
// EXTENT ,SYSnnn,volser,balance of extent information
// LIBDEF PHASE.SEARCH=(PRIVLIB.xxxx,M204LIB.V220)
// DLBL CCAJRNL,'MODEL204.CCAJRNL'
// EXTENT SYS001,balance of extent information
// DLBL CCATEMP,'MODEL204.CCATEMP',,DA
// EXTENT SYS001
// DLBL CCASTAT,'MODEL204.CCASTAT'
// EXTENT SYS001,balance of extent information
// ASSGN SYS001,DISK,VOL=SYSWK1,SHR
// UPSI 10111000
// EXEC IFAMTEST,SIZE=AUTO
/*
/&
```

Using the LIBDEF statement

In VSE, a LIBDEF JCL card must point to the library and sublibrary that contain the Model 204 IFAM1 interface program and the HLI user's application program. Use the name of the Model 204 load sublibrary that is used to run IFAM1 programs at your site.

If the Model 204 load module library and application load library are in separate libraries, the LIBDEF must specify both the library and the sublibrary in the search chain.

For more information

See page 31 for more information about the job control statements that are used with an IFAM1 job. Note that VSE allows a seven-character file name to be used for DLBL in the JCL.

IFAM1 jobs: Link-editing under CMS

Execute with dynamic loading

In CMS, the process of generating an application module that uses IFAM1 can be done at runtime. No additional facility is required to make the process dynamic.

Figure 3-5 shows an example of the CMS commands that may be used to run an IFAM1 application. This set of commands loads the application module and executes the program. In this example, the program name is IFM1PRGM.

Figure 3-5. Example of CMS link-edit and execution (IFAM1)

```
&CONTROL OFF
* EXAMPLE TO GENERATE AN APPLICATION MODULE THAT USES
* IFAM1. PROGRAM NAME = IFM1PRGM

GLOBAL TXTLIB M204IFM1

LOAD IFM1PRGM IFIF1 (RESET IFM1PRGM)

GENMOD IFM1PRGM
&TYPE
&TYPE IFM1PRGM MODULE A HAS BEEN GENERATED FOR YOUR USE.
&TYPE
&EXIT 0

IFAM1 FILES
```

Using the IFAM1 command

Figure 3-5 uses the IFAM1 command to execute a host language application that uses IFAM1.

The IFAM1 command in Figure 3-5 specifies the name of an EXEC (FILES) that defines system and database files for IFAM1 and stacks the name of the application program to be executed. See Figure 3-6 for an example of the FILES EXEC that is used.

Note that you can use the IFAM1 EXEC command to run your IFAM1 program assuming that CMS Model 204 has been correctly installed and configured at your site. For IFAM1 under CMS, the IFAM1 saved segments must have been generated and the appropriate minidisks that contain Model 204 EXECs and MODULEs must be accessible.

Note: A return code of 0 from the EXEC invokes the application program. A return code of 1 bypasses the invocation of the program. Any other return code is considered an error condition and ends the response to the IFAM1 EXEC command.

Example of the FILES EXEC

Figure 3-6 below shows FILES EXEC, the EXEC named in the IFAM1 command in Figure 3-5 on the previous page, which might contain the following statements (shown in REXX):

Figure 3-6. Example of the FILES EXEC

```
'FILEDEF * CLEAR'
```

```
'FILEDEF CCATEMPDISK CCATEMP M204SYS A'
'FILEDEF CCAJRNLDISK CCAJRNL M204SYS A'
'FILEDEF CCAGRPDISK CCAGRP M204SYS A'
'FILEDEF CCASTATDISK CCASTAT M204SYS A'
'FILEDEF CCASNAPDISK CCASNAP M204SYS A'
'FILEDEF CCAPRINTDISK CCAPRINT M204SYS A'
'FILEDEF CCAAUDITDISK CCAAUDIT M204SYS A'
'FILEDEF CHKPOINTDISK CHKPOINT M204SYS A'

/**/
/*! include additional FILEDEF commands for database files */
/**/
push 'IFM1PGRM'
exit 0
```

For more information

See page 31 for more information about the job control statements that are used with an IFAM1 job.

IFAM1 jobs: Job control statements

The control statements in Table 3-1 are used in an IFAM1 job.

Table 3-1. IFAM1 control statements

Statement	Use
EXEC	required
STEPLIB/LIBDEF/TXTLIB	required
CCAJRNL	optional (used for audit information only)
CCAJLOG	optional (used for audit information)
CCAAUDIT	optional
CHKPOINT	required for roll-back recovery
CCATEMP	required
CCASNAP, SYSUDUMP	required for error diagnostics
CCAPRINT	required
CCASTAT	required for security
CCAGRP	required for permanent file groups
Model 204 files	required
Application program files	optional

Note: VSE allows a seven-character DLBL name while z/OS allows an eight-character DD name. The actual names that are used in VSE JCL may differ in this regard from those that are listed above.

General information about the EXEC, Model 204 file, and application program file statements for IFAM1 jobs is provided on the next page.

For a detailed description of the Model 204 data sets that are used for HLI jobs, refer to the *Rocket Model 204 Host Language Interface Programming Guide*. For additional information about the Model 204 job control statements, refer to the Rocket Model 204 documentation wiki:

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

EXEC statement

For IFAM1, the EXEC statement does not refer to Model 204 directly. When an IFAM1 job is running, Model 204 parameters cannot be specified in the EXEC statement, that is, not in the PARM field for z/OS, and not in the OPTION SYSPARM statement for VSE.

In IFAM1, any Model 204 parameter settings must be passed to Model 204 inside the application program using HLI calls. See page 32 for more information about using IFSTRT and IFDIAL threads in IFAM1.

Model 204 files

The control statements must contain a data set or file definition statement for each data set of each Model 204 file to be used in the run.

A disposition of either SHR or OLD can be specified, depending whether the Host Language Interface job runs concurrently with other Model 204 programs which use the file. Usually SHR is the recommended disposition. Model 204 uses its own system of enqueuing to resolve updating conflicts.

Application program files

The application program may need data set or file definition statements of its own for various data sets. Be careful not to specify file names that are the same as the names of any of the data set or file definition statements needed by the IFAM1 Host Language Interface itself.

IFAM1 jobs: Using an IFSTRT or IFDIAL thread

Using a single thread

IFAM1 is a single-region configuration of Model 204 that supports a single user using a private copy of Model 204. IFAM1 supports a single thread, either an IFSTRT thread or an IFDIAL thread.

The specific Model 204 interface modules that are used by the Host Language Interface for the IFAM1 job run are automatically invoked by Model 204 and

depend on whether you are using an IFSTRT or an IFDIAL thread in your application program.

Specifying Model 204 runtime parameters

In IFAM1 the EXEC parameter which is specified in the job control EXEC statement is not available to Model 204.

The IFAM1 application programmer can optionally specify Model 204 system and User 0 runtime parameters inside the application program using either the IFSTRT call, if the job starts an IFSTRT thread, or the IFSETUP call, if the job starts an IFDIAL thread.

Note that there are other differences in coding calls when using either an IFSTRT or an IFDIAL thread. See Chapter 6 for detailed descriptions of the IFSTRT (IFAM1) and IFSETUP calls. Refer to Appendix A to see sample IFAM1 applications.

IFAM2 jobs

IFAM2 jobs can run under z/OS, VSE, or CMS.

Except for IFDIAL under CMS, an IFAM2 job can establish multiple threads. Under z/OS and VSE, an IFAM2 job can start one or more IFSTRT threads and one IFDIAL thread. Under CMS, a IFAM2 job can start either an IFSTRT thread (or threads) or an IFDIAL thread.

The Model 204 service program operates in its own region or virtual machine. The region processes requests from an arbitrary number of host language programs, each of which operates in its own region or virtual machine. IFAM2 requires the use of communications facilities.

Communications facilities (CRAM or IUCV)

For z/OS and VSE, communication between Model 204 and a host language program in a different region is enabled by the Model 204 Cross-Region Access Method (CRAM), a special inter-region communications facility.

For CMS, a host language program communicates with Model 204 in a separate virtual machine through the IBM inter-user communication vehicle (IUCV).

See the next section for specific information about the subsystem name, channel names and IODEV settings that are used for IFAM2 jobs running under z/OS, VSE, and CMS.

Subsystem names, channel names, and IODEV settings

Model 204 uses the following subsystem names, channel names, and IODEV settings for IFAM2 threads:

- For IFSTRT threads Model 204 uses the default channel name supplied by Rocket.
- For IFSTRTN threads Model 204 uses the channel name that is specified in the HLI call.
- For IFSTRTN threads and z/OS with XDM, Model 204 uses the default subsystem name found in the IGCLM244 load module or the subsystem name found in the HLI as well as the channel name in the HLI call.

  The following default channel names and IODEV settings are used for IFSTRT threads:

Facility	Parameter	Default subsystem name	Default channel name	IODEV setting
CRAM	IFAMCHNL	Value in the IGCLM244 load module	IFAMPROD	23
IUCV	VMIFCHNL	Not Applicable	M204VMIF	43

- For IFDIAL threads Model 204 uses the default channel name supplied by Rocket.
- For IFDIALN threads Model 204 uses the channel name that is specified in the HLI call.
- For IFDIALN threads and z/OS with XDM, Model 204 uses the default subsystem name found in the IGCLM244 load module or the subsystem name found in the HLI as well as the channel name in the HLI call.

  The following default channel names and IODEV settings are used for IFDIAL threads:

Facility	Parameter	Default subsystem name	Default channel name	IODEV setting
CRAM	CRIOCHNL	Value in the IGCLM244 load module	M204PROD	29
IUCV	VMIOCHNL	Not Applicable	M204VMIO	39

Note that the IODEV settings listed above are the ones that are used by Model 204 for IFAM2 processing. You can specify VMIOCHNL and VMIFCHNL (and the corresponding IODEVs) only for CMS; you can specify CRIOCHNL and IFAMCHNL (and the corresponding IODEVs) only for z/OS and VSE.

These threads must be defined in the Model 204 Online run to provide IFAM2 communications. One IODEV is required by Model 204 for each thread that is started in the HLI program.

Compiling under Enterprise PL/I for z/OS

When compiling a PL/I application under the Enterprise PL/I for z/OS compiler, the following compiler parameter is required:

```
DEFAULT(LINKAGE(SYSTEM))
```

This causes the parameter list to be built in the same way that it was built by the old compilers (including turning on the high-order bit of the address of the last parameter).

For example:

```
//PLICMPL EXEC PGM=IBMZPLI,PARM='OBJECT,OPTIONS,
//             DEFAULT(LINKAGE(SYSTEM))',REGION=512K,...
```

If this compiler option is not specified, subsequent executions of the application will fail with 0C4 abends.

IFAM2 jobs: Running under z/OS and VSE

Using the CRAM facility

CRAM is an interregion facility that allows two or more programs to communicate with each other over an arbitrary number of distinct connections, called channels.

In any communication handled by CRAM, one program is the master and the other is a user. The master is always the Model 204 service program, which can communicate with several users over the same channel. After a connection is established, the user thread can issue requests, as required, for the application.

Several versions of Model 204 can run concurrently by establishing a distinct CRAM channel name for each version. Also, one user program can simultaneously communicate with multiple Model 204 service programs in different regions. For more information about CRAM, refer to the Rocket Model 204 documentation wiki:

http://m204wiki.rocketsoftware.com/index.php/Defining_the_User_Environment_(CCAIN)#CRAM_.28IODEV.3D11.2C_23.2C_29.29

Linking with the IFIF module

A batch IFAM2 program running under z/OS and VSE must be linked with the IFIF object module, which condenses the data pertinent to each Host Language Interface call, passes it to CRAM, and sends the information to an interface routine in the Model 204 region.

Model 204 services the call and returns the information to the appropriate host language program that uses CRAM as the intermediary.

Link-editing and executing the application program

The IFAM2 application program requires its own job control statements to execute.

Note that the IFAM2 job control statements are independent of the Model 204 job because the IFAM2 application program executes as a separate job in its own region. The control statements for Model 204 files are included in the Model 204 run.

IFAM2 programs must be link-edited with the NODYNAM option set. If NODYNAM is not set, your program may abend when executed.

Figure 3-7 is a sample job excerpt showing the z/OS JCL which runs an IFAM2 application program. In this example, COBUCLG is a COBOL compile-link-and-go procedure.

Figure 3-7. Example of z/OS JCL to run application (IFAM2)

```
//CPLLKGO EXEC COBUCLG,
//PARM.COB='LOAD,NOSEQ,NODYNAM,APOST',
//REGION.LKED=200K,
//PARM.LKED='LIST,LET,SIZE=(192K,100K),MAP',REGION,GO=64K
//COB.SYSIN DD *
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CRAMDIAL.
•
•
•
 STOP RUN.
//LKED.OBJLIBDD DSN=M204.V220.OBJLIB,DISP=SHR
//LKED.SYSINDD *
INCLUDE OBJLIB(IFIF)
//GO.SYSUDUMPDD SYSOUT=A
//GO.SYSOUTDD SYSOUT=A
//GO.INFILEDD *
•
•
•
/*
//
```

To link-edit your IFAM2 job running under z/OS, link the IFAM2 application with the Model 204 module IFIF using the INCLUDE OB statement in SYSIN DD.

IFAM2 jobs: Running under CMS

Using the M204IFAM object modules

The IBM inter-user communication vehicle (iUCV) allows HLI programs running under CMS to communicate with a Model 204 system running in a separate virtual machine.

The application program must be loaded to include a library of object modules, named M204IFAM TXTLIB, that contains the CMIF, IFIF, IFCM, and IUCV interface routines that are necessary for this communication.

Using the M204IFAM EXEC

The CMIF routine in M204IFAM TXTLIB invokes a procedure called M204IFAM EXEC to obtain the name of the virtual machine that is running Model 204 and to obtain the needed IUCV communications method.

Both M204IFAM TXTLIB and M204IFAM EXEC must be available on the CMS machine for an IFAM2 host language program.

You can use any of the following values in the line that indicates the communication type to the M204IFAM EXEC:

- IUCVVMCF (the default)
- IUCV
- VMCF
- VMCFIUCV

Note: Regardless of the option specified, Model 204 establishes communications with IUCV. The VMCF facility is not supported as of Version 2, Release 2 of Model 204. To ensure compatibility with existing programs, the VMCF option is still accepted but has no effect. The VMCF option may be dropped in future releases of Model 204. If this occurs, you will need to change your HLI programs to eliminate all references to VMCF.

See Appendix A and Appendix B for examples of a CMS M204IFAM EXEC, which establishes the IUCV link.

Using VM immediate commands

You can use the HX, HT, and RT VM immediate commands described below when an IFAM2 host program is executing.

If one of the following commands is entered after the attention key for the terminal is pressed, VM takes the immediate actions summarized below:

- The HX command abnormally terminates (ABENDs) execution of a program. Control returns to CMS. This is equivalent to cancelling a program under z/OS.
- The HT command halts the display of output on the terminal, but continues execution. Use the RT command to resume output.

Loading and executing an IFAM2 program

To load and execute an IFAM2 application program under CMS, use the following steps as guidelines. The specific procedures for loading and running IFAM2 programs depend upon the particular language in which the program is written.

1. Use the GLOBAL TXTLIB command to specify the list of object libraries (called TXTLIBs) that are used to resolve references by the IFAM2 program to library subroutines.

   Note that the GLOBAL TXTLIB command includes M204IFAM TXTLIB in the list of libraries, which enables the CMS Loader to include the IFAM2 interface in the program. Specify the following command in the GLOBAL TXTLIB command:

   ```
   GLOBAL TXTLIB M204IFAM
   ```

   M204IFAM TXTLIB contains the interface routines necessary for communication with Model 204 using IUCV.

2. Load the application program using the CMS Loader.

   Note that the LOAD command that is normally used to invoke the CMS Loader must specify the Model 204 IFCM module in the list of programs to be loaded. If this is not done, IFCM is not included and an error occurs when the application program attempts to load the module named IGCLM244.

   To load a program named IFAMTEST, for example, specify the following command:

   ```
   LOAD IFAMTEST IFCM CLEAR
   ```

3. To run the program immediately, specify START in the command line as follows:

   ```
   LOAD IFAMTEST IFCM (START CLEAR
   ```

   or specify START as a separate command:

   ```
   LOAD IFAMTEST IFCM CLEAR
   START
   ```

   Or, use the GENMOD command to create a nonrelocatable program module (with external references resolved).

   For example, for the program named IFAMTEST, specify the following command:

   ```
   LOAD IFAMTEST IFCM
   GENMOD IFAMTEST
   ```

   IFAMTEST is the file name assigned to the module (and the file type is MODULE).

   And next, enter the name of the program to run it. For example,

```
IFAMTEST
```

For more information

Refer to Appendix B for examples of CMS EXECs to compile and link and to run an IFAM2 application.

For more information about running application programs under CMS, refer to the appropriate IBM User's Guide for your operating system.

IFAM2 jobs: Using an IFDIAL thread under CMS

Specifying the IFDIAL operand in the ONLINE command

Under CMS, an IFAM2 job can start an IFDIAL thread, and this is the only connection that is allowed in the job.

Use the IFDIAL operand in the ONLINE command to specify a single-user IFDIAL connection to be made (saved segment mandatory).

The IFDIAL connection must be made on the main (nonrecovery) step. Execution of the ONLINE command creates a Model 204 Online environment. For more information about this facility, refer to the Rocket Model 204 installation instructions for the IBM z/VM platform.

If you specify IFDIAL, the main (nonrecovery) EXEC must provide only one parameter, the user program name, in the stack. IFDIAL connections require that the user program be placed first on the stack.

When defining subsystems that contain Model 204 calls in the SCT for IFDIAL, do not set the DBASE parameter.

IFAM2 jobs: Running under CICS

IFAM2 pseudo conversational support allows CICS users to include pseudo conversational applications in their IFAM2 programs. This allows the host IFAM2 program to enter a pseudo conversational wait at any time during an IFAM2 session with Model 204. The CICS interfaces are compatible with CICS Release 2.1 and later.

Transaction management

The CICS pseudo conversational programming technique ensures that no resources are held by a task during each conversational iteration with the terminal operator.

Such resources include VSAM strings, file record enqueues, enqueues on Temporary Storage queues or transient data destinations, as well as the storage acquired by the transaction during execution. Note, however, that Model 204 found sets are retained.

The pseudo conversational feature prevents potentially serious bottlenecks between transactions that could cause longer response times and deadlocks.

Using the CICS-resident program (DFHPSF)

For applications running under CICS, the IFIF module along with a CICS-specific appendage called IFPS, are linked as a CICS-resident program, such as DFHPSF, when the CICS interface is installed.

Once the CICS interface is installed using the CICS-resident program, CICS users can submit IFAM2 jobs to run by link-editing to IFENTPS, which is described in the next section.

Note: The CICS-resident program name is site-specific and can be named on the CICFG configuration section of the CICS batch program. Several types of DFHPSF (IFIF/IFPS) CICS-resident programs can be available in the CICS environment. The particular copy can be selected through the configuration name.

For more information about CICS-resident programs, refer to the *Rocket Model 204 Host Language Interface Programming Guide*.

Linking to the IFENTPS module

The IFENTPS module provides the IFAM2 application program interface for pseudo conversational CICS transactions and must be link-edited with the user CICS program.

To use the IFAM2 CICS interface, link-edit your program with the IFENTPS module. No other direct changes to your program are required.

Include the data set or file definition statements for Model 204 files that are accessed by CICS applications in the JCL for the Model 204 region, not the CICS region.

Note: The IFENTPS module may be used with either macro-level or command-level CICS application programs. The configuration copy member, CICFG, contains a conditional assemble switch that determines whether a macro-level or command-level version is generated. The Model 204 system manager must ensure that the correct version of IFENTPS is used with the appropriate application programs.

Using macro or command level program code

The version of IFENTPS that is link-edited with your IFAM2 CICS program must be assembled specifically for either macro or command level program code.

Addressing and storage requirements

Note the following IFAM2 CICS requirements:

- The following CICS areas must be addressable: CSA, TWA, and TCA.
- IFAM2 requires 88 bytes of CICS TWA.
- A temporary storage queue is created for each HLI transaction when the first IFSTRT or IFDIAL call is issued.

First and last calls in the program

The following calls are required in the IFAM2 CICS program:

- Before Version 3.1 of Model 204, the IFCSA call was required and needed to be the first call issued during the IFAM2 session. For Model 204 Version 3.1 and later, the IFCSA call no longer performs any function and is not required. For purposes of upward compatibility, the IFCSA call can remain in your current programs.
- The IFFNSH or IFHNGUP call is necessary as the last call in the program to ensure that all CRAM resources are released at program termination.

Abend handling is required

Using the Model 204 IFAM2 CICS interface successfully depends on correct abend handling to prevent hung CRAM channels.

A program must establish its own abend handler or use the IFABXIT call to establish the IFAM2 abend handler. The program must establish the abend handler or issue IFABXIT after any pseudo conversational waits, that is, after the initial connection is established.

If you run CICS IFAM2 applications written in COBOL II, set the &IFABEND parameter in CICFG to ‘NO’. This prevents the abend handler in IFENTPS from causing ASRAs.

If you do this, you may want to include an abend handler routine in your COBOL II applications to prevent hanging threads after an abend. If you do, the abend handler routine must issue an IFFNSH call.

For more information

Refer to the *Rocket Model 204 Host Language Interface Programming Guide* for detailed information about abend handling in an IFAM2 CICS program. Also refer to that guide for the following information about IFAM2 CICS programs:

- The IFENTPS link module
- Addressable CICS areas (CSA, TCA, TWA) and coding example
- TWA requirements, using the CICFG copy member
- Temporary storage queue

Refer to Chapter 6 for a description of the IFCSA, IFFNSH, and IFABXIT calls.

IFAM4 jobs

An IFAM4 job involves a two-step procedure: step one is to link-edit the application program and step two is to execute the IFAM4 batch job which runs Model 204 and the application program.

IFAM4 applications can run under the z/OS operating system. Examples of the IFAM4 job steps showing sample setups are provided in the sections which follow. The following topics are covered:

- Link-editing the application program
- Running the application program

See “IFAM4 jobs: Control statements” on page 45 for a description of the control statements that are used with an IFAM4 job.

Running the IFAM4 application and Model 204

IFAM4 is a single-region configuration of Model 204 that supports a single user using a private copy of Model 204.

When you use the IFAM4 interface, your application program and the Model 204 service routines run as separate tasks in a single region or virtual machine.

Note: In order to use the IFAM4 facility, the Model 204 system manager must initially build the IFAM4 load module for your installation site. IFAM4 must be link-edited with the REUS option and must be named IFAM4.

Contact your Model 204 system manager to get the name of the Model 204 load library that is used to run IFAM4 programs at your site. For more information about creating the IFAM4 module, refer to the Rocket Model 204 installation instructions for your operating system.

Using IFSTRT threads

IFAM4 supports an application that uses one or more threads. An IFAM4 application can use only IFSTRT threads.

If your application uses single cursor IFSTRT threads, IFAM4 provides the most efficient way to run a multithreaded batch HLI program. IFAM4 is also useful in certain test and batch production situations.

Running concurrent IFAM4 applications

Ordinarily, two concurrently running HLI programs do not use IFAM4 to interface with a single copy of Model 204. Each IFAM4 application program runs in its own region and must have its own copy of Model 204.

However, the capability to use multiple applications or copies of an application with a single copy of Model 204 exists. To run multiple applications or copies, the host installation must provide a single controlling or monitor program to attach as subtasks each application program that is to communicate with Model 204.

This monitor program is then considered to be the application referred to subsequently in the IFAM4 control statements. Each of these application programs must have its own copy of IFIF4 link-edited to do the Host Language Interface calls and data transfers.

IFAM1 jobs: Compiling under Enterprise PL/I for z/OS

When compiling a PL/I application under the Enterprise PL/I for z/OS compiler, the following compiler parameter is required:

```
DEFAULT(LINKAGE(SYSTEM))
```

This causes the parameter list to be built in the same way that it was built by the old compilers (including turning on the high-order bit of the address of the last parameter).

For example:

```
//PLICMPL EXEC PGM=IBMZPLI,PARM='OBJECT,OPTIONS,
//             DEFAULT(LINKAGE(SYSTEM))',REGION=512K,...
```

If this compiler option is not specified, subsequent executions of the application will fail with 0C4 abends.

IFAM4 jobs: Link-editing under z/OS

Link with IFIF4

To link-edit your IFAM4 job running under z/OS, link the IFAM4 application with the Model 204 object module IFIF4 using the INCLUDE OB statement in SYSIN DD.

Figure 3-8 shows an example of the JCL that may be used to link-edit an IFAM4 application to build the application load module. In this example, COBUCL is a COBOL compile-and-link procedure and the program name is COBOLTST.

Figure 3-8. Example of z/OS JCL to link-edit COBOL application (IFAM4)

```
//EXECCOBUCL
•
•
•
//LKED.SYSLMODDDDISP=(NEW,CATLG),
//DSNAME=LOCAL.M204.IFAM4.APPLIC
```

```
//LKED.OBDDDSN=LOCAL.M204.OBJECT,DISP=SHR
//LKED.SYSINDD*
INCLUDE OB(IFIF4)
NAME COBOLTST(R)
```

Figure 3-9 below shows another example of the JCL that may be used to link-edit an IFAM4 application to build the application load module. In this example, PL1LFCL is a PL/1 compile-and-link procedure and the program name is PL1TEST.

Figure 3-9. Example of z/OS JCL to link-edit PL/1 application (IFAM4)

```
//EXECPL1LFCL
•
•
•
//LKED.SYSLMODDDDISP=(NEW,CATLG),
//DSNAME=LOCAL.M204.IFAM4.APPLIC
//LKED.OBDDDSN=LOCAL.M204.OBJECT,DISP=SHR
//LKED.SYSINDD*
INCLUDE OB(IFIF4)
NAME PL1TEST(R)
```

IFAM4 jobs: Running under z/OS

Execute IFAM4

Once you have link-edited the application program, run the IFAM4 application in z/OS.

Figure 3-10 shows a sample excerpt of the JCL that may be used to run an IFAM4 application which was link-edited as shown in Figure 3-9. The sample job stream runs Model 204 and an IFAM4 application program (executing IFAM4) with three IFSTRT threads (IODEV=23). In this example, the program name in IFAM4IN DD is PL1TEST.

Figure 3-10. Example of z/OS JCL to run application (IFAM4)

```
//RUN  EXEC  PGM=IFAM4,REGION=1024K,TIME=10,
//           PARM='SYSOPT=160,LIBUFF=1024'
//STEPLIB    DD  DSN=LOCAL.M204.IFAM4.APPLIC,DISP=SHR
//           DD  DSN=LOCAL.M204.LOAD,DISP=SHR
//CCAAUDIT   DD  SYSOUT=A
//CCAJRNL    DD  DSN=M204.JRNL,DISP=OLD
//CCATEMP    DD  UNIT=3380,SPACE=(TRK,40),
//           DISP=(NEW,DELETE)
//CCASNAP    DD  SYSOUT=A
//SYSUDUMP   DD  SYSOUT=A
//VEHICLES   DD  DSN=M204.FILE.VEHICLES,DISP=SHR
//CUSTOMER   DD  DSN=M204.FILE.CUSTOMER,DISP=SHR
//EMPLOYEE   DSN=M204.FILE.EMPLOYEE,DISP=SHR
//CCAPRINT   DD  SYSOUT=A
//CCASERVR   DD  UNIT=3380,DISP=(NEW,DELETE),
```

```
//            SPACE=(CYL,2)
//CCAIN       DD  *
NUSERS=4,NSERVS=3,NFILES=3,NDIR=3
IODEV=23
IODEV=23
IODEV=23
*SLEEP 3600
/*
//IFAM4IN     DD  *
PL1TEST
P1=149,P2=ANDREW
/*
//SYSOUT      DD  SYSOUT=A
//INFILE      DD  DSN=RANDOM.INPUT.DATA,DISP=SHR
•
•
•
```

See page 48 for information about IFAM4 job step return codes and job run ABENDs. Note that the IFAM4 load module must be installed at your site and must be named IFAM4.

IFAM4 jobs: Control statements

Overview of IFAM4 job control statements

The control statements listed in Table 3-2 are used in an IFAM4 job.

Table 3-2. IFAM4 control statements

Statement	Use
EXEC	Required
STEPLIB	Required
CCAJRNL	Required for roll-forward recovery
CHKPOINT	Required for roll-back recovery
CCAAUDIT	Optional
CCATEMP	Required
CCASNAP, SYSUDUMP	Required for error diagnostics
CCAPRINT	Required
CCAIN	Required
CCASERVR	Required for server swapping
CCASTAT	Required for security
CCAGRP	Required for permanent file groups
IFAM4IN	Required

Table 3-2. IFAM4 control statements (Continued)

Statement	Use
Model 204 files	Required
Application program files	Optional

Information about the EXEC, STEPLIB file, CCAIN, IFAM4IN, Model 204 files, and application program file statements for IFAM4 jobs starts on the next page.

For a detailed description of the Model 204 data sets that are used for HLI jobs, refer to the *Rocket Model 204 Host Language Interface Programming Guide*. For additional information about the Model 204 job control statements, refer to the Rocket Model 204 documentation wiki:

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

EXEC statement

You can specify the following parameters in the EXEC statement:

- PGM — Required; Must specify IFAM4 (and not the name of the application program).
- REGION — Required; Indicates the size of the memory area to be allocated for both IFAM4 and the application. Refer to the Rocket Model 204 documentation wiki for information about setting the REGION parameter:

  http://m204wiki.rocketsoftware.com/index.php/Defining_the_runtime_environment_(CCAIN)#Runtime_environment_specifications
- TIME — Required; Indicates how much time the application program and Model 204 together can use before being cancelled by the operating system.
- PARM — Optional; Indicates any Model 204 runtime parameters, to be passed to Model 204. Note that user application parameters are not specified in the EXEC statement for IFAM4 and are handled instead by IFAM4IN, which is described below.

  Specify any Model 204 parameters that can be set in the EXEC statement using the following format:

  ```
  ,PARM='parameter=value[,parameter=value]•••'
  ```

  Note that the following guidelines apply for specifying SYSOPT parameter settings:

  – If option 16 is set, a valid LOGIN account must be specified in the IFSTRT call for each application thread that is started by the job.

  – Setting option 32 (print RK lines) provides an audit trail or journal log of each Host Language Interface call made to Model 204. This log is

invaluable for debugging the application.

STEPLIB statement

The STEPLIB file points to the load module libraries where the application program and the Model 204 IFAM4 interface program reside.

The STEPLIB file statement must specify the Model 204 load module library or the library that contains IFAM4 and the user's application program. If the Model 204 load module library and application load library are in separate libraries, use concatenated DD data set definition statements.

CCAIN file

The CCAIN input file is required for IFAM4 to specify Model 204 User 0 parameters that apply for the rest of the job run. For IFAM4 the following parameters and statements are required in the CCAIN DD:

- NUSERS — Required; NUSERS is required on the User 0 parameter line. Set NUSERS equal to the number of threads to be used (that is, the number of IODEV statements) plus one.
- IODEV=23 — Required; Following the User 0 parameter lines, insert an IODEV=23 statement for each thread to be started in the application program.
- *SLEEP — Required; You must use the *SLEEP command following the IODEV lines to specify an elapsed time in seconds longer than the time the application program needs to finish processing. *SLEEP suspends processing of the CCAIN input stream but allows the application program threads to continue.

  Note that IFAM4 immediately terminates when the application program ends. If *SLEEP times out before the application program completes, IFAM4 ends and, if the application is updating the database, file damage may occur.

  Note: Model 204 expects the CCAIN data set to consist of 80-character lines. Data is in columns 1 through 71. Any nonblank character in column 72 indicates continuation of the line in to the next line. Refer to the Rocket Model 204 documentation wiki for a complete description of the CCAIN file:

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

IFAM4IN file

Use the IFAM4IN file definition statement to specify a data set that contains the following two control statements:

1. The first statement is required and contains the name of the your application program as specified in the compile-and-link job. It must start in

Column 1.

2. The second statement is used to specify parameters to be passed to the application program, as necessary.

   Note: If you are running the application program named APATTACH, omit the entire IFAM4IN data set.

Model 204 files

The control statements must contain a data set or file definition statement for each data set of each Model 204 file to be used in the run.

A disposition of either SHR or OLD can be specified, depending whether the Host Language Interface job runs concurrently with other Model 204 programs which use the file. Usually SHR is the recommended disposition. Model 204 makes use of operating system enqueuing to resolve updating conflicts.

Application program files

The application program may need data set or file definition statements of its own for various data sets. Be careful not to specify file names that are the same as the names of any of the data set or file definition statements needed by the Host Language Interface itself.

IFAM4 jobs: Job errors and ABENDs

Follow the guidelines in Table 3-3 below for troubleshooting IFAM4 job ABEND codes.

Table 3-3. IFAM4 job ABEND codes

Code	Solution
S806	Check that the following items are correctly specified: 1. Verify that IFAM4 is named IFAM4 in the link-edit. 2. Verify that the STEPLIB DD statement points to the library or libraries containing IFAM4 and the application program. 3. Verify that the application program is correctly specified in Column 1 of the first statement of IFAM4IN.

Table 3-3. IFAM4 job ABEND codes (Continued)

Code	Solution
S804 or S80A	Verify that the REUS option was specified in the link-edit of IFAM4. The following conditions indicate that the REUS option was not specified in the link-edit of IFAM4: • IFSTRT returns a completion code of 1001. • A message appears that monitor requested IFAM4 termination. • IBM messages indicate that there is not enough storage for GETMAINS. Follow either step one or two: 1. If the REUS option was not specified, relink IFAM4. 2. If the REUS option was specified, increase REGION by 10K and add 3000 to the SPCORE parameter. **Note:** The SPCORE parameter defaults to 12288 in IFAM4 and can be set in the PARM field of the EXEC statement or on User 0's parameter line.

Job step return codes

IFAM4 sets the return code for the job step according to the highest completion code from Model 204 and the application program.

IFAM4 returns the job step return codes for the error conditions listed below.

Code	Error condition
999	Model 204 is not initialized properly.
998	Either the application or Model 204 ABENDs.

Note that option 64 of the SYSOPT parameter can be set to force an ABEND without a dump at termination when the return code is nonzero. The SYSOPT parameter is specified in the EXEC statement, which is described on page 32.

Part II
HLI Functions

Part II describes in detail each of the calls that is available using the Model 204 Host Language Interface facility. Use this information to code the calls in your HLI application program.

4

HLI Coding Conventions

Overview

This chapter gives the application programmer guidelines for coding HLI call parameters that apply in general, and conventions to follow when using particular programming languages.

For more information

Refer to Chapter 6 for descriptions of all of the HLI calls and their parameter lists. See Appendix A and Appendix B for examples of HLI programs written in different programming languages.

General coding guidelines

Using the host language call protocol (the CALL verb)

Host Language Interface functions are called by the host language program according to the host language subroutine call protocol. Each HLI function is an external subroutine and is called with a CALL verb of the host language.

For example:

Language	Example using call verb
COBOL	CALL “IFOPEN” USING RET-CODE, FILE-NAME
FORTRAN	CALL IFOPEN(RETCD,FILEN)

Language	Example using call verb
PL/I	CALL IFOPEN (RET_CODE,FILE_NAME)
Assembler	CALL IFOPEN,(RETCD,FILEN),VL

Note: If you are using COBOL, your COBOL compiler determines whether or not you use double quotes.

Using function names and aliases

All Host Language Interface function names begin with the letters IF and are seven characters or less in length, for example, IFFIND and IFCOUNT.

Function names that are seven characters in length also have an alias (six characters or less in length) that is used in FORTRAN programming.

The alias name, if available, is provided in the Syntax description of a call. The full name is first, followed by a vertical bar separator, followed by the alias. For example, IFFINDC | IFFDC, where IFFDC is the alias.

See Chapter 5 for a complete listing of HLI calls by name.

Using function numbers

All Host Language Interface functions except IFCALL, IFLOG, and IFSTRT (in IFAM1) have an associated function number. You can use IFCALL to call functions that have numbers only if you are running an IFAM2 or IFAM4 job.

The function number, where available, is provided in the Syntax description of the individual call, enclosed inside parentheses. See Chapter 6 for a description of the standard convention used to identify function numbers in this document.

See Chapter 5 for a complete listing of HLI calls and their numbers.

Specifying HLI call parameters

The following guidelines apply for specifying HLI call parameters:

- The same set of parameters for each call is required with ***all*** languages.
- The order of the parameters in each call is important. Follow the syntax that is specified for each call.
- Except where noted, you cannot omit a parameter. If you do not want to specify a value for a parameter, supply a null placeholder parameter. If, for example, parameters 3 and 4 are optional and you select one, you include both. Note that a missing parameter is not a null parameter.

Defining HLI call parameter data types

With the exception of one parameter of IFCHKPT that is in packed format, all parameters are defined as one of the following types:

- Fullword aligned binary integers in COBOL
- Fullword integer in FORTRAN
- Fullword fixed binary integer in PL/I
- Short character string whose maximum length is 32 bytes
- Full length character string whose maximum length is the buffer size; the input buffer size is determined by the LIBUFF User 0 parameter; the output buffer size is determined by the LOBUFF User 0 parameter.

  For example:

Language	Example of integer variable	
COBOL	05 INTEGER-ARG	PIC 9(5) COMP SYNC.
FORTRAN	INTEGER*4 INTARG	
PL/I	DECLARE INTEGER_ARG	FIXED BIN(31):
Assembler	INTARG DC F’O’	

Length restrictions on character string parameters in HLI calls

Character string parameters are passed to Model 204, but Model 204 does not always know the length. For example, length is known with PL/I dope vectors, but length is not known with COBOL. Length is language-dependent.

The mechanism that governs the maximum length that an HLI call may pass in a single parameter is determined by the setting of the language indicator that is specified (in the IFSTRT, IFSTRTN, IFDIAL, or IFDIALN call) in the HLI application.

For a language indicator equal to 2 (that is, COBOL, FORTRAN, and Assembler), the Model 204 User 0 parameter LIBUFF determines the maximum length of a character string that may be passed by the HLI program in an individual parameter; the Model 204 User 0 parameter LOBUFF determines the maximum character string length to return.

For a language indicator equal to 1 or 3, that is, PL/I, the length of each argument is supplied as part of the parameter. PL/I passes dope vectors in parameter lists which contain the exact length of the parameters.

In the examples of character string variable definitions below, the actual length of the input parameter is specified for COBOL and FORTRAN, and a maximum length of 255 is specified for PL/I.

For example:

Language	Example of character string variable	
COBOL	05 STRING-ARG	PIC X(6) VALUE ‘value;’
FORTRAN	LOGICAL*1 STRING(5)/‘value’/	
PL/I	DECLARE STRING_ARG	CHAR(255) VAR INIT (‘value’)
Assembler	STRING DC C‘value’	

Note: Except where noted, it is important to indicate the end of a character string passed as an parameter with a semicolon (;).

Refer to the *Rocket Model 204 Host Language Interface Programming Guide* for more information about character string variables used in HLI call parameters.

Using the completion return code (RETCODE)

Except for IFABXIT, IFCALL, and IFCSA, the first parameter of each call is a completion return code (RETCODE). In some cases RETCODE is the only parameter that is specified for a call.

Your application program must check return codes because they indicate success or failure or informative messages. Note that nonzero codes do not always mean failure and vice versa.

Completion codes corresponding to particular error conditions are described in the “Notes and Tips” section for individual calls to aid you in coding.

Refer to Chapter 8 for a list of completion return codes. Refer to the *Rocket Model 204 Host Language Interface Programming Guide* for more information about using completion return codes.

COBOL coding guidelines

Use the following coding conventions when COBOL is the host language:

- Call Host Language Interface functions by using a CALL in the PROCEDURE DIVISION. For example:

  ```
  CALL ’IFOPEN’ USING ERROR FILE-NAME.
  ```

- Declare all parameters in the WORKING STORAGE or LINKAGE SECTION of the DATA DIVISION.
  - You must define integer parameters as fullword, aligned, binary numbers. For example:

    ```
    1 ERROR PIC 9(5) COMP SYNC.
    ```

  - Define character parameters as alphanumeric. You must end input of a

parameter string with a semicolon (;). For example:

```
FILE-NAME PIC X(14) VALUE 'TESTFILE;PASS;'.
```

- Model 204 ignores extra embedded blanks within input parameters.
- The maximum character string length that can be passed to a Host Language Interface function in a single parameter is determined by the value of the LIBUFF parameter, which defaults to 255.

  Note that LIBUFF is a Model 204 User 0 parameter that cannot be changed once the Host Language Interface/Model 204 service program is initialized.
- The maximum character string length that a Host Language Interface function returns to the application program is determined by LOBUFF.

  Note that LOBUFF is a Model 204 User 0 parameter that cannot be changed once the Host Language Interface/Model 204 service program is initialized.
- The Host Language Interface function cannot determine the actual length of a COBOL data area parameter.

  Model 204 moves values into the data area, beginning with the first character in the area and continuing until the function has finished processing or until the maximum of LIBUFF character positions has been filled.

  Note: If the data area is longer than the returned values, the remaining data area characters are not filled. If the data area is shorter than the returned values, the characters following the data area are overwritten.
- Some character string parameters are specially designated as short strings. Short strings can be used for input or output.

  The maximum length of a short string in IFAM2 is 32 bytes. In IFAM1 or IFAM4, the maximum length is set by LIBUFF.
- If you run CICS IFAM2 applications written in COBOL II, set the &IFABEND parameter in CICFG to 'NO'. This prevents the abend handler in IFENTPS from causing ASRAs.

  If you do this, you may want to include an abend handler routine in your COBOL II applications to prevent hanging threads after an abend. If you do, the abend handler routine must issue an IFFNSH call.

  Refer to the *Rocket Model 204 Host Language Interface Programming Guide* for detailed information about abend handling in an IFAM2 CICS program.

FORTRAN coding guidelines

Use the following coding conventions when FORTRAN is the host language:

- Call Host Language Interface functions by using the CALL statement. For example:

  ```
  CALL IFOPEN (ERROR, FILENAME)
  ```

- Define integer arguments as fullword integers. For example:

  ```
  INTEGER*4 ERROR
  ```

- End character string arguments with a semicolon. For example:

  ```
  INTEGER*4 FILENAME(4)
  DATA FILENAME/'TEST','FILE',';PAS','S; '/
  ```

- The conventions concerning the lengths of string arguments passed to or returned from Host Language Interface functions are the same as those for COBOL (see page 56).

- For FORTRAN 77, the language indicator is 2 in releases prior to Release 1.3. Add the following @PROCESS command listing all functions:

  ```
  @PROCESS SC (IFSTRT,IFFIND, •••)
  ```

  This compiles the FORTRAN program without dope vectors, which are not supported.

PL/I coding guidelines

Coding conventions

Use the following coding conventions when PL/I is the host language:

- Call Host Language Interface functions by using the CALL statement. For example:

  ```
  CALL IFOPEN (ERROR, FILENAME);
  ```

- Declare all Host Language Interface functions called in a PL/I application program in that program as external entry points.

  Describe integer parameters as fullword, fixed, binary; describe string parameters as character strings. For example:

  ```
  DCL IFOPEN EXT ENTRY (FIXED BIN(31), CHAR(*));
  ```

- Declare integer arguments as fixed, binary, fullword. For example:

  ```
  DCL ERROR FIXED BIN (31);
  ```

- Declare string arguments as character strings, either FIXED or VARYING. For example:

  ```
  DCL FINDSPEC CHAR(8) INITIAL ('STATE=MA');
  DCL OUTDATA CHAR(256) VARYING;
  ```

Note that in PL/I application programs, character string arguments do not need to end with semicolons. All types of PL/I, that is, PL/I F-level (IFSTRT language indicator = 1) or PL/I Optimizer or Checkout (language indicator = 3), pass to Model 204 a dope vector that contains the value length along with the value string itself. A semicolon delimiter at the end of each string argument is not needed.

However, semicolons within argument strings are still required where appropriate. For example:

```
L FILEARGS CHAR(10) INITIAL ('CEN1;PASSW');
```

- The maximum lengths for string arguments that are passed to or returned from Host Language Interface functions are governed by the value of the LIBUFF parameter.
- A string returned from a Host Language Interface function is truncated if the area in the PL/I program to which it is being returned is shorter than the returned value. If the returned value is shorter than the return data area, the unused portion of the data area remains unchanged for FIXED strings, and a string's length is set for VARYING strings.
- The special argument designation of short string does not apply to PL/I.

Passing a channel name as a string in IFSTRTN or IFDIALN

When an IFSTRTN or IFDIALN call is made (in IFAM2) with a language indicator of 1 or 3, Model 204 expects the channel name to be a string and not a PL/I dope vector.

To force the compiler to pass the address of the string instead of the dope vector, you can use a data type that does not use dope vectors, such as FIXED BIN. Either a based or defined variable of this type can overlay the original argument. The variable is passed to the call. A based variable, unlike a defined variable, does not produce any compiler error messages.

The following example passes the address of the string independent of the PL/I compiler in use:

```
DCL
IFSTRTN EXTERNAL ENTRY(FIXED BIN(31),/* completion code */
FIXED BIN(31),/* language ind */
CHAR(*),/* login info */
FIXED BIN(31),/* update indicator */
FIXED BIN(31),/* thread id */
FIXED BIN(31));/* CRAM [subsysname:]channelname */

DCL
RETURN_CODE FIXED BIN(31),
LANGUAGE_INDICATOR FIXED BIN(31),
LOGIN_INFO CHAR(15),
UPDATE_INDICATOR FIXED BIN(31),
THREAD_ID FIXED BIN(31),
```

```
CRAM_CHANNEL CHAR(13) INIT('SSN1:MYCHAN'),
CRAM_CHANNEL_ARG FIXED BIN(31) BASED(P);

/* THE FOLLOWING STATEMENT SETS THE POINTER P TO THE */
/* ADDRESS OF CRAM_CHANNEL. P IS ALSO THE ADDRESS OF */
/* CRAM_CHANNEL_ARG. */

P = ADDR(CRAM_CHANNEL);

CALL IFSTRTN(RETURN_CODE,LANGUAGE_INDICATOR,LOGIN_INFO,
THREAD_ID,CRAM_CHANNEL_ARG);
```

The following call is for IFDIALN:

```
CALL IFDIALN(RETURN_CODE,LANGUAGE_INDICATOR,
CRAM_CHANNEL_ARG)
```

Assembler language coding guidelines

Assembler language programs usually adhere to the standard calling sequence used by COBOL and FORTRAN and are subject to the same conventions that govern parameter formats and lengths.

Note, however, when starting an IFSTRT, IFSTRTN, IFDIAL, or IFDIALN thread, any of the three language indicator values (1, 2, or 3) can be used with an Assembler language application program.

Accordingly, the calling sequences and data structures that correspond to the particular language specified in the thread call must be used in the Assembler application program.

Regardless of the language indicator specified, use the following coding conventions when Assembler is the host language:

- The IFFIND call must have the following format:

  ```
  CALL IFFIND,(ERR,QUAL1),VL
  ```

- Under z/OS, all calls must end with the VL parameter.

Pascal/VS coding guidelines

Use the following conventions when Pascal/VS is the host language:

- Declare integer arguments as type INTEGER. For example:

  ```
  VAR I:INTEGER;
  ```

- Declare character string parameters as PACKED ARRAY OF CHAR, and end their values with a semicolon. For example:

  ```
  TYPE M204.STR = PACKED ARRAY (.1..255.) OF CHAR;
  VAR FILENAME: M204.STR;
  FILENAME:='TESTFILE;PASS;';
  ```

- Declare HLI functions as external FORTRAN procedures and pass all arguments by REFERENCE, using the VAR keyword. For example:

  ```
  PROCEDURE IFOPEN (VAR ERROR:INTEGER;VARFILENAME:M204.STR);
  FORTRAN;
  ```

- Use language indicator 2 in the IFSTRT call.

Coding guidelines for other languages

You can use Host Language Interface functions for languages other than COBOL, FORTRAN, PL/I, Assembler, and Pascal.

Follow the conventions below for passing parameters depending on the language indicator specified in the IFSTRT, IFSTRTN, IFDIAL, or IFDIALN call that establishes the thread:

- For language indicator 1, you can call a Host Language Interface function if the compiler can generate parameter lists that adhere to standard register usage.

  Note: Register 1 must point to the parameter list and consist of a list of memory addresses.

- For language indicator 2, the addresses point directly to the parameter values. The last valid parameter is flagged by setting the high-order byte of the parameter address.

- Many language compilers that do not ordinarily follow the parameter passing conventions listed above have special options for calling external FORTRAN and Assembler language routines. For example, Pascal/VS passes parameters by value in some cases.

  These special options allow Host Language Interface functions to be invoked using language indicator 2.

5

HLI Function Summary

Overview

This chapter summarizes the calls that are available using the Model 204 Host Language Interface facility. Read this chapter if you are using the HLI facility for the first time.

See page 73 for a complete listing of HLI function calls.

For more information

See Chapter 6 for a detailed description of each call.

This chapter does not provide specific descriptions of HLI calls or information about coding an application program using the calls. For information about coding application programs using HLI calls, refer to the *Rocket Model 204 Host Language Interface Programming Guide*.

IFDIAL thread calls

IFDIAL thread

An IFDIAL thread provides a line-at-a-time terminal type interface between Model 204 and a host language program that is running in batch.

With an IFDIAL thread, an application that is written in a host language can transmit data to and from Model 204 using the IFWRITE and IFREAD calls.

See Chapter 3 for information about setting up and running the batch Model 204 job using an IFDIAL thread.

Summary of IFDIAL calls

Table 5-1 summarizes the HLI calls that are available for use only with an IFDIAL thread. See Chapter 6 for a detailed description of each call.

Table 5-1. IFDIAL functions

Call	Function
IFATTN	Sends an attention interrupt.
IFDIAL	Starts an IFDIAL thread.
IFDIALN	Starts an IFDIAL thread using a specified communications channel name.
IFHNGUP	Ends all threads that are started in a job.
IFREAD	Gets a line of output from Model 204.
IFSETUP	Initiates contact with Model 204 and sets the PARM parameters and CCAIN statements for the IFAM1 job.
IFWRITE	Sends a line of input to Model 204.

IFSTRT thread calls

IFSTRT thread

An IFSTRT thread provides a user interface between Model 204 and a host language program that is running in batch.

With an IFSTRT thread, an application that is written in a host language can issue calls to Model 204 that perform operations against the database which are similar to Model 204 commands and SOUL statements.

See Chapter 3 for information about setting up and running the Online or batch Model 204 job using an IFSTRT thread. See Chapter 6 for a detailed description of the IFSTRT call.

Different operational levels

The HLI calls that are available for use with an IFSTRT thread are categorized by functionality and operate at the following levels:

- System
- Transaction
- File or group

- Set
- Single record

A typical application program works at all levels, using the HLI calls that are available at each level. The calls that are available in each of these categories are summarized in the tables on the following pages.

Enqueuing action and record locking behavior

When several users have access to the same files or groups, Model 204 prevents conflicting, simultaneous use of records with a facility called enqueuing. Enqueuing is performed at the thread level.

With an IFSTRT thread, most HLI function calls automatically enqueue and dequeue on the resources, such as the files, groups, sets, or records, to which they refer.

See Chapter 6 for detailed information about the record locking behavior of individual calls. Refer to the *Rocket Model 204 Host Language Interface Programming Guide* for more information about enqueueing on an IFSTRT thread.

System level IFSTRT calls

System level functions constitute the highest level of access to the Host Language Interface Model 204 service routines. At the system level, most functions do not require a file or group context.

Table 5-2 summarizes the IFSTRT calls that function at the system level.

Table 5-2. System level IFSTRT calls

Call	Function
IFCALL	Calls another HLI function by number.
IFCHKPT	Requests a checkpoint, checkpoint status information, or both. (Note that IFCHKPT is also a transaction-level function.)
IFDTHRD	Deletes the current thread and switches to another thread.
IFEFCC	Returns specific information about field values or record numbers that cause a field constraint conflict using the IFSTOR, IFUPDT, or IFPUT HLI calls
IFEPRM	Reads a parameter.
IFERLC	Returns a file name, record number, and user name after a record locking conflict occurs in an HLI program that issues a call that requires locking a record
IFERR	Places a message on the journal, requests a snap dump of the Model 204 region, or both.

Table 5-2. System level IFSTRT calls

Call	Function
IFCALL	Calls another HLI function by number.
IFFNSH	Deallocates all threads; finishes HLI portion of program.
IFGERR	Reads the latest error message produced by Model 204.
IFLOG	Logs in user with user ID and password (IFAM1).
IFRPRM	Resets a parameter.
IIFSPRM	Sets a parameter.
IFSTHRD	Switches from one thread to another.
IIFSTRT	Establishes an IFSTRT thread connection.
IFSTRTN	Establishes an IFSTRT thread connection to an alternate HLI Model 204 service program.
IFUTBL	Resets the size of a user's server tables.

Transaction level IFSTRT calls

Transaction level functions operate in a job that uses the Host Language Interface facility to manage transactions for a logical unit of work processing against the Model 204 database.

Table 5-3 summarizes the IFSTRT calls that function at the transaction level.

Table 5-3. Transaction level IFSTRT calls

Call	Function
IFBOUT	Backs out current transaction.
IFCHKPT	Requests a checkpoint or checkpoint status or both. (Note that IFCHKPT is also a system-level function.)
IFCMMT	Commits the current transaction.
IFCMTR	Releases all record sets and commits the current transaction.
IFRELA	Releases all record sets.

File or group level IFSTRT calls

HLI functions at the file or group level operate with a file or group.

For example, whenever an application program requires access to a Model 204 file, a file or group must be opened. IFOPEN must be called before any other file or group level functions. IFOPEN establishes the current file or group on which subsequent file or group level, record level, and set level functions operate.

Note: On a single cursor IFSTRT thread, each thread may have only one current file or group. All functions operate against this file or group. On a multiple cursor IFSTRT thread multiple files or groups may be accessed, and the file or group that is specified in the HLI call is the one that is current for processing. If the file specification is optional and no file is specified, the file that was opened last is current, by default.

Table 5-4 summarizes the IFSTRT calls that function at the file or group level.

Table 5-4. File or group level IFSTRT calls

Call	Function
IFCLOSE	Closes all files and groups for the current thread.
IFDELF	Deletes a field definition in a file. IFDELF can be used only if the current context is a file.
IFDFLD	Defines new fields in a file.
IFDISP	Displays file, group, and other information.
IFFLS	Checks for field-level security violations.
IFINIT	Initializes a file. IFINIT can be used only if the current context is a file.
IFNFLD	Renames fields in a file. IFNFLD can be used only if the current contest is a file.
IFOPEN	Opens a file or group.
IFOPENX	Opens a file or group, enqueuing upon the file(s) in exclusive status.
IFRFLD	Redefines fields in a file. IFRFLD can be used only if the current context is a file.

Record set level IFSTRT calls

Functions at the record set level operate with sets of records or field values.

For example, the IFFIND function must be called to establish a current set before any other set level functions can operate successfully.

On a multiple cursor IFSTRT thread, the set that is specified in the HLI call is the one that is current for processing. On a single cursor IFSTRT thread, the current set is the one last created.

The IFSTRT calls that function at the record set level are summarized in Tables 5-5, 5-6, and 5-7 on the following pages. The tables are described briefly below.

- Table 5-5 beginning on page 68 lists those calls that may be used on both multiple cursor and single cursor IFSTRT threads.

- Table 5-6 on page 69 lists those calls that may be used only on a multiple cursor IFSTRT thread.
- Table 5-7 on page 69 lists those calls that may be used only on a single cursor IFSTRT thread.

Record set level calls on any IFSTRT thread

Table 5-5 lists calls that can be used on both multiple cursor and single cursor IFSTRT threads.

Table 5-5. Record set level calls on any IFSTRT thread

Call	Function
IFCOUNT	Counts the number of records in the current set.
IFDSET	Deletes the current set of records from the file or group.
IFFAC	Finds and returns record count.
IFFACC	Compiles an IFFAC specification.
IFFACE	Executes a precompiled IFFAC specification.
IFFDV	Selects a set of field values to become the current value set.
IFFDVC	Compiles an IFFDV specification.
IFFDVE	Executes a precompiled IFFDV specification.
IFFILE	Adds an invisible key field to all records in the current set.
IFFIND	Selects a set of records as the current set.
IFFINDC	Compiles an IFFIND specification.
IFFINDE	Executes a precompiled IFFIND specification.
IFFNDX	Selects a set of records and enqueues upon them exclusively.
IFFNDXC	Compiles an IFFNDX specification.
IFFNDXE	Executes a precompiled IFFNDX specification.
IFFWOL	Selects a set of records as the current set, without locking.
IFFWOLC	Compiles an IFFWOL specification.
IFFWOLE	Executes a precompiled IFFWOL specification.
IFSKEY	Sorts the records in the current IFFIND set; only the record key is written to the sort records.
IFSKYC	Compiles the IFSKEY specification.
IFSKYE	Executes the precompiled IFSKEY specification.
IFSORT	Sorts the records in the current set.
IFSRTC	Compiles the IFSORT sort specification.

Table 5-5. Record set level calls on any IFSTRT thread (Continued)

Call	Function
IFSRTE	Executes the precompiled IFSORT or IFSRTC specification.
IFSRTV	Sorts the values in the current value set.
IFSTVC	Compiles the IFSRTV specification.
IFSTVE	Executes the precompiled IFSRTV specification.

Record set level calls on a multiple cursor IFSTRT thread

Table 5-6 lists those calls that may be used only on a multiple cursor IFSTRT thread.

Table 5-6. Record set level calls on a multiple cursor IFSTRT thread

Call	Function
IFCCUR	Closes a cursor on a set.
IFCLST	Clears a list.
IFOCUR	Opens a cursor on a set.
IFOCURC	Compiles the IFOCUR specification.
IFOCURE	Executes the precompiled IFOCUR specification.
IFPROLS	Places records from a found set onto a list.
IFRELR	Releases records in a found set.
IFRRFLS	Removes records in a found set from a list.

Record set level calls on a single cursor IFSTRT thread

Table 5-7 lists those calls that may be used only on a single cursor IFSTRT thread.

Table 5-7. Record set level calls on a single cursor IFSTRT thread

Call	Function
IFDEQL	Dequeues the set of records on a specified list.
IFENQL	Enqueues in share or exclusive mode on the set of records of the specified list.
IFLIST	Places the records in the current set onto a named list.

Individual record level IFSTRT calls

Individual record functions operate on a record.

On a single cursor IFSTRT thread, IFGET, IFBREC, and IFPOINT establish the current record on which they and other functions at this level operate. On a multiple cursor IFSTRT thread, the current record in the cursor that is specified in the HLI call is the record that is current for processing. The current record in a cursor is established by an IFFTCH, IFSTOR, or IFFRN call.

The IFSTRT calls that function at the single record level are summarized in Table 5-8, and in Tables 5-9 and 5-10. The tables are described briefly below.

- Table 5-8 lists those calls that may be used on both multiple cursor and single cursor IFSTRT threads.
- Table 5-9 on page 70 lists those calls that may be used only on a multiple cursor IFSTRT thread.
- Table 5-10 on page 71 lists those calls that may be used only on a single cursor IFSTRT thread.

Table 5-8. Individual record level calls on any IFSTRT thread

Call	Function
IFDALL	Deletes all the occurrences of a field from the current record.
IFDREC	Deletes the current record from its file.
IFDVAL	Deletes a field name = value pair.
IFPROL	Places the current record on a named list.
IFRRFL	Removes the current record from a named list.

Individual record level calls on a multiple cursor IFSTRT thread

Table 5-9 lists those calls that may be used only on a multiple cursor IFSTRT thread.

Table 5-9. Individual record level calls on a Multiple Cursor IFSTRT thread

Call	Function
IFFRN	Points to the specified record in the specified file and makes it the current record.
IFFRNC	Compiles the IFFRN specification.
IFFRNE	Executes the precompiled IFFRN specification.
IFFTCH	Processes the next logical record or value and returns the specified data.
IFFTCHC	Compiles the IFFTCH specification.
IFFTCHE	Executes the precompiled IFFTCH specification.

Table 5-9. Individual record level calls on a Multiple Cursor IFSTRT thread (Continued)

Call	Function
IFOCC	Counts the number of occurrences of the specified field in the current record and returns a count.
IFOCCC	Compiles the IFOCC specification.
IFOCCE	Executes the precompiled IFOCC specification.
IFRNUM	Returns the number of the current record in the specified cursor.
IFSTOR	Creates a new record with the specified data and adds the record to the specified file.
IFSTRC	Compiles the IFSTOR specification.
IFSTRE	Executes the precompiled IFSTOR specification.
IFUPDT	Updates the current record with the specified data.
IFUPDTC	Compiles the IFUPDT specification.
IFUPDTE	Executes the precompiled IFUPDT specification.

Individual record level calls on a single cursor IFSTRT thread

Table 5-10 lists those calls that may be used only on a single cursor IFSTRT thread.

Table 5-10. Individual record level calls on a single cursor IFSTRT thread

Call	Function
IFBREC	Creates a new record.
IFCTO	Counts the number of field occurrences in the current record.
IFCTOC	Compiles an IFCTO specification.
IFCTOE	Executes a precompiled IFCTO specification.
IFGET	Reads information from the next record in the current set.
IFGETC	Compiles an IFGET specification.
IFGETE	Executes a precompiled IFGET specification.
IFGETV	Reads the next value from the current value set.
IFGETX	Reads information from the next record, enqueuing upon it exclusively.
IFGETXE	Executes a precompiled IFGET specification, enqueuing upon the record exclusively.
IFGTVC	Compiles the IFGETV specification.

Table 5-10. Individual record level calls on a single cursor IFSTRT thread (Continued)

Call	Function
IFGTVE	Executes the precompiled IFGETV.
IFMORE	Reads more information from the current record.
IFMOREC	Compiles an IFMORE specification.
IFMOREE	Executes a precompiled IFMORE specification.
IFMOREX	Reads more information from the current record, enqueuing upon the record exclusively.
IFMORXE	Executes a precompiled IFMORE specification, enqueuing upon the record exclusively.
IFPOINT	Specifies a new current record.
IFPUT	Updates the current record.
IFPUTC	Compiles an IFPUT record.
IFPUTE	Executes a precompiled IFPUT specification.

IFSTRT thread calls and compiled IFAM

Compiled IFAM facility

The Compiled IFAM (Inverted File Access Method) facility allows IFSTRT thread calls to be compiled and stored.

Using the Compiled IFAM facility, 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.

A compilation must be uniquely identified so that multiple calls can use it. The name parameter included in all Compiled IFAM calls specifies a character string that is used to identify the compilation. A null name string is the same as an omitted parameter.

Three forms of Compiled IFAM calls

Three forms of IFSTRT calls are available using the Compiled IFAM facility: compile and execute, compile-only, and execute-only. The following options are available to accommodate different programming styles:

- Using a single call that compiles and executes with the name parameter that identifies the compilation. The call executes and the compiled version of the call is saved.

When the same call is executed again or when another call containing the same name parameter is executed, the stored compilation is executed without requiring recompilation.

- Using two calls, one is compile-only and one is execute-only, 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. The compilation form of the call is used outside the loop to compile (but not execute) the call specification. Within the loop, the execution form of the call is issued, thereby executing the previously compiled call.

See Table 5-11 for an overview of the calls that provide Compiled IFAM functionality for IFSTRT thread processing.

Refer to the *Rocket Model 204 Host Language Interface Programming Guide* for more information about IFSTRT calls and the Compiled IFAM facility.

Complete listing of HLI function calls

Table 5-11 lists the HLI calls in alphabetical order. Names that are underlined indicate an IFDIAL thread call, all others are IFSTRT thread calls. For each call, this table provides the following information:

- # lists the number which may be used to reference the function using IFCALL. Note that IFCALL, IFLOG, and IFSTRT (in IFAM1) do not have numbers.
- An asterisk under Alias indicates that the name is an alias, that is, it is the shortened COBOL form (six characters or less in length) for the equivalent call (under Equivalent). A dash indicates that the call name is not an alias.
- An entry under Compiled IFAM indicates whether the call performs compile and execute, compile only, or execute only processing. Related call(s) lists functionally similar calls which provide alternate Compiled IFAM processing. For example, for IFCTO (which compiles and executes), related calls are: IFCTOC (compiles only) and IFCTOE (executes only).
- An asterisk under MC indicates that the call is valid for use with a multiple cursor IFSTRT thread.

Table 5-11 provides a summary overview of the calls that are available using the HLI facility, including their Compiled IFAM and multiple cursor (MC) functionality.

Table 5-11. HLI function calls

Name	#	Alias	Equivalent	Compiled IFAM	Related call(s)	MC
IFABXIT	97	-	-	-	-	*
IFATTN	43	-	-	-	-	-

Table 5-11. HLI function calls (Continued)

Name	#	Alias	Equivalent	Compiled IFAM	Related call(s)	MC
IFBOUT	71	-	-	-	-	*
IFBREC	20	-	-	-	-	-
IFCALL	-	-	-	-	-	*
IFCCUR	125	-	-	-	-	*
IFCHKP	30	*	IFCHKPT	-	-	*
IFCHKPT	30	-	IFCHKP	-	-	*
IFCLOS	12	*	IFCLOSE	-	-	*
IFCLOSE	12	-	IFCLOS	-	-	*
IFCLST	108	-	-	-	-	*
IFCMMT	72	-	-	-	-	*
IFCMTR	105	-	-	-	-	*
IFCNT	14	*	IFCOUNT	-	-	*
IFCOUNT	14	-	IFCNT	-	-	*
IFCSA	96	-	-	-	-	*
IFCTO	65	-	-	Compile and execute	IFCTOC, IFCTOE	-
IFCTOC	66	-	-	Compile only	IFCTO, IFCTOE	-
IFCTOE	67	-	-	Execute only	IFCTO, IFCTOC	-
IFDALL	73	-	-	-	-	*
IFDELF	62	-	-	-	-	*
IFDEQ	38	-	-	-	-	*
IFDEQL	41	-	-	-	-	-
IFDFLD	24	-	-	-	-	*
IFDIAL	6	-	-	-	-	-
IFDIALN	7	-	IFDILN	-	-	-
IFDILN	7	*	IFDIALN	-	-	-
IFDISP	60	-	-	-	-	*
IFDREC	19	-	-	-	-	*
IFDSET	21	-	-	-	-	*
IFDTHRD	42	-	IFDTRD	-	-	*

Table 5-11. HLI function calls (Continued)

Name	#	Alias	Equivalent	Compiled IFAM	Related call(s)	MC
IFDTRD	42	*	IFDTHRD	-	-	*
IFDVAL	32	-	-	-	-	*
IFEFCC	139	-	-	-	-	*
IFENQ	39	-	-	-	-	*
IFENQL	40	-	-	-	-	-
IFEPRM	25	-	-	-	-	*
IFERLC	138	-	-	-	-	*
IFERR	28	-	-	-	-	*
IFFAC	126	-	-	Compile and execute	IFFACC, IFFACE	*
IFFACC	127	-	-	Compile only	IFFAC, IFFACE	*
IFFACE	128	-	-	Execute only	IFFAC, IFFACC	*
IFFCHC	99	*	IFFTCHC	Compile only	IFFTCH, IFFCHE	*
IFFCHE	100	*	IFFTCHE	Execute only	IFFTCH, IFFCHC	*
IFFD	13	*	IFFIND	Compile and execute	IFFDC, IFFDE	*
IFFDC	46	*	IFFINDC	Compile only	IFFD, IFFDE	*
IFFDE	47	*	IFFINDE	Execute only	IFFD, IFFDC	*
IFFDV	74	-	-	Compile and execute	IFFDVC, IFFDVE	*
IFFDVC	75	-	-	Compile only	IFFDV, IFFDVE	*
IFFDVE	76	-	-	Execute only	IFFDV, IFFDVC	*
IFFDX	56	*	IFFNDX	Compile and execute	IFFDXC, IFFDXE	*
IFFDXC	57	*	IFFNDXC	Compile only	IFFDX, IFFDXE	*
IFFDXE	58	*	IFFNDXE	Execute only	IFFDX, IFFDXC	*
IFFILE	22	-	-	-	-	*
IFFIND	13	-	IFFD	Compile and execute	IFFINDC, IFFINDE	*
IFFINDC	46	-	IFFDC	Compile only	IFFIND, IFFINDE	*
IFFINDE	47	-	IFFDE	Execute only	IFFIND, IFFINDC	*
IFFLS	61	-	-	-	-	*

Table 5-11. HLI function calls (Continued)

Name	#	Alias	Equivalent	Compiled IFAM	Related call(s)	MC
IFFLSH	45	*	IFFLUSH	-	-	*
IFFLUSH	45	-	IFFLSH	-	-	*
IFFNDX	56	-	IFFDX	Compile and execute	IFFNDXC, IFFNDXE	*
IFFNDXC	57	-	IFFDXC	Compile only	IFFNDX, IFFNDXE	*
IFFNDXE	58	-	IFFDXE	Execute only	IFFNDX, IFFNDXC	*
IFFNSH	3	-	-	-	-	*
IFFRN	118	-	-	Compile and execute	IFFRNC, IFFRNE	*
IFFRNC	119	-	-	Compile only	IFFRN, IFFRNE	*
IFFRNE	120	-	-	Execute only	IFFRN, IFFRNC	*
IFFTCH	98	-	-	Compile and execute	IFFTCHC, IFFTCHE	*
IFFTCHC	99	-	IFFCHC	Compile only	IFFTCH, IFFTCHE	*
IFFTCHE	100	-	IFFCHE	Execute only	IFFTCH, IFFTCHC	*
IFFWO	87	*	IFFWOL	Compile and execute	IFFWOC, IFFWOE	*
IFFWOC	88	*	IFFWOLC	Compile only	IFFWO, IFFWOE	*
IFFWOE	89	*	IFFWOLE	Execute only	IFFWO, IFFWOC	*
IFFWOL	87	-	IFFWO	Compile and execute	IFFWOLC, IFFWOLE	*
IFFWOLC	88	-	IFFWOC	Compile only	IFFWOL, IFFWOLE	*
IFFWOLE	89	-	IFFWOE	Execute only	IFFWOL, IFFWOLC	*
IFGERR	29	-	-	-	-	*
IFGET	15	-	-	Compile and execute	IFGETC, IFGETE	-
IFGETC	48	-	-	Compile only	IFGET, IFGETE	-
IFGETE	49	-	-	Execute only	IFGET, IFGETC	-
IFGETV	77	-	-	Compile and execute	IFGTVC, IFGTVE	-
IFGETX	36	-	-	Compile and execute	IFGETXE, IFGETC	-

Table 5-11. HLI function calls (Continued)

Name	#	Alias	Equivalent	Compiled IFAM	Related call(s)	MC
IFGETXE	50	-	IFGTXE	Execute only	IFGETX	-
IFGTVC	78	-	-	Compile only	IFGETV, IFGTVE	-
IFGTVE	79	-	-	Execute only	IFGETV, IFGTVC	-
IFGTXE	50	*	IFGETXE	Execute only	IFGETX	-
IFHNGP	8	*	IFHNGUP	-	-	-
IFHNGUP	8	-	IFHNGP	-	-	-
IFINIT	23	-	-	-	-	*
IFLIST	17	-	-	-	-	-
IFLOG	-	-	-	-	-	*
IFMORE	16	-	-	Compile and execute	IFMREC, IFMREE, IFMOREC, IFMOREE	-
IFMOREC	51	-	IFMREC	Compile only	IFMORE, IFMOREE	-
IFMOREE	52	-	IFMREE	Execute only	IFMORE, IFMOREC	-
IFMOREX	37	-	IFMREX	Compile and execute	IFMORXE, IFMOREC	-
IFMORXE	53	-	IFMRXE	Execute only	IFMOREX	-
IFMREC	51	*	IFMOREC	Compile only	IFMORE, IFMREE	-
IFMREE	52	*	IFMOREE	Execute only	IFMORE, IFMREC	-
IFMREX	37	*	IFMOREX	Compile and execute	IFMRXE	-
IFMRXE	53	*	IFMORXE	Execute only	IFMREX	-
IFNFLD	63	-	-	-	-	*
IFOCC	122	-	-	Compile and execute	IFOCCC, IFOCCE	*
IFOCCC	123	-	-	Compile only	IFOCC, IFOCCE	*
IFOCCE	124	-	-	Execute only	IFOCC, IFOCCC	*
IFOCRC	106	*	IFOCURC	Compile only	IFOCUR, IFOCRE	*
IFOCRE	107	*	IFOCURE	Execute only	IFOCUR, IFOCRC	*
IFOCUR	95	-	-	Compile and execute	IFOCRC, IFOCRE, IFOCURC, IFOCURE	*

Table 5-11. HLI function calls (Continued)

Name	#	Alias	Equivalent	Compiled IFAM	Related call(s)	MC
IFOCURC	106	-	IFOCRC	Compile only	IFOCUR, IFOCURE	*
IFOCURE	107	-	IFOCRE	Execute only	IFOCUR, IFOCURC	*
IFOPEN	11	-	-	-	-	*
IFOPENX	35	-	IFOPNX	-	-	*
IFOPNX	35	*	IFOPENX	-	-	*
IFPNT	44	*	IFPOINT	-	-	-
IFPOINT	44	-	IFPNT	-	-	-
IFPRLS	109	*	IFPROLS	-	-	*
IFPROL	33	-	-	-	-	*
IFPROLS	109	-	IFPRLS	-	-	*
IFPUT	18	-	-	Compile and execute	IFPUTC, IFPUTE	-
IFPUTC	54	-	-	Compile only	IFPUT, IFPUTE	-
IFPUTE	55	-	-	Execute only	IFPUT, IFPUTC	-
IFREAD	9	-	-	-	-	-
IFRELA	104	-	-	-	-	*
IFRELR	103	-	-	-	-	*
IFRFLD	59	-	-	-	-	*
IFRFLS	110	*	IFRRFLS	-	-	*
IFRNUM	121	-	-	-	-	*
IFRPRM	27	-	-	-	-	*
IFRRFL	34	-	-	-	-	*
IFRRFLS	110	-	IFRFLS	-	-	*
IFSETP	86	*	IFSETUP	-	-	-
IFSETUP	86	-	IFSETP	-	-	-
IFSKEY	83	-	-	Compile and execute	IFSKYC, IFSKYE	*
IFSKYC	84	-	-	Compile only	IFSKEY, IFSKYE	*
IFSKYE	85	-	-	Execute only	IFSKEY, IFSKYC	*

Table 5-11. HLI function calls (Continued)

Name	#	Alias	Equivalent	Compiled IFAM	Related call(s)	MC
IFSORT	68	-	-	Compile and execute	IFSRTC, IFSRTE	*
IFSPRM	26	-	-	-	-	*
IFSRTC	69	-	-	Compile only	IFSORT, IFSRTE	*
IFSRTE	70	-	-	Execute only	IFSORT, IFSRTC	*
IFSRTV	80	-	-	Compile and execute	IFSTVC, IFSTVE	*
IFSTHRD	2	-	IFSTRD	-	-	*
IFSTOR	112	-	-	Compile and execute	IFSTRC, IFSTRE	*
IFSTRC	113	-	-	Compile only	IFSTOR, IFSTRE	*
IFSTRD	2	*	IFSTHRD	-	-	*
IFSTRE	114	-	-	Execute only	IFSTOR, IFSTRC	*
IFSTRN	4	*	IFSTRTN	-	-	*
IFSTRT[1]	-	-	-	-	-	*
IFSTRT[2]	1	-	-	-	-	*
IFSTRTN	4	-	IFSTRN	-	-	*
IFSTVC	81	-	-	Compile only	IFSRTV, IFSTVE	*
IFSTVE	82	-	-	Execute only	IFSRTV, IFSTVC	*
IFUPDC	116	*	IFUPDTC	Compile only	IFUPDT, IFUPDE	*
IFUPDE	117	*	IFUPDTE	Execute only	IFUPDT, IFUPDC	*
IFUPDT	115	-	-	Compile and execute	IFUPDC, IFUPDE, IFUPDTC, IFUPDTE	*
IFUPDTC	116	-	IFUPDC	Compile only	IFUPDT, IFUPDTE	*
IFUPDTE	117	-	IFUPDE	Execute only	IFUPDT, IFUPDTC	*
IFUTBL	64	-	-	-	-	*
IFWRIT	10	*	IFWRITE	-	-	-
IFWRITE	10	-	IFWRIT	-	-	-

1. In IFAM1

2. In IFAM2 and IFAM4

6

HLI Function Calls

Overview

This chapter describes in detail all of the calls that are available using the Host Language Interface. Use the information in this chapter to code the HLI calls in your application program.

The calls are arranged in alphabetical order to allow for quick reference.

For more information

When designing program logic, the application programmer must be familiar with the basic difference between IFSTRT and IFDIAL call protocols. Also, when coding the calls the programmer needs to know about differences in functionality using the different types of threads, and must be aware of the difference between using multiple cursor and single cursor IFSTRT threads.

For more information about differences in HLI functionality, see the *Rocket Model 204 Host Language Interface Programming Guide*.

See Chapter 4 for information about coding HLI call parameters using different host languages.

Function call notation conventions

Standard notation conventions are used in the detailed description of each call. The conventions for call syntax and parameters that are described in this section are used in this document.

Identifying which type of thread for the call

The code(s) beside the call name at the top of a page indicate(s) the type of thread usage that is valid for the call. The following codes are used:

mc	Indicates a multiple cursor IFSTRT thread
sc	Indicates a single cursor IFSTRT thread
di	Indicates an IFDIAL thread

For example, the IFFIND call, which selects records and creates a found set, is identified in the section “IFFIND call -mc,sc” on page 167 as:

IFFIND Call -mc, sc

Both the **mc** and **sc** codes indicate that IFFIND may be used with either a multiple cursor (mc) or a single cursor (sc) IFSTRT thread.

Call name and syntax

The syntax of host language calls is presented in the following format:

`callname | alias (parm`*`1`*`, parm`*`2`*`,•••parm`*`n`*`)`

where:

- *callname* is a keyword which specifies the name of the HLI function; an *alias*, if available, is also provided. The alias is a name that is six characters or less in length to be used for FORTRAN programming. A vertical bar separates the callname from the alias, for example:

  ```
  IFFIND | IFFD
  ```

- *parm1* is the first parameter in the call list. Parameters must be specified in the proper syntax order, as shown (parm2 follows parm1, and so on, parmn is last).
- A comma delimiter (,) separates parameters in the list, and the entire parameter list is enclosed inside parentheses, for illustrative purposes only. (When coding calls, use the delimiter and format that is valid in your host language.)

Different forms of call syntax

The individual call description includes the full syntax and, where available, the compile-only and execute-only forms of the call. For example, with IFFIND (full syntax), the IFFINDC (compile-only syntax) and IFFINDE (execute-only syntax) forms are also provided. Compile-only or execute-only syntax is used with the Compiled IFAM facility.

Also, each form of the call, that is, full syntax, compile-only syntax, and execute-only syntax, is identified by a call number inside parentheses, where available. Note that you can use IFCALL to call functions that have numbers.

For example, for the three forms of the find function call:

- IFFIND Full syntax (13)
- IFFINDC Compile-only syntax (46)
- IFFINDE Execute-only syntax (47)

Parameters

Each parameter description provides a three-character code having the following format:

`[X,y,z]`

where:

X	Specifies that the parameter is used for input or output operations, and is either of the following codes:
I (input)	
O (output)	
y	Specifies that the parameter, if coded, must be defined in the host language program as one of three possible data type variables, and is one of the following codes:
i	For an integer whose maximum length is 4 bytes
s	For a short character string whose maximum length is 32 bytes
c	For a character string whose maximum length is the buffer size
z	Specifies whether the parameter must be coded in the call, and is either (or both) of the following codes:
r (required)	
o (optional)	

For example, RETCODE [O,i,r] where:

O	Specifies that RETCODE is an output parameter
i	Specifies an integer type variable (maximum length is 4 bytes)
r	Specifies that RETCODE is required

Certain parameters may be either r or o depending, in some cases, on whether the call is being coded on a multiple cursor IFSTRT thread or, in other cases, whether the call is used for Compiled IFAM.

For example, for IFFAC, the compilation name parameter is FAC_NAME [I,s,r/o] where:

I	Specifies that FAC_NAME is an input parameter
s	Specifies a short character string type variable (maximum length is 32 bytes)
r/o	Specifies that FAC_NAME is required for use with a multiple cursor IFSTRT thread, and is only required for a single cursor IFSTRT thread if using the compiled IFAM facility (IFFACC and IFFACE).

For an r/o code, read the parameter description to determine usage requirements.

IFABXIT call *-mc,sc*

Function

The IFABXIT call (ABEND EXIT) establishes the IFAM2 abend handler. IFABXIT is available only for use with the IFAM2 CICS interface.

Full syntax (97)

```
IFABXIT
```

Compile-only syntax

A compile-only form of IFABXIT is not available.

Execute-only syntax

An execute-only form of IFABXIT is not available.

Parameters

No parameters available for use with the IFABXIT call.

Notes and tips

Use the IFABXIT call with an IFAM2 connection to Model 204 only in a CICS environment.

Use IFABXIT if the host language application does not require its own abend handler. Using IFABXIT protects the application against a hung CRAM channel in the event of an abend prior to the first functional IFAM2 call.

Issue the IFABXIT call as soon as possible after entering the program, after the initial IFCSA call.

See the *Rocket Model 204 Host Language Interface Programming Guide* for more information about CICS abend handling in IFAM2.

Coding example (COBOL)

```
•
•
•
PROCEDURE DIVISION.
•
•

CALL “IFABXIT”.
```

IFATTN call *-di*

Function The IFATTN call (ATTENTION) sends an attention interrupt signal.

Full syntax (43)

```
IFATTN(RETCODE)
```

Compile-only syntax A compile-only form of IFATTN is not available.

Execute-only syntax An execute-only form of IFATTN is not available.

Parameters

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required output parameter. The code is a binary integer value.

Notes and tips Use the IFATTN call only with an IFDIAL connection.

The IFATTN call can be issued when a completion code of 1, 2, or 12 is returned from the previous call to IFREAD or IFWRITE call.

When the IFATTN call is issued, any current SOUL request is purged. Follow IFATTN with IFWRITE if processing is to continue.

Completion return code (RETCODE) If the IFATTN call is unsuccessful, Model 204 returns an error code of 4 if the connection was lost.

Coding example (COBOL)

```
•
•
•
PROCEDURE DIVISION.
•
•
•
CALL “IFATTN” USING RETCODE.
```

IFBOUT call *-mc,sc*

Function

The IFBOUT call (BACK OUT) backs out an incomplete transaction on the thread or threads that are active for the transaction in which the thread participates.

Full syntax (71)

```
IFBOUT(RETCODE)
```

Compile-only syntax

A compile-only form of IFBOUT is not available.

Execute-only syntax

An execute-only form of IFBOUT is not available.

Parameters

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required output parameter. The code is a binary integer value.

Notes and tips

Use the IFBOUT call to activate the Model 204 backout mechanism. The backout function is valid only for transaction backout (TBO) files.

You can issue the IFBOUT call on the following IFSTRT threads:

- In IFAM2/IFAM4, a single cursor IFSTRT thread with update privileges or a multiple cursor IFSTRT thread.

  Note that you cannot issue IFBOUT on a single cursor IFSTRT thread with read-only privileges, that is, with a thread type indicator of 0.

- In IFAM1, a single cursor or multiple cursor IFSTRT thread

  Note: Only the thread, or threads, that participate in the current transaction are backed out. Any IFAM1 type IFSTRT thread involves a single-threaded transaction.

Single cursor IFSTRT threads that are started in a multithreaded IFAM2 or IFAM4 HLI job participate in a multithreaded transaction. See the *Rocket Model 204 Host Language Interface Programming Guide* for more information about transaction processing in an HLI job.

Completion return code (RETCODE)

IFBOUT is not valid on a read-only type thread, that is, any thread that is started by IFSTRT with an update indicator of 0 for an IFAM2 / IFAM4 connection.

If the IFBOUT call is unsuccessful, Model 204 returns the following error completion codes:

Code	Error condition
4	IFBOUT called for a non-transaction backout (non-TBO) file or for a read-only thread.
40	IFBOUT call is encountered on a single cursor IFSTRT thread having read-only privileges.

Record locking behavior

IFBOUT releases the lock pending updates (LPU) exclusive lock on updated records and the single record enqueue (SRE) on the current record (if one exists) that were obtained by this thread. The transaction is ended for all participating threads and the backout and constraints log are freed.

Found sets are not released and the current record does not change. In a multithreaded transaction the LPU and SRE locks obtained by other threads are still held.

To ensure that all of the records and resource locks are released after the IFBOUT call executes in a multithreaded transaction, issue IFCMMT on all of the other single cursor IFSTRT threads participating in the transaction.

Note that a completed transaction cannot be backed out. For more information about transaction backout, see the *Rocket Model 204 Host Language Interface Programming Guide*.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE  PIC 9(5)  COMP SYNC.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL “IFBOUT” USING RETCODE.
```

IFBREC call *-sc*

Function The IFBREC call (BEGIN RECORD) creates a new record and adds it to the specified file or to the file that is current.

Full syntax (20) `IFBREC(RETCODE,KEY_SPEC,FILE_SPEC,%VARBUF,%VARSPEC)`

Compile-only syntax A compile-only form of IFBREC is not available.

Execute-only syntax An execute-only form of IFBREC is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
KEY_SPEC	[I,c,o] The key specification is an optional input parameter which specifies a key value. If the file is sorted or hashed, the record's sort key value or hash key value must be supplied, and the value is automatically placed in the record by IFBREC. If the file is not sorted or hashed, a null value must be supplied to avoid receiving an error message. You may specify the key value as a single %variable. If you use a %variable, records that contain floating-point or bit string keys can be stored. Model 204 processes the %VARBUF and %VARSPEC parameters before KEY_SPEC. If you specify a %variable, do not include quotation marks in the entry. If you specify a character string instead of a %variable, you must use a single quotation mark to start and end the string if the key contains a reserved word, equal sign, or parenthesis. If the file is not hashed or sorted, you must enter a semicolon or null string.

Parameter	Description
FILE_SPEC	[I,s,o] The file specification is an optional input parameter which identifies the Model 204 file that will be updated to contain the new record. The guidelines for this optional parameter are: In individual file context, the file parameter is not required. If you specify the parameter, it must contain the name of the current file. In a group context, this parameter is not required if IFBREC is to create the record in the group update file. The group update file, named in the group CREATE command, receives all new records in the group unless you specify otherwise. In a group context, if a group update file has not been defined for the group, or if IFBREC is to create the record in a file of the group other than the group update file, the file parameter must contain the name of the file in which the new record is to be placed.
	Specify the file name as a short character string. Alternatively, you can use one of the following special functions in place of a known file name string constant: $CURFILE function (the current file of the group) $UPDATE function (the group update file) See the Rocket Model 204 documentation wiki for information about the $file functions: http://m204wiki.rocketsoftware.com/index.php/$Curfile http://m204wiki.rocketsoftware.com/index.php/$Update
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation#Declaring_.25variables_and_.25variable_arrays
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %VARBUF parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFBREC call to create and store a record. The new record becomes the current record. Use the IFPUT call after IFBREC to fill in the data fields in the record. See the IFPUT call.

When FOPT=X'10' and the date/time stamp feature is installed, the IFBREC function is *not* supported for DTS files.

Completion return code (RETCODE)

If the IFBREC call is unsuccessful, Model 204 returns one of the following completion return codes:

Code	Error condition
4	The sort or hash key is a preallocated field that has a LENGTH attribute and the value specified in IFBREC is null or too long.
10	Model 204 encountered invalid data values for BINARY and FLOAT numeric field for a file having FILEMODL set to NUMERIC VALIDATION.
200	A uniqueness violation (field level constraint) has occurred.

Note: The current record number is not changed if an error occurs. If an error occurs, avoid IFPUT calls that are aimed at building the new record.

See the Rocket Model 204 documentation wiki for information about BINARY and FLOAT field values:

http://m204wiki.rocketsoftware.com/index.php/Data_maintenance#Storing_data_in_fields

Coding example (PL/1)

The PL/1 coding example below is for a sorted file, and the key is a required entry.

```
KEY='JONES,JACK';
CALL IFBREC (ERROR, KEY);
```

Coding examples (COBOL)

The COBOL coding example below is for an unsorted file.

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5)  COMP SYNC.
    05  NULL     PIC X(1) VALUE ';'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFBREC" USING RETCODE, NULL.
```

The COBOL example below is for an unsorted file in a group. In this example, FILE-NAM is nine characters long, which enables you to store the maximum file name size. Blanks following the file name or the semicolon are ignored.

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE   PIC 9(5) COMP SYNC.
    05  NULL      PIC X(1) VALUE';'.
    05  FILE-NAM  PIC x(9) VALUE'ACCTPAY;'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFBREC" USING RETCODE, NULL, FILE-NAM.
```

The COBOL example below is for the current file in the group.

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE   PIC 9(5) COMP SYNC.
    05  NULL      PIC X(1) VALUE';'.
    05  FILE-NAM  PIC x(9) VALUE'$CURFILE;'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFBREC" USING RETCODE, NULL, FILE-NAM.
```

IFCALL call *-mc,sc*

Function

The IFCALL call (CALL) issues a call to a Host Language Interface function by number.

Full syntax

```
IFCALL(FUNC_NO,RETCODE,PARM_LIST)
```

Compile-only syntax

A compile-only form of IFCALL is not available.

Execute-only syntax

An execute-only form of IFCALL is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
FUNC_NO	[I,i,r] The function number is the required first parameter (input) which specifies the preassigned number of a particular HLI call to be run. Specify the function number as an integer value using one of the values listed on page 94. You can specify one number per call.
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. Model 204 returns a completion code for the call whose function number is specified. The code is a binary integer value.
PARM_LIST	[I,c,r] The parameter list is a required input parameter which specifies the parameter(s) that is(are) required by the call (excluding return code) whose function number is specified. Specify a character string using the following format: `parm1 [,parm2•••];` See the parameter description, excluding the return code, for the call whose number is specified.

Notes and tips

The IFCALL call is not valid on an IFAM1 type of IFSTRT thread, or on an IFDIAL thread. The form and action of the IFCALL call are no different on a multiple cursor or single cursor IFSTRT thread.

You can use IFCALL to call by number any Host Language Interface call that is assigned a number in Table 6-1, below. Note that IFCALL does not have its own function number. There are also no function numbers for IFSTRT (in IFAM1) and IFLOG.

Function numbers are only available for use with IFCALL. IFCALL provides an alternative for coding calls where the host language limits the name variable length.

Table 6-1. Host Language Interface function numbers

Number-Call Name	Number-Call Name	Number-Call Name
0 - reserved	47 - IFFINDE | IFFDE	94 - reserved
1 - IFSTRT (in IFAM2 and IFAM4)	48 - IFGETC	95 - IFOCUR
2 - IFSTHRD | IFSTRD	49 - IFGETE	96 - IFCSA
3 - IFFNSH	50 - IFGETXE | IFGTXE	97 - IFABXIT
4 - IFSTRTN | IFSTRN	51 - IFMOREC | IFMREC	98 - IFFTCH
5 - reserved	52 - IFMOREE | IFMREE	99 - IFFTCHC | IFFCHC
6 - IFDIAL	53 - IFMORXE | IFMRXE	100 - IFFTCHE | IFFCHE
7 - IFDIALN | IFDILN	54 - IFPUTC	101 - reserved
8 - IFHNGUP | IFHNGP	55 - IFPUTE	102 - reserved
9 - IFREAD	56 - IFFNDX | IFFD	103 - IFRELR
10 - IFWRITE | IFWRIT	57 - IFFNDXC | IFFDXC	104 - IFRELA
11 - IFOPEN	58 - IFFNDXE | IFFDXE	105 - IFCMTR
12 - IFCLOSE | IFCLOS	59 - IFRFLD	106 - IFOCURC | IFOCRC
13 - IFFIND | IFFD	60 - IFDISP	107 - IFOCURE | IFOCRE
14 - IFCOUNT | IFCNT	61 - IFFLS	108 - IFCLST
15 - IFGET	62 - IFDELF	109 - IFPROLS | IFPRLS
16 - IFMORE	63 - IFNFLD	110 - IFRRFLS | IFRFLS
17 - IFLIST	64 - IFUTBL	111 - reserved
18 - IFPUT	65 - IFCTO	112 - IFSTOR
19 - IFDREC	66 - IFCTOC	113 - IFSTRC
20 - IFBREC	67 - IFCTOE	114 - IFSTRE
21 - IFDSET	68 - IFSORT	115 - IFUPDT
22 - IFFILE	69 - IFSRTC	116 - IFUPDTC | IFUPDC
23 - IFINIT	70 - IFSRTE	117 - IFUPDTE | IFUPDE
24 - IFDFLD	71 - IFBOUT	118 - IFFRN
25 - IFEPRM	72 - IFCMMT	119 - IFFRNC

Table 6-1. Host Language Interface function numbers (Continued)

Number-Call Name	Number-Call Name	Number-Call Name
26 - IFSPRM	73 - IFDALL	120 - IFFRNE
27 - IFRPRM	74 - IFFDV	121 - IFRNUM
28 - IFERR	75 - IFFDVC	122 - IFOCC
29 - IFGERR	76 - IFFDVE	123 - IFOCCC
30 - IFCHKPT | IFCHKP	77 - IFGETV	124 - IFOCCE
31 - reserved	78 - IFGTVC	125 - IFCCUR
32 - IFDVAL	79 - IFGTVE	126 - IFFAC
33 - IFPROL	80 - IFSRTV	127 - IFFACC
34 - IFRRFL	81 - IFSTVC	128 - IFFACE
35 - IFOPENX | IFOPNX	82 - IFSTVE	129 - reserved
36 - IFGETX	83 - IFSKEY	130 - reserved
37 - IFMOREX | IFMREX	84 - IFSKYC	131 - reserved
38 - IFDEQ	85 - IFSKYE	132 - reserved
39 - IFENQ	86 - IFSETUP | IFSETP	133 - reserved
40 - IFENQL	87 - IFFWOL | IFFWO	134 - reserved
41 - IFDEQL	88 - IFFWOLC | IFFWOC	135 - reserved
42 - IFDTHRD | IFDTRD	89 - IFFWOLE | IFFWOE	136 - reserved
43 - IFATTN	90 - reserved	137 - reserved
44 - IFPOINT | IFPNT	91 - reserved	138 - IFERLC
45 - IFFLUSH | IFFLSH	92 - reserved	139 - IFEFCC
46 - IFFINDC | IFFDC	93 - reserved	

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05  FUNC-NUM  PIC 9(5) VALUE 14.
    05  RETCODE   PIC 9(5) COMP SYNC.
    05  COUNT     PIC 9(5).
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL “IFCALL” USING FUNC-NUM, RETCODE, COUNT.
```

In this example, IFCALL calls function number 14, IFCOUNT, which requires a single parameter for the output parameter, called COUNT.

IFCCUR call *-mc*

Function

The IFCCUR call (CLOSE CURSOR) closes the named cursor on the current thread.

Full syntax (125)

```
IFCCUR(RETCODE,CURSOR_NAME)
```

Compile-only syntax

A compile-only form of IFCCUR is not available.

Execute-only syntax

An execute-only form of IFCCUR is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
CURSOR_NAME	[I,s,r] Is a required input parameter which specifies the name of the cursor to be closed. This is a short character string (maximum 32 characters), the name previously assigned to the cursor in a corresponding IFOCUR call. See CURSOR_NAME on page 226 for a description of the cursor name for the IFOCUR call.

Notes and tips

Use the IFCCUR call to close a cursor that is open on a thread. The IFCCUR call indicates that processing against the cursor is complete.

You can reference a cursor again after it has been closed by reopening it. For example, before referencing a closed cursor in a subsequent IFFTCH statement, reopen it by issuing an IFOCUR call.

Note that when you reopen a cursor, Model 204 locates the cursor one position before the first record or value in the found set.

The IFCCUR call provides companion functionality to the IFOCUR call in the host language multiple cursor environment. See the IFOCUR call.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE      PIC 9(5) COMP SYNC.
    •
    •
    •
    05  CURSOR-NAME  PIC X(7) VALUE “CRFORD;”.
•
•
```

```
•
PROCEDURE DIVISION.
•
•
•
  CALL “IFCCUR” USING RETCODE, CURSOR-NAME.
```

Note: In the example, IFCCUR closes a cursor (named CRFORD) that was opened and named by a previous IFOCUR call (not shown). See the coding example for IFOCUR on page 226. An IFCCUR call can only be used to close a cursor that is open.

IFCHKPT call *-mc,sc*

Function

The IFCHKPT call (CHECKPOINT) accesses the Model 204 checkpoint facility and performs the specified function. The following functions can be performed using IFCHKPT:

- Query checkpointing status
- Indicate that a checkpoint can take place
- Initiate an attempt to take a checkpoint
- Wait for a checkpoint to complete or time out

Full syntax (30)

```
IFCHKPT|IFCHKP(RETCODE,FUNC_CODE,CHK_ID)
```

Compile-only syntax

A compile-only form of IFCHKPT is not available.

Execute-only syntax

An execute-only form of IFCHKPT is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value. See the completion codes in Table 6-2 on page 102.

Parameter	Description
FUNC_CODE	[I,i,r] The function code is a required input parameter which specifies the code number for the checkpointing function to be performed. Specify the code number as an integer value using one of the codes listed below. You can specify one code (0–3) per call. *0*— Samples checkpoint status. (Does not initiate checkpoint Model 204 returns the status in the return code output parameter. *1*— Quiesces the thread, initiates an attempt to take a checkpoint, and returns immediately. In a multithreaded transaction, this code is valid only for single cursor IFSTRT threads with update indicator 1 that are participating in the transaction. This function code is not valid for a multiple cursor IFSTRT thread. *2*— Quiesces the thread, indicates that a checkpoint can be taken, and waits for a checkpoint to occur or time out. (Does not initiate an attempt to take a checkpoint.) In a multithreaded transaction, this code is valid only if all the other single cursor updating IFSTRT threads that are participating in the transaction have called IFCHKPT with function code 1. In a multithreaded transaction, this code is valid only if all the other single cursor updating IFSTRT threads that are participating in the transaction have called IFCHKPT with function code 1.
	See “Notes and Tips” below for information about quiescing threads.
CHK_ID	[O,c,r/o] The checkpoint ID is an output parameter that is required only for a corresponding function code of 1, 2, or 3 in the IFCHKPT call. The ID identifies the checkpoint by date and time on return from the IFCHKPT call. Model 204 returns an eight-byte packed value in the following format: YYYYDDDttttttt where: *YYYYDDD* is the year and day (in Julian date format) that the checkpoint is taken or attempted. *ttttttt* is the exact time that the checkpoint is taken or attempted.

Notes and tips

The IFCHKPT call can be used to take checkpoints that are used for recovery on IFSTRT threads from within an IFAM2 or IFAM4 job.

IFCHKPT automatically closes all files or groups that are open on the thread.

There are other differences in IFCHKPT processing between single cursor IFSTRT update threads and multiple cursor IFSTRT threads. On a multiple

cursor IFSTRT thread, a transaction is activated only when, and if, updating calls are executed. This allows checkpoints to be taken using IFCHKPT between update units while the host language program is still in execution.

See the *Rocket Model 204 Host Language Interface Programming Guide* for more information about checkpoints and transaction management in the HLI processing environment.

Checkpointing for a multithreaded job

For a job that uses only multiple cursor IFSTRT threads, follow the steps below:

1. Call IFCMMT on all but the last thread with update units active.
2. Call IFCHKPT with function code 3 on the last updating thread.

   Note: Typically, a job would only have one multiple cursor IFSTRT thread which would eliminate step 1 above.

For a job that mixes both single cursor and multiple cursor IFSTRT threads, follow the steps below:

1. Call IFCMMT on all multiple cursor IFSTRT threads with update units active.
2. Follow guidelines for checkpointing on single cursor IFSTRT threads which are described on the next page.

   Note: Rocket does not recommend that you mix multiple cursor and single cursor IFSTRT threads in an HLI application.

Checkpointing on single cursor IFSTRT threads

Once a host language job issues a single cursor IFSTRT (or IFSTRTN) call with the update indicator set to 1, Model 204 considers an update unit to be in progress and checkpoint requests from other sources time out.

With a multithreaded transaction, each single cursor IFSTRT update thread in the job must indicate to Model 204 that it is quiescing in preparation for an attempt to take a checkpoint.

Using single cursor IFSTRT update threads, in order to enable a checkpoint request to be completed, the host language job must perform one of the following:

- Issue an IFFNSH, which effectively resets the update indicator, but also terminates the connection to Model 204.
- Call IFCHKPT with function code 1, 2, or 3. With multiple threads, call IFCHKPT following these steps:

  a. Issue IFCHKPT with function code 1 on all but the last single cursor IFSTRT updating thread.

b. Issue IFCHKPT with function code 2 or 3 on the last single cursor IFSTRT updating thread. See the following sections.

IFCHKPT on last updating thread with function code 2

If the last updating thread issues IFCHKPT with function code 2, the thread waits until the next attempt to take a checkpoint. It does not start an attempt to take a checkpoint. It is impossible to predict how long the thread waits.

Note: The IFCHKPT function code 1 also initiates an attempt to take a checkpoint on a single cursor IFSTRT updating thread. If a job has more than one single cursor IFSTRT thread, and issues IFCHKPT with function code 1 on all but the last thread, the IFCHKPT function code 2 on the last thread may wait on the checkpoint attempt that was started by the first IFCHKPT call.

Whether or not this is the case depends on the setting of the CPTQ User 0 parameter and the amount of time between the first (function code 1) IFCHKPT call and the last (function code 2) IFCHKPT call.

IFCHKPT on last updating thread with function code 3

If the last updating thread issues IFCHKPT with function code 3, an attempt to take a checkpoint is initiated. The thread waits for the checkpoint to complete or time out.

Note: The maximum time that the thread may wait is approximately CPTO + CPTQ. CPTO and CPTQ are Model 204 User 0 parameters which control the number of seconds to wait for threads to quiesce before timing out a checkpoint.

See the *Rocket Model 204 Host Language Interface Programming Guide* for a information about CPTO and CPTQ parameters in the HLI processing environment.

Completion return code (RETCODE)

Table 6-2 lists the Model 204 completion codes (RETCODE) for the IFCHKPT call. Each completion code corresponds to a particular function code (FUNC_CODE).

Table 6-2. IFCHKPT completion return codes (RETCODE)

Return code	Function code	Completion condition
0	0	Checkpoint is not currently in progress. CHK_ID set to ID of last successful checkpoint.
1	0	Checkpoint is in progress.
	1	Checkpoint is started; CHK_ID set.
	2,3	Checkpoint completed successfully; CHK_ID set.

Table 6-2. IFCHKPT completion return codes (RETCODE) (Continued)

Return code	Function code	Completion condition
2	2, 3	Checkpoint timed out or was cancelled. Checkpoint has not been taken, CHK_ID is set to the time the checkpoint attempt was started.
3	2,3	Caller is not allowed to wait for checkpoints, and call is ignored. Indicates either that the caller is online (CICS) or that the caller's thread belongs to a batch job with other updating threads that have not called IFCHKPT.
4	1	Call has been issued on a multiple cursor IFSTRT thread. IFCHKPT with a function code 1 is not allowed on a multiple cursor IFSTRT thread.
	all	Checkpointing is not active, and call is ignored. Either module CHKP is not linked with the Host Language Interface/Model 204 load module being used, or the CHKPOINT data set definition statement is absent. A return code of 4 is also returned for invalid function codes.
5	all	Roll forward recovery is running. Checkpointing is not active and the call is ignored.

Sub-transaction checkpoints and IFAM

Sub-transaction checkpoints are transparent to IFAM users. You can implement transaction-only or transaction and sub-transaction checkpoints in IFAM4, as for an Online. You may use them to recover files in active update by IFAM threads at the time of the sub-transaction checkpoint.

Continuing as transaction checkpoints

Checkpoints that accompany the following messages continue to be transaction checkpoints:

```
M204.0962 'SIGN ON, JOB NAME = '

M204.0963 'SIGN OFF. JOB NAME = '
```

Checkpoints requested and/or waited for by IFAM threads issuing various IFCHKPT function codes also continue to be transaction checkpoints. And, checkpoint status queried by IFCHKPT function code 0, refers only to the status of the last transaction checkpoint attempted, whether an intervening sub-transaction checkpoint has occurred or not.

An IFCHKPT call issues only transaction checkpoints. Thus to use sub-transaction checkpoints an IFCHKPT call is not necessary. If not IFCHKPT calls are made, IFAM updates are recoverable with sub-transaction checkpoints.

Coding example (COBOL)

The following example is for checkpointing on a multiple cursor IFSTRT thread (in an IFAM2 or IFAM4 job). A function code of 3 initiates an attempt to take a checkpoint and waits for it to complete or time out.

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE   PIC 9(5) COMP SYNC.
    05  FUNCTION  PIC 9(5) VALUE 3.
    05  CHKPTID   PIC X(8).
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL “IFCHKPT” USING RETCODE, FUNCTION, CHKPTID.
```

IFCLOSE call *-mc,sc*

Function

The IFCLOSE call (CLOSE FILE) closes a Model 204 file or group that is open, or closes all the Model 204 files or groups that are open on the thread.

Full syntax (12)

```
IFCLOSE|IFCLOS(RETCODE,FILE_SPEC)
```

Compile-only syntax

A compile-only form of IFCLOSE is not available.

Execute-only syntax

An execute-only form of IFCLOSE is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FILE_SPEC	[I,s,o] The file specification is an optional input parameter for use only with a multiple cursor IFSTRT thread for specifying the name of a particular Model 204 file or group to be closed. Specify the name as a short character string. If a file or group that is not open is specified, the call is unsuccessful and Model 204 returns a completion code equal to 4. If the file specification parameter is omitted, IFCLOSE closes all files and groups that are open on the thread.

Notes and tips

Use the IFCLOSE call to close files or groups that are open on a thread. IFCLOSE indicates that processing against the file or group for the current request is complete.

The IFCLOSE call is valid on *all* types of IFSTRT threads. On a single cursor IFSTRT thread, IFCLOSE closes all files or groups that are open. The action is the same on a multiple cursor IFSTRT thread, unless you specify a particular file or group to be closed.

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.

Note that IFCLOSE causes all saved compilations to be discarded.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE   PIC 9(5) COMP SYNC.
    05  FILESPEC  PIC X(5) VALUE “CARS;”.
```

```
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL “IFCLOSE” USING RETCODE, FILESPEC.
```

Note: This example illustrates the use of the file specification parameter (FILESPEC) which is available for use only with a multiple cursor IFSTRT thread.

IFCLST call *-mc*

Function

The IFCLST call (CLEAR LIST) performs a clear list function for a specified list on the current thread.

Full syntax (108)

```
IFCLST(RETCODE,LIST_SPEC,%VARBUF,%VARSPEC)
```

Compile-only syntax

A compile-only form of IFCLST is not available.

Execute-only syntax

An execute-only form of IFCLST is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LIST_SPEC	[I,c,r] The list specification is a required input parameter which specifies the name of a list. Specify the list as a character string using the following format: `LIST listname [{IN FILE filename | GROUP groupname}]` where: *listname* is required and specifies the name of a particular list; *IN* clause is optional and specifies a file or group context other than the default. *filename* specifies the name of a particular file context for the list. *groupname* specifies the name of a particular group context for the list. **Note:** If the specified list exists, Model 204 removes all records from it. If the specified list does not exist and the IN FILE/GROUP clause is not coded, Model 204 creates and initializes a list within the context of the default file or group on the thread.

Parameter	Description
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for more information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation#Declaring_.25variables_and_.25variable_arrays
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFCLST call to remove records from an existing list, or to create a new cleared list.

The IFCLST call is the equivalent of the CLEAR LIST statement in SOUL in the host language multiple cursor environment. See the Rocket Model 204 documentation wiki for information about the CLEAR LIST statement:

http://m204wiki.rocketsoftware.com/index.php/Lists#Clearing_a_list

The %VARBUF and %VARSPEC parameters can be used to pass a value for the MEMBER %variable when using the following syntax:

```
IN GROUP groupname MEMBER %variable
```

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE   PIC 9(5)  COMP SYNC.
    05  LISTSPEC  PIC X(23) VALUE "LIST FORD IN FILE CARS;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL “IFCLST” USING RETCODE, LISTSPEC.
```

IFCMMT call *-mc,sc*

Function

The IFCMMT call (COMMIT) commits and ends the current update unit for the active thread or threads.

Full syntax (72)

```
IFCMMT(RETCODE)
```

Compile-only syntax

A compile-only form of IFCMMT is not available.

Execute-only syntax

An execute-only form of IFCMMT is not available.

Parameters

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required output parameter. The code is a binary integer value.

Notes and tips

Use the IFCMMT call to end an update unit. IFCMMT applies to the active threads in a host language job. You can issue the IFCMMT call on the following types of threads:

- Single cursor IFSTRT update thread (thread type is 1)
- Multiple cursor IFSTRT thread

Note: If an IFCMMT call is encountered on a read-only single cursor IFSTRT thread, Model 204 returns an error completion code of 40.

Record locking behavior

IFCMMT releases the lock pending updates (LPU) exclusive lock on updated records and the single record enqueue (SRE) on the current record (if one exists) that were obtained by this thread. The transaction is ended for all participating threads and the backout and constraints logs are freed.

Found sets are not released and the current record does not change. In a multithreaded transaction the LPU and SRE locks obtained by other threads are still held. Also, you must issue IFCMMT on all of the single cursor IFSTRT threads participating in a multi-transaction if you issue the call on any one of the threads.

For more information about transaction management, see the *Rocket Model 204 Host Language Interface Programming Guide*.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE  PIC 9(5) COMP SYNC.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL “IFCMMT” USING RETCODE.
```

IFCMTR call *-mc*

Function

The IFCMTR call (COMMIT RELEASE) releases all record sets held by the thread and commits the current transaction.

Full syntax (105)

```
IFCMTR(RETCODE)
```

Compile-only syntax

A compile-only form of IFCMTR is not available.

Execute-only syntax

An execute-only form of IFCMTR is not available.

Parameters

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the only required parameter. The code is a binary integer value.

Notes and tips

Use the IFCMTR call to end the current transaction and release all records. The IFCMTR call is the equivalent of the SOUL COMMIT RELEASE statement in the host language multiple cursor environment. See the Rocket Model 204 documentation wiki for information about the COMMIT RELEASE statement:

http://m204wiki.rocketsoftware.com/index.php/Files,_groups,_and_reference_context#Using_COMMIT_and_COMMIT_RELEASE_statements

To ensure that all of the records and resource locks associated with a transaction are released after the IFCMTR call executes on one of the single cursor IFSTRT threads in a multithreaded transaction, issue IFCMTR on all of the other single cursor IFSTRT threads participating in the transaction.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5) COMP SYNC.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL “IFCMTR” USING RETCODE.
```

IFCOUNT call *-mc,sc*

Function

The IFCOUNT call (COUNT) counts the number of records in a found set or list and returns the record count in an output parameter.

Full syntax (14)

```
IFCOUNT|IFCNT(RETCODE,COUNT,SET_QUAL)
```

Compile-only syntax

A compile-only form of IFCOUNT is not available.

Execute-only syntax

An execute-only form of IFCOUNT is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
COUNT	[O,i,r] The count parameter is a required parameter which specifies the location of the output parameter for the return count value. Specify an integer variable. Model 204 returns the record count as a fullword binary number.
SET_QUAL	[I,c,r] The set qualifier is available only for use with a multiple cursor IFSTRT thread and it is required for specifying the record set or list whose records will be counted. Specify a character string using either one of the following formats: `{IN label | ON [LIST] listname}` where: *label* is the name of a saved IFFIND, IFFNDX, IFFWOL, or IFFAC compilation from the previously compiled call which established the record set. *listname* specifies the name of a list. **Note:** The set qualifier is not a valid parameter for use with a single cursor IFSTRT thread.

Notes and tips

Use the IFCOUNT call to count the records in a found set. IFCOUNT returns a value of 0 for a null set. You can use IFCOUNT after an IFFIND call having a completion code of 0 to check for a null set.

The IFCOUNT call is valid on *all* types of IFSTRT threads. You must specify the found set to be counted on a multiple cursor IFSTRT thread. On a single cursor IFSTRT thread, IFCOUNT returns the count for the set that is current.

Note that the count that is returned by the IFCOUNT call is written to the Model 204 audit trail if RK lines are written to the journal. See the Rocket

Model 204 documentation wiki for information about Model 204 journals and audit trails:

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

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL  COMP SYNC.
    05  RETCODE    PIC 9(5).
    05  COUNT      PIC 9(5).
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL “IFCOUNT” USING RETCODE, COUNT.
```

IFCSA call *-mc,sc*

Function

The IFCSA call (Common System Area) passes the address of the Common System Area to the CICS interface in IFAM2.

Full syntax (96)

```
IFCSA(CSA_ADDR)
```

Compile-only syntax

A compile-only form of IFCSA is not available.

Execute-only syntax

An execute-only form of IFCSA is not available.

Parameters

Parameter	Description
CSA_ADDR	[I,i,r] The CSA (Common System Area) address is a required input parameter for the CICS interface. Specify the address as an integer value.

Notes and tips

Use the IFCSA call only for an IFAM2 host language job with CICS. IFCSA is valid for use Model 204 releases before Version 3.1. For Model 204 Version 3.1 and later, the CSA control block is no longer needed and therefore, the IFCSA call is not required. For purposes of upward compatibility, the IFCSA call remains; however, it no longer performs any function.

Coding examples

Before a program issues any operational Host Language Interface calls, the address of the CSA must be passed to the interface by using one of the following special calls in the host language program:

COBOL:	CALL 'IFCSA' USING DFHCSADS
PL/1 Optimizer:	CALL IFCSA (DFHCSADS)
Assembler:	CALL IFCSA, ((13)),VL

where the address is declared as:

```
DCL IFCSA ENTRY OPTIONS (ASM ENTER):
```

IFCTO call -SC

Function

The IFCTO call (COUNT OCCURRENCES) counts the number of occurrences of the specified field, or fields, in the current record and returns a count value for each field in an output parameter.

Full syntax (65)

```
IFCTO(RETCODE,BUFFER,FIELD_LIST,CTO_NAME,%VARBUF,%VARSPEC)
```

Compile-only syntax (66)

```
IFCTOC(RETCODE,FIELD_LIST,CTO_NAME)
```

Execute-only syntax (67)

```
IFCTOE(RETCODE,BUFFER,CTO_NAME,%VARBUF,%VARSPEC)
```

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
BUFFER	[O,c,r] The buffer location is a required output parameter which specifies the address of the user's data area. Specify a character string. The buffer contains the occurrence counter, or counters, returned by IFCTO for each field that is defined by the FIELD_LIST parameter, described below. For each field specified, a four-byte value is returned. Multiple count values are positioned in order corresponding to the location of the fields in the current record. If a specified field does not occur in the record, its counter is set to zero.
FIELD_LIST	[I,c,r] The field specification is a required input parameter which defines the field, or fields, that are to be counted in the current record. Specify the name of a field as a character string. You must list at least one field to be counted in the current record. You may list additional fields by separating field names with a comma.
CTO_NAME	[I,s,o/r] The name of the IFCTO compilation is a required input parameter with IFCTOC and optional with IFCTO or IFCTOE. If specified, Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). Any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. The first character in the name must be alphanumeric. A null value is equivalent to omitting the name parameter.

Parameter	Description
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation#Declaring_.25variables_and_.25variable_arrays
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFCTO call to count occurrences of fields in the current record. Issue the IFCTO call after a current record is selected by IFGET, IFPOINT, or IFBREC. The record is enqueued with share status.

IFCTO is available for use only on a single cursor IFSTRT thread. IFCTO operates similarly to the IFOCC call that is used with multiple cursor IFSTRT. See the IFOCC call.

The IFCTO call is the equivalent of the SOUL COUNT OCCURRENCES OF statement in the host language environment. See the Rocket Model 204 documentation wiki for information about the COUNT OCCURRENCES OF statement:

http://m204wiki.rocketsoftware.com/index.php/Processing_multiply_occurring_fields_and_field_groups#COUNT_OCCURRENCES_OF_statement

Coding example (PL/1)

```
DCLERROR FIXED BIN (31),
BUFFER CHAR (8)
FIELDS CHAR (20) VARYING INITIAL ('CAR,CHILD NAME');
•
•
•
CALL IFCTO (ERROR,BUFFER,FIELDS)
```

Note: After the IFCTO call shown above is issued to count the number of field occurrences in a record that contains two cars and five children, BUFFER contains the following hexadecimal value:

```
0000000200000005
```

IFDALL call *-mc,sc*

Function

The IFDALL call (DELETE ALL) deletes all occurrences of a specified Model 204 field from the current record.

Full syntax (73)

```
IFDALL(RETCODE,FIELD_NAME,CURSOR_NAME)
```

Compile-only syntax

A compile-only form of IFDALL is not available.

Execute-only syntax

An execute-only form of IFDALL is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FIELD_NAME	[I,c,r] The field name is a required input parameter which specifies the name of the Model 204 field to be deleted. You can specify only one field to be deleted per call. Specify a character string variable.
CURSOR_NAME	[I,s,r] The name of the cursor is an input parameter that is available only for use with a multiple cursor IFSTRT thread and is required for specifying the current record from which the field will be deleted. Specify the cursor name as a short character string, using the name previously assigned to the cursor in a corresponding IFOCUR call. See CURSOR_NAME on page 226 for a description of the cursor name for the IFOCUR call. **Note:** The cursor name is not a valid parameter for use with a single cursor IFSTRT thread.

Notes and tips

You can issue the IFDALL call only when a current record exists. The IFDALL call is valid on *all* types of IFSTRT threads. On a single cursor IFSTRT thread, issue the IFDALL call for the current record after using an IFBREC, IFPOINT, or IFGET call.

When FOPT=X'10' and the date/time stamp feature is installed, the IFDALL function is *not* supported for DTS files.

On a multiple cursor IFSTRT thread, you must specify the cursor for the current record. The cursor that is specified must have a current record from a previous IFSTOR, IFFRN, or IFFTCH call.

Use the IFDALL call to delete a Model 204 field except for a field that has been defined as having one of the following attributes:

- INVISIBLE attribute
- Sort key
- Hash key

Note: You cannot use IFDALL on a sorted record set.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE            PIC 9(5) COMP SYNC.
    05  DELETE-FIELD-NAME  PIC X(4) VALUE "VIN;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFDALL" USING RETCODE, DELETE-FIELD-NAME.
```

IFDECL call

Function

The IFDECL call lets you declare STRING %variables to support the use and display of double-byte character set (DBCS) characters.

Full syntax

```
IFDECL %variable [IS] [EBCDIC | MIXED {DBCS | KANJI} | DBCS
| KANJI] [options]
```

Parameters

The parameters for IFDECL are the same as for the Model 204 DECLARE statement.

Notes and tips

To support the use and display of Kanji characters, which require two bytes each, the IFDECL call declares STRING %variables as one of three types for use in an IFFIND:

- **STRING EBCDIC**—the default type, contains single byte characters, in the EBCDIC collating sequence
- **STRING DBCS**—can contain only pure DBCS characters (double-byte data only, with no shift sequences)
- **STRING MIXED DBCS**—Model 204 assumes the field contains both DBCS and EBCDIC data and that all DBCS characters are contained within balanced shift sequence pairs. (The shift sequences define whether a series of bytes is interpreted as DBCS or EBCDIC.)

When you use a pure or mixed DBCS %variable in an IFFIND statement, Model 204:

1. Performs the appropriate data type conversions, following the conversion rules for assignment types.
1. Compares the pure and mixed DBCS fields.

IFDECL declares only simple string variables. You cannot use IFDECL for arrays, ASCII, FLOAT, or BINARY strings, or for lists, labels, or subroutines.

IFDECL is allowed only on Multi-Cursor IFAM threads.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01 CALL-ARGS.
   77 DECLARE-MIXED PIC X(35) VALUE ‘%MIXED IS STRING MIXED DBCS LEN 20;’.
   77 DECLARE-PURE  PIC X(28) VALUE ‘%PURE IS STRING DBCS LEN 20;’.
   77 FIND-DBCS     PIC X(31) VALUE ‘PURE.DBCS = ’Kanji-data’;END;’.
   77 FIND-NAME     PIC X(08) VALUE ‘FIND.DBCS;’
   •
   •
```

```
   •
PROCEDURE DIVISION.
   •
   •
   •
CALL IFDECL WITH DECLARE-MIXED.
CALL IFDECL WITH DECLARE-PURE.
CALL IFFIND WITH RETCODE, FIND-DBCS, FIND
```

IFDELF call *-mc,sc*

Function

The IFDELF call (DELETE FIELD) deletes a Model 204 field definition and all occurrences of data for that field. IFDELF requires Model 204 file manager privileges.

Full syntax (62)

```
IFDELF(RETCODE,FIELD_NAME,FILE_SPEC)
```

Compile-only syntax

A compile-only form of IFDELF is not available.

Execute-only syntax

An execute-only form of IFDELF is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FIELD_NAME	[I,c,r] The field name is a required input parameter which specifies the name of the Model 204 field to be deleted. Specify a character string variable.
FILE_SPEC	[I,s,o] The file specification is an optional input parameter for use only with a multiple cursor IFSTRT thread for specifying the name of the Model 204 file containing the field that is to be deleted. Specify the name of the file as a short character string using the following format: `IN [FILE] filename` The specified file must be open on the thread, otherwise the call is unsuccessful and Model 204 returns a completion code of 4.

Notes and tips

The IFDELF call is valid on *all* types of IFSTRT threads. Use the IFDELF call to delete field definitions *except* for the following types of fields:

- Record security is defined using the field
- The field is defined as a sort key
- The field is used as a hash key

Note: The file context can change on a multiple cursor thread and, if the file specification parameter (FILE_SPEC) is omitted, IFDELF deletes the field definition for the default file on the thread (that is, the last file opened).

When FOPT=X’10’ and the date/time stamp feature is installed, the IFDELF function is supported for DTS files.

The IFDELF call requires Model 204 file manager privileges. Additional privileges are required for the following security conditions:

Type of security	Required privileges
File defined with record security	Record security override
Security level field definition	Field level security (FLS) allows update access

For more information about Model 204 field security, see the Rocket Model 204 documentation wiki page on Security:

http://m204wiki.rocketsoftware.com/index.php/Security#Field-level_security

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5) COMP SYNC.
    05  STATUS   PIC X(7) VALUE “STATUS;”.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL “IFDELF” USING RETCODE, STATUS.
```

IFDEQ call *-mc,sc*

Function

The IFDEQ call (DEQUEUE) dequeues a resource previously specified in a call to IFENQ.

Full syntax (38)

```
IFDEQ(RETCODE,RESOURCE)
```

Compile-only syntax

A compile-only form of IFDEQ is not available.

Execute-only syntax

An execute-only form of IFDEQ is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	{O,i,r]The Model 204 return code is the required first parameter. The code is a binary integer value.
RESOURCE	[I,s,r] The resource is a required input parameter which specifies the name of the previously enqueued resource to be dequeued. Specify a short character string, up to 32 characters in length. You may specify any data in the input string except for 32 bytes of binary zeroes.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5) COMP SYNC.
    05  RNAME    PIC X(9) VALUE "CUSTFILE;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFDEQ" USING RETCODE, RNAME.
```

IFDEQL call -sc

Function

The IFDEQL call (DEQUEUE LIST) dequeues that set of records on the specified list.

Full syntax (41)

```
IFDEQL(RETCODE,LIST_NAME)
```

Compile-only syntax

A compile-only form of IFDEQL is not available.

Execute-only syntax

An execute-only form of IFDEQL is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LIST_NAME	[I,c,r] The list name is a required input parameter which specifies the name of a list. Specify the list as a character string.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE   PIC 9(5) COMP SYNC.
    05  LISTNAME  PIC X(9) VALUE "CARSLIST;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFDEQL" USING RETCODE, LISTNAME.
```

IFDFLD call *-mc,sc*

Function The IFDFLD call (DEFINE FIELD) defines a new field for a Model 204 file.

Full syntax (24) `IFDFLD(RETCODE,FIELD_DESC,FILE_SPEC)`

Compile-only syntax A compile-only form of IFDFLD is not available.

Execute-only syntax An execute-only form of IFDFLD is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.

Parameter	Description
FIELD_DESC	[I,c,r] The field description is a required input parameter which specifies the name of the new field and a list of its attributes. Specify the field as a character string using the following format: `fieldname [(attribute •••)];` where: *fieldname* is required and specifies the name of the new field. The name must be unique within the context of the file where it will be stored. Specify the name as a character string, up to 255 characters in length. The name must begin with a letter and it can contain any alphanumeric character except the following: • At sign (@) • Pound sign (#) • Semi-colon (;) • Double question marks (??) • Question mark followed by a dollar sign (?$) • Question mark followed by an ampersand (?&) See the Rocket Model 204 documentation wiki for the detailed list of rules that apply to naming fields: http://m204wiki.rocketsoftware.com/index.php/Field_names *attribute* is optional and specifies a particular characteristic that controls how the field is used, stored, or accessed. You may specify more than one attribute, separating each by a comma or a blank. The attributes that may be specified using the IFDFLD call are identical to those that are used with the SOUL DEFINE FIELD command. See the Rocket Model 204 documentation wiki for a description of field attributes used with the SOUL DEFINE command: http://m204wiki.rocketsoftware.com/index.php/Field_design#Field_descriptions_and_attributes **Note:** If no attributes are specified, Model 204 defines the field assigning all of the default attributes.
FILE_SPEC	[I,s,o] The file specification is an optional input parameter for use only with a multiple cursor IFSTRT thread for specifying the name of the Model 204 file that will contain the new field. Specify the name of the file as a short character string variable using the following format: `IN [FILE] filename` The specified file must be open on the thread, otherwise the call is unsuccessful and Model 204 returns a completion code equal to 4.

Notes and tips

Use the IFDFLD call to define a new field in a Model 204 file. The IFDFLD call is valid on *all* types of IFSTRT threads. You can use the IFDFLD call once the Model 204 file has been initialized (using the IFINIT call).

When FOPT=X'10' and the date/time stamp feature is installed, the IFDFLD function is supported for DTS files.

Note: The file context can change on a multiple cursor thread and, if the file specification parameter (FILE_SPEC) is omitted, IFDFLD defines the field for the default file on the thread.

Note that certain types of fields must be defined using IFINIT rather than IFDFLD. You cannot use IFDFLD to define fields when:

- Record security is defined using the field
- The field is defined as a sort key
- The field is used as a hash key

See the IFINIT call.

Note also that IFDFLD follows the same basic rules for specifying field attribute definitions as the Model 204 DEFINE FIELD command.

See the Rocket Model 204 documentation wiki for information about using the DEFINE FIELD command and about field names and attributes:

http://m204wiki.rocketsoftware.com/index.php/Field_design#Field_descriptions_and_attributes

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5) COMP SYNC.
    05  FIELDA   PIC X(20) VALUE "FIELDA(BINARY KEY);".
    05  FIELDB   PIC X(32) VALUE "FIELDB(FEW-VALUED,CODED,
        RANGE);".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFDFLD" USING RETCODE, FIELDA, FIELDB.
```

IFDIAL call *-di*

Function The IFDIAL call (DIAL) starts an IFDIAL thread which establishes a direct connection to the Model 204 SOUL facility through the default CRAM communications channel.

Full syntax (6)

```
IFDIAL(RETCODE,LANG_IND,BUFFER,IO_LEN)
```

Compile-only syntax A compile-only form of IFDIAL is not available.

Execute-only syntax An execute-only form of IFDIAL is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LANG_IND	[I,i,r] The language indicator is a required input parameter which establishes the calling sequence convention to be used corresponding to the host language. The indicator specifies the format of parameters that are passed in subsequent calls. Specify one of the following integer values: 1 = PL/1 F-level, and BAL languages 2 = COBOL, FORTRAN, and BAL languages 3 = PL/1 with +Optimizer/Checkout compilers, VS/FORTRAN, and BAL languages **Note:** Any convention may be specified for use with the BAL language, and the BAL programmer must adhere to the convention that is specified when coding parameters.
BUFFER	[O,i,o] The buffer size is an optional output parameter that specifies the size of the CRAM buffer which is the maximum buffer length for I/O to Model 204. Model 204 returns the size of the buffer as an integer value. BUFFER is required if IO_LEN is specified.
IO_LEN	[I,i,o] The I/O length is an optional input parameter which specifies the default length for I/O to Model 204. This is the length that is used if a length parameter is not specified in an IFREAD or IFWRITE call. See the LINE_LEN parameter for IFREAD on page 252.

Notes and tips Use the IFDIAL call to establish an IFDIAL connection through the default CRAM channel M204PROD.

An HLI job can start both IFDIAL and IFSTRT threads. Note, however, that any single HLI job may have only one active IFDIAL thread.

Releasing an IFDIAL thread

To release an IFDIAL thread, use either one of the following HLI functions:

- IFHNGUP, which disconnects the IFDIAL thread.
- IFFNSH, which deletes *all* outstanding threads, including the IFDIAL connection.

Note: If your job started both IFSTRT and IFDIAL threads and you use the IFDTHRD call to detach threads, IFDTHRD releases an IFSTRT thread without disconnecting the IFDIAL thread.

Using the image feature to transmit data

The image feature of SOUL allows a request to process terminal input and output, such as an IFDIAL connection. Large blocks of data can thus be transferred from a host language program to Model 204.

See the Rocket Model 204 documentation wiki for more information about READ IMAGE and WRITE IMAGE:

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

Completion return code (RETCODE)

If the IFDIAL call is unsuccessful, Model 204 returns one of the following error codes:

Code	Error condition
80	No current thread. (Action: Call IFSETUP.)
90	An IFDIAL connection already exists for this application program. The attempt to establish more than one simultaneous connection for this application is ignored.
101	Invalid module load. (Action: Check STEPLIB for correct release.)
500	Invalid function called.
800	All Model 204 IFDIAL connections are busy. (Action: Check to see that the proper number of IODEV 29 initialization statements were included in the HLI Model 204 service program.)
1001	HLI Model 204 service program is not up, the HLI is halted or drained, or no host language threads were defined in Model 204. (This may indicate that the IFAM4 load module was link-edited without the REUS option.)
1003	Not enough memory for CRAM.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE    PIC 9(5)  COMP SYNC.
    05  COBOL-IND  PIC 9(5)  VALUE 2.
•
•
•
PROCEDURE DIVISION.
  OPEN INPUT...
  CALL "IFDIAL" USING RETCODE, COBOL-IND.
•
•
•
```

Coding example (Assembler)

```
•
•
•
  CALL  IFDIAL,(RETCODE,LANGID),VL
•
•
•
RETCODE    DC  F’0’
LANGID     DC  F’2’
    END
```

IFDIALN call *-di*

Function

The IFDIALN call (DIAL) starts an IFDIAL thread which establishes a direct connection to the Model 204 SOUL facility with a specified Host Language Interface/Model 204 service program through the named channel (in IFAM2).

Full syntax (7)

```
IFDIALN|IFDILN(RETCODE,LANG_IND,CHAN,BUFFER,IO_LEN)
```

Compile-only syntax

A compile-only form of IFDIALN is not available.

Execute-only syntax

An execute-only form of IFDIALN is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LANG_IND	[I,i,r] The language indicator is a required input parameter which establishes the calling sequence convention to be used corresponding to the host language. The indicator specifies the format of parameters that are passed in subsequent calls. Specify one of the following integer values: 1 =PL/1 F-level, and BAL languages 2 =COBOL, FORTRAN, and BAL languages 3 =PL/1 with +Optimizer/Checkout compilers, VS/FORTRAN, and BAL languages **Note:** Any convention may be specified for use with the BAL language, and the BAL programmer must adhere to the convention that is specified when coding parameters.
CHAN	[I,c,r] The channel name is a required input parameter which specifies the CRAM, or IUCV/ VMCF communications channel name for a particular service program. Specify the name as an eight-character string. See “Subsystem names, channel names, and IODEV settings” on page 33 for the default channel names. **Note:** Do *not* append a semicolon. If the host language is PL/1, pass the address of the string using a based variable that overlays the original parameter.
BUFFER	[O,i,o] The buffer size is an optional output parameter that specifies the size of the CRAM buffer which is the maximum buffer length for I/O to Model 204. Model 204 returns the size of the buffer as an integer value. BUFFER is required if IO_LEN is specified.

Parameter	Description
IO_LEN	[I,i,o] The I/O length is an optional input parameter which specifies the default length for I/O to Model 204. This is the length that is used if a length parameter is not specified in an IFREAD or IFWRITE call. See the LINE_LEN parameter for IFREAD on page 252.

Notes and tips

Use the IFDIALN call to establish an IFDIAL connection using a specific Host Language Interface/Model 204 service program. IFDIALN performs the same basic function as IFDIAL. The IFDIALN call includes a parameter which is not available with IFDIAL that is used to specify the communications channel name for the service program.

Note: When an IFDIALN call is made with a language indicator of 1 or 3, Model 204 expects the channel name to be a string and not a PL/1 dope vector.

For more information about IFAM2, CRAM, and the Host Language Interface/Model 204 service program, see Chapter 3, and to the *Rocket Model 204 Host Language Interface Programming Guide*.

Completion return code (RETCODE)

If the IFDIALN call is unsuccessful, Model 204 returns one of the following error codes:

Code	Error condition
80	No current thread. (Action: Call IFSETUP.)
90	An IFDIAL connection already exists for this application program. The attempt to establish more than one simultaneous connection for this application is ignored.
101	Invalid module load. (Action: Check STEPLIB for correct release.)
500	Invalid function called.
800	All Model 204 IFDIAL connections are busy. (Action: Make sure the proper number of IODEV 29 init statements were included in the HLI Model 204 service program.)
1001	HLI Model 204 service program is not up, the HLI is halted or drained, or no host language threads were defined in Model 204. (This may indicate that the IFAM4 load module was link-edited without the REUS option.)
1003	Not enough memory for CRAM.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE    PIC 9(5)  COMP SYNC.
    05  COBOL-IND  PIC 9(5)  VALUE 2.
    05  CHAN-NAME  PIC X(8)  VALUE "M204CHNB".
•
```

```
•
•
PROCEDURE DIVISION.
  OPEN INPUT...
  CALL "IFDIALN" USING RETCODE, COBOL-IND, CHAN-NAME.
•
•
•
```

Coding example (Assembler)

```
•
•
•
  CALL  IFDIALN,(RETCODE,LANGID,CHAN0,CRAMSIZE),VL
•
•
•
RETCODE    DC  F’0’
LANGID     DC  F’2’
CHAN0      DC  C’USR204C0’
CRAMSIZE   DC  F’0’
    END
```

IFDISP call *-mc,sc*

Function The IFDISP call (DISPLAY) returns display output for the specified Model 204 field, file, group, record, or stored procedure.

Full syntax (60)
```
IFDISP(RETCODE,BUFFER,DISP_SPEC)
```

Compile-only syntax A compile-only form of IFDISP is not available.

Execute-only syntax An execute-only form of IFDISP is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value.
BUFFER	[O,c,r] The buffer location is a required output parameter which specifies the address of the user's data area. The buffer contains the information that is returned by IFDISP for the field, file, group, record, or stored procedure that is specified by the DISP_SPEC parameter, described on page 135. Specify a character string. **Note:** The format of the data area is identical to that produced by the SOUL DISPLAY command, except that Model 204 inserts semicolons, instead of terminal end-of-line characters, between lines of output. There is no limit to the length of each line. The maximum length of the display output depends on the host language. If the actual display is longer than the maximum, the display is truncated and no semicolon is appended to the truncated line. See the Rocket Model 204 documentation wiki for examples of output from the DISPLAY command: http://m204wiki.rocketsoftware.com/index.php/DISPLAY_command

Parameter	Description

DISP_SPEC	[I,c,r] The display specification is a required input parameter which specifies the entity for which information is to be displayed. Specify a character string using one of the following formats:
	`FIELD [(display option[,display option •••])]`
	`{ALL | fieldname[,fieldname...]}`
	where, if you specify ALL, the display options apply to all of the fields in the currently open file, and all the fields are listed.
	`FILE (display option[,display option •••])]`
	`{ALL | filename[,filename•••]}`
	where if you specify ALL, the display options apply to all open files, otherwise, the command applies to only those files that are listed.
	`[PERM | TEMP] GROUP`
	`[(display option[,display option •••])]`
	`{ALL | groupname[,groupname •••]}`
	where if you specify ALL, the display options apply to all existing permanent groups, otherwise the command applies to only those group that are listed.
	`RECORD [(NOUSE)]`
	You can use the record option only in file context.
	Note: Alternatively, you may specify the name of a stored procedure to be displayed; do not specify any of the display clauses (keywords FIELD, FILE, GROUP, RECORD) listed above.
	See the Rocket Model 204 documentation wiki for a list of the options and specifications for the DISPLAY command.
	http://m204wiki.rocketsoftware.com/index.php/DISPLAY_command
	These options are also available for use with the IFDISP call.

Notes and tips

Use the IFDISP call to access information about a Model 204 field, record, file, group, or stored procedure from inside a host language program. You can use the IFDISP call on any type of connection to Model 204 from a host language program.

Completion return code (RETCODE)

If the IFDISP call is unsuccessful, Model 204 returns a completion code of 4 if any one of the following error conditions occurs:

- The parameter syntax is incorrect.
- A specified field is undefined.
- The specified file or group is not currently open.

Coding example (PL/1)

The example below shows a PL/1 code excerpt which calls IFDISP to display the names of all the fields for the current file.

```
DCL IFDISP ENTRY (FIXED BIN(16), CHAR(*) VAR, CHAR(*));
DCL BUFFER CHAR(500) VARYING;


•
•
CALL IFDISP (RETCODE,BUFFER, 'FIELD (NAMES) ALL');
```

Coding example (COBOL)

The example below shows a COBOL code excerpt which calls IFDISP to display the value of the file characteristics parameters for the file named FILEX.

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE  PIC 9(5) COMP SYNC.
    05 BUFFER   PIC X(500) VALUE SPACES.
    05 DISPLAY  PIC X(20) VALUE "FILE (FPARMS) FILEX;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFDISP" USING RETCODE, BUFFER, DISPLAY.
```

IFDREC call *-mc,sc*

Function The IFDREC call (DELETE RECORD) deletes the current record from its file.

Full syntax (19) `IFDREC(RETCODE,CURSOR_NAME)`

Compile-only syntax A compile-only form of IFDREC is not available.

Execute-only syntax An execute-only form of IFDREC is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
CURSOR_NAME	[I,s,r] The name of the cursor is an input parameter that is available only for use with a multiple cursor IFSTRT thread and is required for specifying the current record to be deleted. Specify the cursor name as a short character string, using the name previously assigned to the cursor in a corresponding IFOCUR call. See CURSOR_NAME on page 226 for a description of the cursor name for the IFOCUR call. **Note:** The cursor name is not a valid parameter for use with a single cursor IFSTRT thread.

Notes and tips Use the IFDREC call to delete a record from a file. Note that you cannot use IFDREC with a sorted record set.

When FOPT=X'10' and the date/time stamp feature is installed, the IFDREC function is supported for DTS files.

IFDREC reclaims the file storage space that is occupied by the deleted record and allows the record numbers to be reused. In order to reuse record numbers from records that will be deleted, first use the IFDVAL call to eliminate any INVISIBLE fields in the record, then use IFDREC to delete the record. See the IFDVAL call.

The IFDREC call is valid on *all* types of IFSTRT threads. On a multiple cursor IFSTRT thread, you must specify the cursor name whose current record is to be deleted. On a single cursor IFSTRT thread, IFDREC deletes the current record from the current record set.

Note: There is another call, IFDSET, that also deletes records from a file and executes faster than IFDREC, but does not reclaim storage space. See the IFDSET call.

Completion return code (RETCODE)

If the IFDREC call is unsuccessful, Model 204 returns an error completion code of 4 if there is no current record.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE        PIC 9(5) COMP SYNC.
    •
    •
    •
    05 CURSOR-NAME    PIC X(7) VALUE "CRFORD;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFDREC" USING RETCODE, CURSOR-NAME.
```

Note: In the example, IFDREC deletes a field on a multiple cursor IFSTRT thread. The IFDREC call references the cursor (named CRFORD) that was opened to a found set by a previous IFOCUR call (not shown) and deletes the record that was last fetched. See the IFOCUR coding example on page 226.

IFDSET call *-mc,sc*

Function

The IFDSET call (DELETE SET) deletes the records that are in a found set from a file or group.

Full syntax (21)

```
IFDSET(RETCODE,SET_QUAL)
```

Compile-only syntax

A compile-only form of IFDSET is not available.

Execute-only syntax

An execute-only form of IFDSET is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
SET_QUAL	[I,c,r] The set qualifier is available only for use with a multiple cursor IFSTRT thread and it is required for specifying the record set or list that will be used to delete records from a file or group. Specify a character string using either one of the following formats: `{IN label | ON [LIST] listname}` where: *label* is the name of a saved IFFIND, IFFNDX, IFFWOL, or IFFAC compilation from the previously compiled call, which established the record set. *listname* specifies the name of a list. **Note:** The set qualifier is not a valid parameter for use with a single cursor IFSTRT thread.

Notes and tips

Use the IFDSET call to delete records from a file or group. Note that you cannot use IFDSET with a sorted record set.

When FOPT=X'10' and the date/time stamp feature is installed, the IFDSET function is supported for DTS files.

There is another call, IFDREC, that also deletes records from a file or group. IFDSET executes faster than IFDREC. However, with IFDSET the file storage space that is occupied by the deleted set of records is not reclaimed and the record numbers are not reused. See the IFDREC call.

The IFDSET call is valid on *all* types of IFSTRT threads. You must specify the found set for records that are to be deleted on a multiple cursor IFSTRT thread. On a single cursor IFSTRT thread, IFDSET deletes records from the current set.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE    PIC 9(5) COMP SYNC.
    05 DEL-SET    PIC X(10) VALUE "IN PREVFD;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFDSET" USING RETCODE, DEL-SET.
```

Note: The example above illustrates the use of the set qualifier parameter (DEL-SET) which is available for use only with a multiple cursor IFSTRT thread.

IFDTHRD call *-mc,sc*

Function

The IFDTHRD call (DETACH THREAD) detaches the current thread and activates the specified IFSTRT thread.

Full syntax (42)

```
IFDTHRD|IFDTRD(RETCODE,NEW_ID,OLD_ID)
```

Compile-only syntax

A compile-only form of IFDTHRD is not available.

Execute-only syntax

An execute-only form of IFDTHRD is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value.
NEW_ID	[I,i,r] The new thread identifier is a required input parameter which identifies the thread to be made current. This is the thread identifier previously assigned by the IFSTRT or IFSTRTN call which started the thread. (See the THRD_ID parameter for IFSTRT (IFAM2/IFAM4) on page 295.) Specify an integer value.
OLD_ID	[O,i,r] The old thread identifier is a required output parameter which identifies the thread that is being detached. Model 204 returns the integer value which identifies the current thread.

Notes and tips

Use the IFDTHRD call to switch from the current thread to another, while freeing the old thread. IFDTHRD entails low overhead. You cannot use IFDTHRD to detach the current thread without specifying a new thread.

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

The IFDTHRD call is useful for switching threads in a multithreaded IFAM2 or IFAM4 transaction using single cursor IFSTRT threads. IFDTHRD is not valid for use with an IFAM1 thread.

Note: IFDTHRD is valid for use only with an IFSTRT thread. If your job started both IFSTRT and IFDIAL threads and you use the IFDTHRD call to detach threads, IFDTHRD releases an IFSTRT thread without disconnecting the IFDIAL thread.

Completion return code (RETCODE)

If the IFDTHRD call is unsuccessful, Model 204 returns one of the following error codes:

Code	Error condition
95	Nonexistent new thread specified. Call ignored.
96	New thread is already the current thread and is not detached.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE    PIC 9(5) COMP SYNC.
    05 NEWID      PIC 9(5).
    05 OLDID      PIC 9(5).
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFDTHRD" USING RETCODE, NEWID, OLDID.
```

IFDVAL call *-mc,sc*

Function The IFDVAL call (DELETE VALUE) deletes from the current record the first occurrence of a Model 204 field that matches the specified name=value pair.

Full syntax (32)
```
IFDVAL(RETCODE,FIELD_NAME,VALUE,CURSOR_NAME)
```

Compile-only syntax A compile-only form of IFDVAL is not available.

Execute-only syntax An execute-only form of IFDVAL is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FIELD_NAME	[I,c,r] The field name is a required input parameter which specifies the name of the Model 204 field to be deleted. You can specify only one field to be deleted per call. Specify a character string variable followed by a semicolon (;).
VALUE	[I,c,r] The value is a required input parameter which specifies the field value corresponding to the specified field name. Taken together, the field name and value form a pair for matching against the record. You can specify only one value for the field. Specify a character string variable followed by a semicolon (;).
CURSOR_NAME	[I,s,r] The name of the cursor is an input parameter that is available only for use with a multiple cursor IFSTRT thread and is required for specifying the record to be deleted. Specify the cursor name as a short character string, using the name previously assigned to the cursor in a corresponding IFOCUR call. See CURSOR_NAME on page 226 for a description of the cursor name for the IFOCUR call. **Note:** The cursor name is not a valid parameter for use with a single cursor IFSTRT thread.

Notes and tips Use the IFDVAL call to delete a Model 204 field that has been defined as having an INVISIBLE or VISIBLE attribute. Note the following rules:

- If a VISIBLE field is multiply occurring and the same field name = value pair occurs more than once, IFDVAL deletes only the first occurrence of the identical pairs.

- Use IFDVAL to delete any INVISIBLE fields for a record before using the IFDREC call to delete the record. This allows record numbers to be reused in the current file. See the Rocket Model 204 documentation wiki for information about reusing record numbers:

  http://m204wiki.rocketsoftware.com/index.php/File_design#Reuse_Record_Number_.28X.2740.27.29

Note: You cannot use IFDVAL on a sorted record set.

The IFDVAL call is valid on *all* types of IFSTRT threads. On a multiple cursor IFSTRT thread, you must specify the cursor name for the record that is to be deleted. On a single cursor IFSTRT thread, IFDVAL deletes a field from the current record using the current record set.

When FOPT=X'10' and the date/time stamp feature is installed, the IFDVAL function is *not* supported for DTS files.

Coding example (PL/1)

The following PL/1 call deletes from the current record the field name = value pair COLOR = BLUE.

```
•
•
•
CALL IFDVAL (RETCODE, 'COLOR;','BLUE;');
```

IFEFCC call *-mc,sc*

Function

The IFEFCC call (EXAMINE FIELD CONSTRAINT CONFLICT) returns specific information about field values or record numbers that cause a field constraint conflict using the IFSTOR, IFUPDT, or IFPUT HLI calls.

IFEFCC is similar to the SOUL ON FCC statement and its related $functions.

Full syntax (139)

```
IFEFCC(RETCODE,FIELDNAME,VALUE,RECNUM,STATEMENT,OLD-VALUE,
OLD-RECNUM,FILENAME)
```

Compile-only syntax

A compile-only form of IFEFCC is not available.

Execute-only syntax

An execute-only form of IFEFCC is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required output parameter. The code is a binary integer value.
FIELDNAM	[O,c,o] The field name is an optional output parameter which returns the name of the field in which the constraint violation occurred. In SOUL this name is available through $UPDFLD. The name is a character string.
VALUE	[O,c,o] The value is an optional output parameter which returns the field value causing the constraint violation. In SOUL this value is available through $UPDVAL. The value is returned as a character string.
RECNUM	[O,c,o] The record number is an optional output parameter that returns the internal number of the record whose update causes the conflict. In SOUL this name is available through $UPDREC. This parameter is a character string.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>STATEMENT</td><td>[O,c,o] The statement is an optional output parameter that returns the type of update operation causing the conflict. In SOUL it is available through $UPDSTMT.
The possible values are:<table><tr><th>Value</th><th>Use</th></tr><tr><td>STORE</td><td>For IFSTOR call</td></tr><tr><td>ADD</td><td>When the field name(*) form is used in the IFUPDT call</td></tr><tr><td>INSERT</td><td>When the field name(+n) form is used in the IFUPDT call</td></tr><tr><td>CHANGE</td><td>When the field name or field name(n) form is used in the IFUPDT call</td></tr></table>This parameter is a character string.</td></tr>
<tr><td>OLD-VALUE</td><td>[O,c,o] The old value is an optional output parameter that, in the case of an AT-MOST-ONE violation, returns the value of the original field occurrence causing the constraint violation. Otherwise, it returns a blank. In SOUL this value is available through $UPDOVAL. The value is returned as a character string.</td></tr>
<tr><td>OLD-RECNUM</td><td>[O,c,o] The old record number is an optional output parameter that, in the case of a uniqueness violation, returns the internal record number of the record already containing the field=value pair. Otherwise, it returns a -1. In SOUL this name is available through $UNQREC. This parameter is a character string.</td></tr>
<tr><td>FILENAME</td><td>[O,c,o] The file name is an optional output parameter that returns the name of the file in which the constraint violation occurred. In SOUL this name is available through $UPDFILE. The name is a character string.</td></tr>
</table>

Notes and tips

Use the IFEFCC call to determine the exact cause of a field constraint conflict when updating or storing data. Field constraint conflicts are caused when fields within a file violate either the UNIQUE or AT-MOST-ONE Model 204 field-level attribute.

A uniqueness conflict (return code 200) occurs when you try to store a non-unique field value (such as a duplicate telephone number) into a *file*. An AT-MOST-ONE conflict (return code 202) occurs when you try to store a second occurrence of a field into a *record* (such as a HEIGHT or EYE_COLOR field).

See the Rocket Model documentation wiki for more information about uniqueness and AT-MOST-ONE violations:

http://m204wiki.rocketsoftware.com/index.php/Field_design#AT-MOST-ONE.2C_REPEATABLE.2C_and_EXACTLY-ONE_attributes

Completion return code (RETCODE)

If no field constraint conflict is found, IFEFCC returns an error code of 15.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE     PIC 9(5) COMP SYNC.
    05  FIELDNAM    PIC X(255).
    05  VALUE       PIC X(255).
    05  RECNUM      PIC X(11).
    05  STATEMENT   PIC X(6).
    05  OLD-VALUE   PIC X(255).
    05  OLD-RECNUM  PIC X(11).
    05  FILENAME    PIC X(8) VALUE "FN1;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFEFCC" USING RETCODE, FIELDNAM, VALUE, RECNUM,
STATEMENT, OLD-VALUE, OLD-RECNUM, FILENAME.
```

IFENQ call *-mc,sc*

Function

The IFENQ call (ENQUEUE) enqueues on the arbitrary resource name specified.

Full syntax (39)

```
IFENQ(RETCODE,RESOURCE,ACTION,TIME)
```

Compile-only syntax

A compile-only form of IFENQ is not available.

Execute-only syntax

An execute-only form of IFENQ is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
RESOURCE	[I,s,r] The resource is a required input parameter which specifies the name of the resource to be enqueued. Specify a short character string, up to 32 characters in length. You may specify any data in the input string except for 32 bytes of binary zeroes.
ACTION	[I,i,r] The action is a required input parameter which specifies whether to enqueue on the named resource in share or exclusive mode. Specify one of the following integer values: 1 = Enqueues in share mode 2 = Enqueues in exclusive mode
TIME	[I,i,r] The time is a required input parameter. Specify an integer value which is the wait time in seconds.

Notes and tips

All users share an internal enqueuing table and should develop standard resource naming conventions to make arbitrary resource enqueuing effective.

Completion return code (RETCODE)

Model 204 returns the following completion codes for IFENQ:

Code	Condition
0	Resource was enqueued successfully.
3	Control of the requested resource could not be obtained within the specified time limit.

If the first attempt to enqueue on the specified resource fails, Model 204 waits the number of seconds specified in the time parameter, then tries again. If the second attempt to enqueue fails, Model 204 returns a completion code of 3 to the HLI program.

See the Rocket Model 204 documentation wiki page for more information on the ENQRETRY parameter:

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

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05  RETCODE     PIC 9(5) COMP SYNC.
    05  RESOURCE    PIC X(9) VALUE "CUSTFILE;".
    05  ACTION      PIC 9(5) COMP SYNC VALUE 1.
    05  TIME        PIC 9(5) COMP SYNC VALUE 10.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFENQ" USING RETCODE, RESOURCE, ACTION, TIME.
```

IFENQL call -sc

Function The IFENQL call (ENQUEUE LIST) enqueues that set of records on the specified list.

Full syntax (40)

```
IFENQL(RETCODE,LIST_NAME,ACTION,TIME)
```

Compile-only syntax A compile-only form of IFENQL is not available.

Execute-only syntax An execute-only form of IFENQL is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LIST_NAME	[I,c,r] The list name is a required input parameter which specifies the name of a list. Specify the list as a character string.
ACTION	[I,i,r] The action is a required input parameter which specifies whether to enqueue on the set of records in share or exclusive mode. Specify one of the following integer values: 1 = Enqueues in share mode 2 = Enqueues in exclusive mode
TIME	[I,i,r] The time is a required input parameter. Specify an integer value which is the number of times to retry; the wait time is in three second periods.

Notes and tips On a single cursor IFSTRT thread, a subsequent call to IFPROL places the current record on the list and enqueues on the record itself with the status specified for the list. On a single cursor IFSTRT thread, a subsequent call to IFRRFL removes the record from the list and dequeues it.

On a single cursor IFSTRT thread, a call to IFLIST following a call to IFENQL dequeues the set of records currently on the list, clears the list, and places the current set on it with no enqueuing.

Note that calls to IFLIST, IFPROL, and IFRRFL do not perform any enqueuing operations if IFENQL has not first been executed.

If a call to IFENQL results in an enqueuing conflict, Model 204 waits at most three seconds and then tries again, for as many times as specified in the time parameter (the wait time is in three-second periods). After trying unsuccessfully

for the number of times indicated in the time parameter, Model 204 returns a completion code of 3 to the HLI program.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05  RETCODE     PIC 9(5) COMP SYNC.
    05  LISTNAME    PIC X(9) VALUE "CARSLIST;".
    05  ACTION      PIC 9(5) COMP SYNC VALUE 1.
    05  TIME        PIC 9(5) COMP SYNC VALUE 5.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFENQL" USING RETCODE, LISTNAME, ACTION, TIME.
```

IFEPRM call *-mc,sc*

Function The IFEPRM call (EXAMINE PARAMETER) returns the value of the specified Model 204 parameter.

Full syntax (25)
```
IFEPRM(RETCODE,PARM_NAME,PARM_VALUE,FILE_SPEC)
```

Compile-only syntax A compile-only form of IFEPRM is not available.

Execute-only syntax An execute-only form of IFEPRM is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
PARM_NAME	[I,c,r] The parameter name is a required input parameter, which specifies the name of the Model 204 parameter whose value will be returned. Specify a character string, the keyword name of any Model 204 system, file or user parameter. Note that on a multiple cursor IFSTRT thread you may specify the file context for a Model 204 file parameter by using FILE_SPEC, described below.
PARM_VALUE	[O,c,r] Value is a required output parameter which specifies the location of the output parameter for the return parameter value. Specify a character string variable. Model 204 returns the value as a character string or as a character representation of a numeric value, up to eleven characters in length.
FILE_SPEC	[I,s,o] The file specification is an optional input parameter for use only with a multiple cursor IFSTRT thread for specifying the name of the file for which the Model 204 file parameter will be returned. Specify the Model 204 file name as a short character string using the following format: `IN [FILE] filename` The specified file must be open on the thread, otherwise the call is unsuccessful and Model 204 returns a completion code equal to 4.

Notes and tips Use the IFEPRM call to examine or to retrieve a Model 204 parameter value.

The IFEPRM call is valid on *all* types of IFSTRT threads. IFEPRM can be used on a single cursor IFSTRT thread to access the current record number when it is executed following an IFGET call and specifies CURREC as the parameter name.

Alternatively, on a multiple cursor thread every cursor has a current record, so CURREC need not be specified and its use is illegal. You can use IFRNUM to get the current record from the specified cursor. See the IFRNUM call.

Note: The file context can change on a multiple cursor thread. If a Model 204 file parameter is specified for PARM_NAME and the file specification parameter (FILE_SPEC) is omitted, IFEPRM returns a value for the default file on the thread (that is, the last file opened).

Completion return code (RETCODE)

If the IFEPRM call is unsuccessful, Model 204 returns an error code of 4 if an attempt was made to view a parameter that cannot be viewed.

Coding example (PL/1)

The IFEPRM call below sets the string variable INVAR to the value of the Model 204 system parameter LIBUFF:

```
CALL IFEPRM (RETCODE,'LIBUFF',INVAR);
```

IFERLC call *-mc,sc*

Function

The IFERLC call (EXAMINE RECORD LOCKING CONFLICT) returns a file name, record number, and user name after a record locking conflict occurs in an HLI program that issued IFFIND, IFUPDT, or any other call that requires locking a record. See the *Rocket Model 204 Host Language Interface Programming Guide* for information about which calls lock records and record sets.

IFERLC is similar to the SOUL ON RLC statement.

Full syntax (138)

```
IFERLC(RETCODE,RECNUM,USERNUM,FILENAME)
```

Compile-only syntax

A compile-only form of IFERLC is not available.

Execute-only syntax

An execute-only form of IFERLC is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required output parameter. The code is a binary integer value.
RECNUM	[O,c,o] The record number is an optional output parameter that returns the internal record number for which the record locking conflict occurred. This parameter is a character string.
USERNUM	[O,c,o] The user number is an optional output parameter that returns the user number of the user with which the IFAM program conflicted. The parameter is a character string.
FILENAME	[O,c,o] The file name is an optional output parameter which returns the name of the file in which the record locking conflict occurred. The name is a character string.

Notes and tips

Use the IFERLC call after a record locking conflict (return code 3) is returned by Model 204 to an HLI program that issued a call that locks a record (such as IFFIND, IFSTOR or IFUPDT). See the Rocket Model 204 documentation wiki for more information about record locking conflicts:

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

Completion return code (RETCODE)

If no record locking conflict is found, IFERLC returns an error code of 15.

See the descriptions of the $RLCFILE, $RLCREC, and $RLCUSER functions in the Rocket Model 204 documentation wiki for more information:

http://m204wiki.rocketsoftware.com/index.php/Category:SOUL_$functions

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
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) VALUE "FN1;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFERLC" USING RETCODE, RECNUM, USERNUM, FILENAME.
```

IFERR call *-mc,sc*

Function The IFERR call (ERROR) places an error message on the Model 204 journal.

Full syntax (28) `IFERR(RETCODE,PRINT_OP,ERR_MSG)`

Compile-only syntax A compile-only form of IFERR is not available.

Execute-only syntax An execute-only form of IFERR is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value.
PRINT_OP	[I,i,r] The print option is a required input parameter. Specify any of the following integer values for printing the message, or sum any of the values: 0 = Print the message only on the journal. 1 = Print the message on the operator's console and on the journal. 128 = Print the message on the journal and snap the current thread's storage areas including the Host Language Interface/Model 204 server area and control blocks.
ERR_MSG	[I,c,r] The error message is a required input parameter which specifies the message to be printed. Specify a character string less than 256 characters in length.

Notes and tips Use the IFERR call to document errors encountered during host language program processing. Model 204 writes the message in the journal as an AD line. You can use the IFERR call on any type of IFSTRT thread.

Coding example (COBOL) In the example below, a message is written to the Model 204 journal.

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE       PIC 9(5) COMP SYNC.
    05  AUDIT-ERROPT  PIC 9(5) VALUE 0.
•
•
•
```

```
01  MESSAGES.
    05  MESSAGE-A       PIC X(19) VALUE "INPUT CODE INVALID;".
    05  MESSAGE-B       PIC X(19) VALUE "RECORD NOT FOUND;".


•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFERR" USING RETCODE, AUDIT-ERROPT, MESSAGE-B.
```

IFFAC call *-mc,sc*

Function

The IFFAC call (FIND AND COUNT) creates a found set and returns the record count in an output parameter.

Full syntax (126)

```
IFFAC(RETCODE,FIND_SPEC,COUNT,FAC_NAME,%VARBUF,%VARSPEC)
```

Compile-only syntax (127)

```
IFFACC(RETCODE,FIND_SPEC,FAC_NAME)
```

Execute-only syntax (128)

```
IFFACE(RETCODE,COUNT,FAC_NAME,%VARBUF,%VARSPEC)
```

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FIND_SPEC	[I,c,r] The find specification is a required input parameter which is the selection criteria to be used for retrieving records. Specify a character string. See FIND_SPEC on page 168 for a detailed description of the find specification used for the IFFIND call that is also valid for IFFAC. The file context can change on a multiple cursor thread and, if the file specification is omitted, IFFAC processes records from the file or group that is the default on the thread at the time that the IFFAC is compiled.
COUNT	[O,i,r] The count parameter is a required parameter which specifies the location of the output parameter for the return count value. Specify an integer variable. Model 204 returns the record count as a fullword binary number.
FAC_NAME	[I,s,r/o] The name of the IFFAC compilation is an input parameter that is required for use with a multiple cursor IFSTRT thread, and is only required for a single cursor IFSTRT thread if using the Compiled IFAM facility (IFFACC and IFFACE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). On a single cursor IFSTRT thread, any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. On a multiple cursor IFSTRT thread, the first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). **Note:** A null value is equivalent to omitting the name parameter, and is not valid for a multiple cursor thread.

Parameter	Description
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFFAC call to retrieve records and to obtain a count of the records that meet the retrieval conditions.

Note: The IFFAC call is permitted on *all* types of IFSTRT threads and is useful for reducing CRAM overhead.

The IFFAC call is the equivalent of the FIND AND PRINT COUNT statement in SOUL in the host language environment. See the Rocket Model 204 documentation wiki for information about the FIND AND PRINT COUNT statement:

http://m204wiki.rocketsoftware.com/index.php/Basic_SOUL_statements_and_commands#Counting_records

Processing records from a found set

There are differences between single cursor and multiple cursor IFSTRT threads in processing records from a found set.

When a set is found on a single cursor IFSTRT thread, and while it is the current set, you can use IFGET to retrieve individual records. On a single cursor IFSTRT thread, you must save the current found set on a list before issuing any call which creates a new set if you want to access the previously found records.

On a multiple cursor thread, use IFOCUR to open a cursor to a found set any time after it is established and use the IFFTCH call to retrieve individual records.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5) COMP SYNC.
    05  COUNT    PIC 9(5) COMP SYNC.
    05  FDSPEC   PIC X(12) VALUE "MAKE='FORD';END;".
```

```
    05  FDNAME   PIC X(7) VALUE "FDFORD;".
•
•
•
PROCEDURE DIVISION.

•
•
•
CALL "IFFAC" USING RETCODE, FDSPEC, COUNT, FDNAME.
```

IFFDV call *-mc,sc*

Function The IFFDV call (FIND ALL VALUES) finds all values of a specified Model 204 field and creates a value set. The field must be defined having either KEY and FRV or ORDERED attributes.

Full syntax (74)
```
IFFDV(RETCODE,FIELD_NAME,FDV_NAME,%VARBUF,%VARSPEC)
```

Compile-only syntax (75)
```
IFFDVC(RETCODE,FIELD_NAME,FDV_NAME)
```

Execute-only syntax (76)
```
IFFDVE(RETCODE,FDV_NAME,%VARBUF,%VARSPEC)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FIELD_NAME	[I,c,r] The field name is a required input parameter which specifies the name of the Model 204 field whose values are to be found. A field name variable or a %variable is valid. Specify the field as a character string using the following format: `[IN {FILE filename; | GROUP groupname;}] FDV fieldname; [FROM value1] [TO valuen]; [[NOT] LIKE pattern];END;` where: • *IN* clause is available only for a multiple cursor thread and its use is optional for specifying a file or group context other than the default.The file context can change on a multiple cursor thread, and if the file specification is omitted, IFFDV processes values from the file or group that is the default on the thread at the time that the IFFDV call is compiled. • *filename*|*groupname* specifies the name of a particular file or group context for the value set. • *FDV* is a required keyword that must be specified if the IN clause is used. • *fieldname* is required and specifies the name of a particular Model 204 field or field name variable. • *FROM* and *TO* clauses are optional and specify a minimum (greater than or equal to) value (FROM), a maximum (less than or equal to) value (TO), or a range of values (FROM and TO) for selection criteria. • *LIKE*|*NOT LIKE* clause is optional and specifies a string pattern for selection criteria. See the Rocket Model 204 documentation wiki for information about value specifications: http://m204wiki.rocketsoftware.com/index.php/Statement_syntax#Value_specification_syntax

Parameter	Description
FDV_NAME	[I,s,r/o] The name of the IFFDV compilation is an input parameter that is required for use with a multiple cursor IFSTRT thread, and is only required for a single cursor IFSTRT thread if using the Compiled IFAM facility (IFFDVC and IFFDVE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). On a single cursor IFSTRT thread, any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. On a multiple cursor IFSTRT thread, the first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). A null value is equivalent to omitting the name parameter, and is not valid for a multiple cursor thread.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %VARBUF parameter and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified

SQL performance

The SQL driver, if possible, uses the IFFDV call to process the DISTINCT qualifier in SQL, which avoids triggering Model 204 Long Requests and improves performance. This behavior applies to any SQL interface, such as Connect★ or a C program. Model 204 stipulates that an IFFDV call contain only one field with either an FRV attribute or an ORDERED attribute.

IFFDV applies to all SQL aggregate functions, AVG, COUNT, MAX, MIN and SUM. The MIN (DISTINCT *column-name*) requires only one fetch of the value set in question if the corresponding Model 204 field has an ORDERED attribute.

Limiting considerations

- Because of the discrepancy between Model 204 and SQL, a Model 204 string field must have either the ORDERED or the FRV attribute; a binary or float field must have the ORDERED attribute.
- The WHERE clause in the SELECT statement is limited to the scope of the arguments in an IFFDV call. For example:

  ```
  SELECT DISTINCT LAST_NAME FROM filename
  WHERE LAST_NAME BETWEEN 'A' AND 'C'
  AND LAST_NAME NOT LIKE 'ABC%'
  ```

- If SELECT statements reference only a single column in the query, the IFFDV performance enhancement is used. For example, the following query does *not* invoke an IFFDV call because the query references two columns, LAST_NAME and FIRST_NAME.

  ```
  SELECT DISTINCT LAST_NAME FROM filename
  WHERE LAST_NAME BETWEEN 'A' AND 'C'
  AND FIRST_NAME (NOT) LIKE 'ABC%'
  ```

- If you have more than one FROM or more than one TO clause, then IFFDV optimization is not performed, as the following query illustrates:

  ```
  SELECT DISTINCT LAST_NAME FROM filename
  WHERE LAST_NAME BETWEEN 'A' AND 'C'
  AND LAST_NAME <> 'ABC%'
  ```

- A query that contains an OR keyword is not optimized by IFFDV, as the following example illustrates:

  ```
  SELECT DISTINCT LAST_NAME FROM filename
  WHERE LAST_NAME BETWEEN 'A' AND 'C'
  OR LAST_NAME LIKE 'NEL%'
  ```

- A query that contains a NOT BETWEEN clause is not optimized by IFFDV, as the following example illustrates:

  ```
  SELECT DISTINCT LAST_NAME FROM filename
  WHERE LAST_NAME NOT BETWEEN 'A' AND 'M'
  ```

Notes and tips

Use the IFFDV call to retrieve the stored values of a particular field. You can specify only one field per call. You may specify retrieval conditions, a range or a pattern, to limit values in the found set. Note that in group context, the found value set is automatically sorted in ascending order, following the standard EBCDIC collating sequence.

The IFFDV call is permitted on *all* types of IFSTRT threads.

The IFFDV call is the equivalent of the FIND ALL VALUES statement in SOUL in the host language environment. See the Rocket Model 204 documentation wiki for information about the FIND ALL VALUES statement:

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

Processing records from a value set

There are differences between single cursor and multiple cursor IFSTRT threads in processing records from a value set.

When a set is found on a single cursor IFSTRT thread, and while it is the current set, you can use IFGETV to retrieve individual values. On a multiple cursor thread, use IFOCUR to open a cursor to a value set any time after it is established using Compiled IFAM and use the IFFTCH call to retrieve individual values.

Coding examples (COBOL)

The examples below find all the values of the field COLOR. In this example, IFFDV is called without the compilation name or a %variable:

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE     PIC 9(5) COMP SYNC.
    05  FIELD-NAME  PIC X(6) VALUE "COLOR;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFFDV" USING RETCODE, FIELD-NAME.
```

In the example below, IFFDV is called using the compilation name and a %variable:

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE     PIC 9(5) COMP SYNC.
    05  FIELD-NAME  PIC X(8) VALUE "%%FNVAR;".
    05  FD-NAME     PIC X(23) VALUE "IFFDV COMPILATION NAME;".
    05  PCV         PIC X(6) VALUE "COLOR;".
    05  PCV-SPEC    PIC X(20) VALUE "EDIT(%FNVAR)(A(5));".
•
•
•
PROCEDURE DIVISION.
•
•
•
    CALL "IFFDV" USING RETCODE, FIELD-NAME, FD-NAME, PCV,
    PCV-SPEC.
```

IFFILE call *-mc,sc*

Function The IFFILE call (FILE) adds the specified field to each record in a found set and saves the updated records.

Full syntax (22)

```
IFFILE(RETCODE,FIELD_SPEC,SET_QUAL)
```

Compile-only syntax A compile-only form of IFFILE is not available.

Execute-only syntax An execute-only form of IFFILE is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FIELD_SPEC	[I,c,r] The field specification is a required input parameter which specifies the name and value pair for the field that will be added to the records in the set. Specify a character string using the following name=value pair format: `fieldname=fieldvalue;` where: *fieldname* is the name of the pre-defined Model 204 field to be added to each record in the set. You may specify only one field per IFFILE call. The field must be predefined to Model 204 as having both KEY and INVISIBLE attributes. Specify the name as a character string, up to 255 characters in length. *fieldvalue* is the value for the specified field in the pair. A value is required. Values may be specified in decimal form, such as 193, in hexadecimal form, such as X‘C1', or in character form, such as C‘A'.

Parameter	Description
SET_QUAL	[I,c,r] The set qualifier is available only for use with a multiple cursor IFSTRT thread and it is required for specifying the record set or list that will be used to add fields to records in a file or group. Specify a character string using either one of the following formats: `{IN label | ON [LIST} listname}` where: *label* is the name of a saved IFFIND, IFFNDX, IFFWOL, or IFFAC compilation from the previously compiled call which established the record set. *listname* specifies the name of a list which contains the found set. **Note:** The set qualifier is not a valid parameter for use with a single cursor IFSTRT thread.

Notes and tips

Use the IFFILE call to update and retain a set of records for future use. The IFFILE call adds the specified field to all the records that are in the found set. IFFILE does not enqueue on a set of records. Note that you cannot use IFFILE with a sorted record set.

When FOPT=X'10' and the date/time stamp feature is installed, the IFFILE function is supported for DTS files.

Once records are updated with the IFFILE call, you can retrieve all the records in the found set by referencing the added field in a retrieval call. Overall, this is more efficient than updating records one at a time.

The IFFILE call is valid on *all* types of IFSTRT threads. You must specify the found set of records that are to be updated on a multiple cursor IFSTRT thread. On a single cursor IFSTRT thread, IFFILE updates records using the found set that is current.

Note: IFFILE deletes a field from a record in a file or group if the field already exists. Use IFFILE with caution because it can delete a field from records which are not in the current set.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE        PIC 9(5) COMP SYNC.
    05 KEY-INV-FIELD  PIC X(10) VALUE "SAVREC=2;".
•
•
•
PROCEDURE DIVISION.
•
•
•
CALL "IFFILE" USING RETCODE, KEY-INV-FIELD.
```

IFFIND call *-mc,sc*

Function The IFFIND call (FIND) selects records from a Model 204 file or group in share mode and creates a found set.

Full syntax (13)
```
IFFIND|IFFD(RETCODE,FIND_SPEC,FIND_NAME,%VARBUF,%VARSPEC)
```

Compile-only syntax (46)
```
IFFINDC|IFFDC(RETCODE,FIND_SPEC,FIND_NAME)
```

Execute-only syntax (47)
```
IFFINDE|IFFDE(RETCODE,FIND_NAME,%VARBUF,%VARSPEC)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.

Parameter	Description

FIND_SPEC	[I,c,r] The find specification is a required input parameter which is the selection criteria to be used for retrieving records. This find specification may also be used for the IFFAC call. Specify the selection criteria as a character string using the where:
	`[IN {FILE filename: | GROUP groupname;}]`
	`FD [set qualifier]`
	`spec1;...specn; END;`
	where:
	IN clause is available only for a multiple cursor thread and its use is optional for specifying a file or group context other than the default.
	Note: The file context can change on a multiple cursor thread and, if the file specification is omitted, IFFIND is compiled against the last file opened which is the default file or group on the thread.
	filename|*groupname* specifies the name of a particular file or group context for the record set.
	FD is a required keyword that must be specified if the IN clause is used.
	set qualifier is available only for a multiple cursor thread and its use is optional for specifying the previously established record set or list from which records will be retrieved. Note that a set qualifier is illegal if the IN FILE clause is specified. Specify the set qualifier as a character string using one of the following formats:
	`{IN label | ON [LIST} listname}`
	where:
	• *label* is the name of a saved IFFIND or IFFAC compilation from a previously compiled call.
	• *listname* specifies the name of a list.
	spec is a valid retrieval specification (1 through n). End the specification with a semicolon (;).

Parameter	Description
	Note: To be selected, a record must meet all of the retrieval specifications. A specification can be any Boolean combination of conditions using the following elements: • Files from the current group • Lists of records • Physical record numbers • Sort fields • KEY or NON-KEY fields • NUMERIC RANGE or NON-RANGE (alphanumeric) fields • ORDERED or NON-ORDERED fields. *END;* is the required keyword and semicolon delimiter which indicates the end of the find specification. See the Rocket Model 204 documentation wiki for information about retrieval conditions and selection results: http://m204wiki.rocketsoftware.com/index.php/Basic_SOUL_statements_and_commands#Find_statement
FIND_NAME	[I,s,r/o] The name of the IFFIND compilation is an input parameter that is required for use with a multiple cursor IFSTRT thread, and is only required for a single cursor IFSTRT thread if using the Compiled IFAM facility (IFFINDC and IFFINDE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). On a single cursor IFSTRT thread, any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. On a multiple cursor IFSTRT thread, the first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). **Note:** A null value is equivalent to omitting the name parameter, and is not valid for a multiple cursor thread.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation

Parameter	Description
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFFIND call to retrieve records. The find specification is based on a specified combination of retrieval conditions, the selection criteria.

Note that IFFIND does not verify the number of records in the found set. A return code of 0 indicates normal completion of the call whether none, one, or many records are found. To determine whether or not the selected set is empty, use the IFCOUNT call.

The IFFIND call is permitted on *all* types of IFSTRT threads.

Completion return code (RETCODE)

A Model 204 return code of 4 indicates abnormal completion of the IFFIND call for any of the following error conditions:

- An incorrect find specification (Model 204 does not save the compilation, the compilation name that is specified is not defined).
- A field name variable replaced by a nonexistent field (Model 204 saves the compilation but does not execute the find function).
- An incorrect %variable parameter

Record locking behavior

Records in the found set are retrieved in share mode. Alternatively, to lock the records of the current set in exclusive mode, use the IFFNDX call, or to select records without obtaining any locks, use the IFFWOL call.

The IFFIND call is the equivalent of the FIND statement in SOUL in the host language environment. See the Rocket Model 204 documentation wiki for information about the FIND statement:

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

Processing records from a found set

There are differences between single cursor and multiple cursor IFSTRT threads in processing records from a found set.

When a set is found on a single cursor IFSTRT thread, and while it is the current set, you can use IFGET to retrieve individual records. On a single cursor

IFSTRT thread, you must save the current found set on a list before issuing any call which creates a new set if you want to access the previously found records.

On a multiple cursor thread, use IFOCUR to open a cursor to a found set any time after it is established. Use the IFFTCH call to retrieve individual records.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5)  COMP SYNC.
    05  QUAL-1   PIC X(75) VALUE 'SEX=FEMALE;
    OCCUPATION=DOCTOR OR DENTIST;
    CITY=BOSTON;NAME LIKE "PAT*";END;'.
•
•
•
PROCEDURE DIVISION.



•
•
•
  CALL "IFFIND" USING RETCODE, QUAL-1.
```

IFFLS call *-mc,sc*

Function The IFFLS call (FIELD LEVEL SECURITY) checks security access level to a specified field, or to all fields, in a file or group on the current thread.

Full syntax (61) `IFFLS(RETCODE,FIELD_NAME,ACCESS_SPEC,FILE_IND)`

Compile-only syntax A compile-only form of IFFLS is not available.

Execute-only syntax An execute-only form of IFFLS is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value.
FIELD_NAME	[I,c,o] The field name is an optional input parameter which specifies the name of the field or identifies a field name variable for a field whose security access level is to be checked. **Note:** If the field name is not specified, IFFLS defaults to checking all of the fields in the indicated file or group. In this case, Model 204 returns a completion code of 0 only if the thread has the requested access to every field. Use IFFLS without specifying a field name with caution; since this requires that Model 204 examine every field, it can slow performance.
ACCESS_SPEC	[I,s,o] The access specification is an optional input parameter which summarizes the needed field access. Specify a short character string which contains any of the following character codes: *S* = SELECT *R* = READ *U* = UPDATE *A* = ADD **Note:** If you do not specify the access codes, the specification defaults to a value of SRUA, for all privileges.

Parameter	Description
FILE_IND	[I,s,o] The file indicator is an optional input parameter which specifies the file or group context for the access check. If you do not specify the file indicator, IFFLS uses the context established by the IFOPEN call. You may specify the file indicator in one of the following ways on a single cursor IFSTRT thread: • If the access is to be checked for a particular file in the group, specify that file name. • If the access is to be checked for the file containing the current record, specify the $CURFILE string. • To indicate the group update file, specify the $UPDATE string. In group context, IFFLS returns a completion code of 0 if there is a set of files in the group for which the access would be allowed. On a multiple cursor IFSTRT thread, you may specify any file or group using the following format: `filename` **Note:** On a multiple cursor IFSTRT thread, $UPDATE and $CURFILE are illegal.

Notes and tips

Use the IFFLS call to avoid or diagnose errors that occur because of field level security violations. You can use the IFFLS call on any type of IFSTRT thread.

Completion return code (RETCODE)

If the IFFLS call is unsuccessful, Model 204 returns an error code of 4 if any one of the following error conditions occurs:

- The indicated access (ACCESS_SPEC) is not allowed.
- The access string (ACCESS_SPEC) is invalid.
- The field name (FIELD_NAME) is not defined.
- No file or group is open.
- The file indicator (FILE_IND) is invalid.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE    PIC 9(5) COMP SYNC.
    05 FIELDNAME  PIC X(7) VALUE "SALARY;".
    05 ACCESS     PIC X(2) VALUE "U;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFFLS" USING RETCODE, FIELDNAME, ACCESS.
```

IFFLUSH call *-mc,sc*

Function

The IFFLUSH call (FLUSH) flushes compilations and %variables from the Model 204 server tables for the current thread.

Full syntax (45)

```
IFFLUSH|IFFLSH(RETCODE,NAME_SPEC)
```

Compile-only syntax

A compile-only form of IFFLUSH is not available.

Execute-only syntax

An execute-only form of IFFLUSH is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value.
NAME_SPEC	[I,c,o] The name specification is an optional input parameter which specifies a list of one or more compilation names, or %variables, or both, to be deleted from the server tables. Specify a short character string. Separate names in the list using a comma and end the string with a semicolon. **Note:** If the name string is not specified, IFFLUSH deletes all compilations and %variables from the corresponding server tables for the current thread.

Notes and tips

Use the IFFLUSH call to control space management for the Model 204 server tables, such as QTBL and STBL, that are occupied by compilations and %variables. You can delete items from storage that are no longer needed and make room for new compilations.

You can use the IFFLUSH call on any type of IFSTRT thread. IFFLUSH is only valid for use with the Compiled IFAM facility.

Using IFFLUSH on a multiple cursor IFSTRT thread

On a multiple cursor IFSTRT thread, except for IFFTCH, IFUPDT, and IFOCC compilations which may be flushed individually, IFFLUSH does not allow the flushing of individual compilations or %variables.

IFFLUSH flushes everything. If IFFLUSH is coded without a name list (as shown in the example below), it empties the server tables and frees all record sets and CCATEMP pages that are held by the HLI program. On a multiple cursor IFSTRT thread, IFFLUSH operates similarly to an END statement in SOUL.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE  PIC 9(5) COMP SYNC.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFFLS" USING RETCODE.
```

IFFNDX call *-mc,sc*

Function The IFFNDX call (FIND EXCLUSIVE) selects records from a Model 204 file or group, enqueuing in exclusive mode, and creates a found set.

Full syntax (56)
```
IFFNDX|IFFDX(RETCODE,FIND_SPEC,TIME_SPEC,FNDX_NAME,
%VARBUF,%VARSPEC)
```

Compile-only syntax (57)
```
IFFNDXC|IFFDXC(RETCODE,FIND_SPEC,FNDX_NAME)
```

Execute-only syntax (58)
```
IFFNDXE|IFFDXE(RETCODE,TIME_SPEC,FNDX_NAME,%VARBUF,%VARSPEC)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FIND_SPEC	[I,c,r] The find specification is a required input parameter which is the selection criteria to be used for retrieving records. Specify the selection criteria as a character string. See FIND_SPEC on page 168 for a detailed description of the find specification used for IFFIND that is also valid for IFFNDX.
TIME_SPEC	[I,i,r] The time specification is a required input parameter. Specify an integer value which is the number of times to try the find in the event of an enqueuing conflict; the wait time is in 3-second periods.
FNDX_NAME	[I,s,r/o] The name of the IFFNDX compilation is an input parameter that is required for use with a multiple cursor IFSTRT thread, and is only required for a single cursor IFSTRT thread if using the Compiled IFAM facility (IFFNDXC and IFFNDXE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). On a single cursor IFSTRT thread, any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. On a multiple cursor IFSTRT thread, the first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). **Note:** A null value is equivalent to omitting the name parameter, and is not valid for a multiple cursor thread.

Parameter	Description
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %VARBUF parameter and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFFNDX call to retrieve records and to hold the found set in exclusive mode. The find specification is based on a specified combination of retrieval conditions and is identical to the one that is used with the IFFIND call.

If a call to IFFNDX results in an enqueuing conflict, Model 204 waits at most three seconds and then tries again, for as many times as specified in the time parameter (the wait time is in 3-second periods). After trying unsuccessfully for the number of times specified in the time parameter, Model 204 returns a completion code of 3 to the HLI program.

For more information, see the description of the ENQRETRY parameter in the Rocket Model 204 documentation wiki:

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

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5)  COMP SYNC.
    05  QUAL-1   PIC X(75) VALUE 'SEX=FEMALE;
                           OCCUPATION=DOCTOR OR DENTIST;
                           CITY=BOSTON;NAME LIKE "PAT*";END;'.
    05  NUMRTRY  PIC 9(5) COMP SYNC VALUE 3.
•
•
•
PROCEDURE DIVISION.
•
```

```
•
•
  CALL "IFFNDX" USING RETCODE, QUAL-1, NUMRTRY.
```

IFFNSH call *-mc,sc,di*

Function

The IFFNSH call (FINISH) closes, dequeues, and deallocates all files and threads to terminate Host Language Interface processing.

Full syntax (3)

```
IFFNSH(RETCODE,USER_RETCODE,URC_TYPE)
```

Compile-only syntax

A compile-only form of IFFNSH is not available.

Execute-only syntax

An execute-only form of IFFNSH is not available.

Parameters

Parameter	Specifies…
RETCODE [O,i,r]	Model 204 return code, the required output parameter. The code is a binary integer value.
USER_RETCODE [O,i,o]	Model 204 return code for the current user. The code is a binary integer value and is output only.
URC_TYPE [I,c,o]	Type of return code to return in USER_RETCODE. Values are: • N for return code = 0 • O for Online value • B for batch value

Notes and tips

USER_RETCODE and URC_TYPE are optional; however, if you enter one, you must enter both. Incorrect URC_TYPE parameter values are equivalent to entering N.

If the connection to Model 204 is lost prior to the call to IFFNSH, the RETCODE is 1000, indicating a successful disconnect. The USER_RETCODE value returned, however, is -1, indicating that the USER_RETCODE value is unknown.

Use the IFFNSH call to complete the update unit and end the current transaction. Call IFFNSH when the application program no longer needs the Host Language Interface/Model 204 service program.

IFFNSH closes, dequeues, and deallocates all files and threads in the current host language job, including any thread initiated by an IFDIAL call. You can use IFFNSH on any type of IFSTRT thread. IFFNSH, together with IFSTRT, initiates CPSORT checkpointing. For more information about CPSORT checkpointing, see the *Rocket Model 204 Host Language Interface Programming Guide*.

Model 204 returns the completion codes for the IFFNSH call that are listed in Table 6-3.

Table 6-3. IFFNSH completion codes (RETCODE)

Code	Job	Condition
1nnn	IFAM1	*nnn* is the highest journal error message return code encountered during the run. For example, the message FILE IS FULL... has an associated return code of 48 and may cause an IFAM1 IFFNSH return code of 1048. Completion codes greater than 1080 indicate that severe system errors were encountered during the run and prohibit further calls to IFSTRT.
1000	IFAM2 IFAM4	The application program disconnected from Model 204 normally. Otherwise, the system returns one of the completion codes listed in Table 8-2 on page 335. All Host Language Interface functions return with register 15 set to 0. The return code that the application program returns to the operating system is not set by Model 204 or the Host Language Interface, but must be set by the application program itself.

IFAM2 CICS Interface

An application that uses the IFAM2 CICS interface must issue IFFNSH before the program ends.

If IFFNSH is not issued, Model 204 does not detect that the IFAM2 program ended. In addition, Model 204 leaves the Host Language Interface threads, which were established through IFSTRT and IFDIAL, unusable by any other user.

See the *Rocket Model 204 Host Language Interface Programming Guide* for more information about using IFFNSH for abend handling in an IFAM2 CICS program.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5) COMP SYNC.
•
•
•
PROCEDURE DIVISION.
OPEN-FILES.
  OPEN OUTPUT...
  CALL "IFSTRT" USING...
•
•
•
CLOSE-FILES.
•
•
•
```

```
CALL "IFFNSH" USING RETCODE.
```

IFFRN call *-mc*

Function

The IFFRN call (FOR RECORD NUMBER) creates a cursor, or opens an existing cursor on the current thread. The IFFRN cursor points to the specified record in the specified file and makes it the current record in the cursor.

Full syntax (118)

```
IFFRN(RETCODE,FILE_SPEC,RECNUM,FRN_NAME,%VARBUF,%VARSPEC)
```

Compile-only syntax (119)

```
IFFRNC(RETCODE,FILE_SPEC,FRN_NAME)
```

Execute-only syntax (120)

```
IFFRNE(RETCODE,RECNUM,FRN_NAME,%VARBUF,%VARSPEC)
```

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FILE_SPEC	[I,c,r] The file specification is a required input parameter which identifies the Model 204 file that contains the current record. Specify a character string using a standard Model 204 IN FILE clause. See the Rocket Model 204 documentation wiki for information about the IN FILE clause: http://m204wiki.rocketsoftware.com/index.php/Record_loops If a group is specified, a MEMBER clause is required.
RECNUM	[I,s,r] The record number is a required input parameter which specifies the internal record number. This is the current record to be processed in the specified file. Specify the record number as a short character string.
FRN_NAME	[I,s,r] The unique name of the IFFRN compilation is a required input parameter. A null string is not valid. Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). The first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). A null value is equivalent to omitting the name parameter, and is not valid. **Note:** Model 204 allocates a cursor as part of the saved IFFRN compilation. The cursor points to the record specified in the RECNUM parameter. You can reference this cursor using the compilation name in any single record host language function.

Parameter	Description
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFFRN call to establish a cursor pointing to a particular record in a particular file. IFFRN opens a cursor to a record which allows the record to be operated on by the IFFTCH and IFUPDT calls.

The IFFRN call is the equivalent of the FOR RECORD NUMBER statement in SOUL and replaces the IFPOINT call in the multiple cursor environment.

See the IFPOINT call. See the Rocket Model 204 documentation wiki for information about the FOR RECORD NUMBER statement:

http://m204wiki.rocketsoftware.com/index.php/Record_loops#FOR_RECORD_NUMBER_processing

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE   PIC 9(5) COMP SYNC.
    05  FILESPEC  PIC X(5) VALUE "IN FILE CARS;".
    05  RECNUM    PIC X(4) VALUE "175;".
    05  FRNCURS   PIC X(8) VALUE "FRNCURS;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFFRN" USING RETCODE, FILESPEC, RECNUM, FRNCURS.
```

IFFTCH call *-mc*

Function The IFFTCH call (FETCH) processes the next logical record or value and returns specified data to the user. IFFTCH specifies the cursor for the next logical record from which data is to be processed.

Full syntax (98)
```
IFFTCH(RETCODE,BUFFER,DIRECTION,CURSOR_NAME,EDIT_SPEC,
FTCH_NAME,%VARBUF,%VARSPEC,RECNUM)
```

Compile-only syntax (99)
```
IFFTCHC|IFFCHC(RETCODE,CURSOR_NAME,EDIT_SPEC,FTCH_NAME)
```

Execute-only syntax (100)
```
IFFTCHE|IFFCHE(RETCODE,BUFFER,DIRECTION,FTCH_NAME,
%VARBUF,%VARSPEC,RECNUM)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
BUFFER	[O,c,r] The buffer location is a required output parameter which specifies the address of the user's data area. The buffer contains the data returned by IFFTCH for the fields that are defined by the EDIT_SPEC parameter, described on page 184. Specify a character string.
DIRECTION	[I,i,r] The direction is a required input parameter which indicates the direction to move in the cursor for the specified found set. Specify the direction as a fullword binary number using a value in the range of -2,147,483,647 to 2,147,483,647. The cursor moves forward or backward a number of positions equal to your DIRECTION value. For example, if you specify a DIRECTION of 5, Model 204 processes every fifth record.
CURSOR_NAME	[I,c,r] Is a required input parameter which specifies the name of the cursor that points to the current record or value from which data is to be selected. This is a character string, the name previously assigned to the cursor in a corresponding IFOCUR call. See CURSOR_NAME on page 229 for a description of the cursor name for the IFOCUR call.
EDIT_SPEC	[I,c,r] The edit specification is a required input parameter which defines the fields that are to be returned from the specified record or value set. The specification describes the format of the data which is returned at the buffer location (see BUFFER on the previous page). For a record set, specify a character string using one of the following LIST, DATA, or EDIT format options:

Parameter	Description

	`LIST (fieldname list);`
	`DATA (fieldname list);`
	`DATA;`
	`EDIT (fieldname list) (edit format);`
	`EDIT (fieldname list1) (edit format1) (fieldname list2) (edit format2);`
	where:
	fieldname list is required and specifies a field name or names. Specify elements in the field name list using one of the following options:
	• fieldname
	• fieldname(n)
	• fieldname(*)
	• fieldname(%variable)
	where:
	• *fieldname* retrieves the first occurrence of the named field.
	• *fieldname(n)* retrieves the nth occurrence of the named field.
	• *fieldname(*)* retrieves all occurrences of the named field in the order of occurrence.
	• *fieldname(%variable)* retrieves the occurrence of the field specified by the %VARBUF and %VARSPEC parameters.
	edit format is required in the EDIT specification and specifies a code or codes which indicate(s) the format of the data to be returned for the named field in the fieldname list-edit format pair. See page 184 for a detailed description of the EDIT format codes that are used with IFFTCH.
	For a value set, specify a character string using either of the following LIST or EDIT format options:
	`LIST;`
	`IT (edit format);`
	• *edit format* is required in the EDIT specification. See page 184 for a detailed description of the EDIT format codes that are used with IFFTCH. See Chapter 7 for a description of LIST, DATA, and EDIT formatting.
	See “Special data handling” on page 186 for information about using multiple edit specifications for a data record.

FTCH_NAME	[I,s,o] The name of the IFFTCH compilation is an optional input parameter. If specified, Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). The first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). A null value is equivalent to omitting the name parameter, and is not valid.

Parameter	Description
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.
RECNUM	[O,i,o] Record number is an optional output parameter that will return the Model 204 internal record number. The number is displayed as an integer. For value set cursors, RECNUM always returns a zero.

Notes and tips

Use the IFFTCH call to access records or data. For a record set, the IFFTCH call operates in the following ways:

- No single record lock is obtained on the record that is being processed
- The record is not removed from the base record set after being returned.

Except for the record processing operations listed above, the IFFTCH call operates in the multiple cursor environment similarly to the single cursor IFGET and IFGETV calls, and (using DIRECTION=0) the IFMORE call. See the IFGET, IFGETV, and IFMORE calls.

Special data handling

You may need to use more than one IFFTCH call to assemble a record. To fetch data in segments, issue two or more successive IFFTCH calls, each having an EDIT_SPEC for a different portion of the record. Note that the second IFFTCH call must specify DIRECTION=0.

Specifying a cursor for IFFTCH

To successfully issue an IFFTCH call, you must specify a cursor which is open on the thread. See the IFOCUR (open cursor) call.

Using forward and backward skip processing

IFFTCH allows you to skip records or values when going either forward or backward. IFFTCH accepts either a positive or negative value of *n* and will skip to the next *nth* record. For example, if you specify a DIRECTION of 5, Model 204 will process every fifth record. Specify the direction as a fullword binary number using a value in the range of -2,147,483,647 to 2,147,483,647. If you set DIRECTION to 0, Model 204 assumes that you are fetching the same record.

Forward and backward skip processing can be used with the following types of record sets:

- Ordered Index record or value sets
- UNORDERED record or value sets
- SORTED record or value sets
- SORTED file record sets

Backward skip processing

You may use backward skip processing with Ordered Index record sets if one of the following is true:

- EACH is specified in the IFOCUR call.
- the field being processed is defined as OCCURS 1.
- the field being processed is defined as AT-MOST-ONE.

Negative settings invoke backward skipping. However, you may not skip backward in a given record set until you have first skipped forward in the record set. That is, you may not start at the beginning of the record set and skip backward.

Coding example (COBOL)

In the coding example below, Model 204 fetches every second (every other) record as specified by the DIRECTION option.

```
WORKING-STORAGE SECTION.
01  WORK-REC.
    05  WORK-SSN     PIC 9(9).
    05  WORK-NAME    PIC X(30).
•
•
•
01  CALL-ARGS.
    05  RETCODE      PIC 9(5) COMP SYNC.
    05  DIRECTION    PIC 9(9) COMP SYNC VALUE "2".
    05  CURSOR-NAME  PIC X(5) VALUE "CUR1;".
    05  EDIT-SPEC    PIC X(28) VALUE "EDIT (SSN,NAME)
                     (A(9),A(30));".
•
```

```
.
.
PROCEDURE DIVISION.




.
.
.
  CALL "IFFTCH" USING RETCODE, WORK-REC, DIRECTION,
                      CURSOR-NAME, EDIT-SPEC.
```

IFFWOL call *-mc,sc*

Function The IFFWOL call (FIND WITHOUT LOCKS) selects records from a Model 204 file or group without obtaining record locks and creates a found set.

Full syntax (87)
```
IFFWOL|IFFWO(RETCODE,FIND_SPEC,FWOL_NAME,%VARBUF,%VARSPEC)
```

Compile-only syntax (88)
```
IFFWOLC|IFFWOC(RETCODE,FIND_SPEC,FWOL_NAME)
```

Execute-only syntax (89)
```
IFFWOLE|IFFWOE(RETCODE,FWOL_NAME,%VARBUF,%VARSPEC)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FIND_SPEC	[I,c,r] The find specification is a required input parameter which is the selection criteria to be used for retrieving records. Specify the selection criteria as a character string. See FIND_SPEC on page 168 for a detailed description of the find specification used for IFFIND that is also valid for IFFWOL.
FWOL_NAME	[I,s,r/o] The name of the IFFWOL compilation is an input parameter that is required for use with a multiple cursor IFSTRT thread, and is only required for a single cursor IFSTRT thread if using the Compiled IFAM facility (IFFWOLC and IFFWOLE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). On a single cursor IFSTRT thread, any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. On a multiple cursor IFSTRT thread, the first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). **Note:** A null value is equivalent to omitting the name parameter, and is not valid for a multiple cursor thread.

Parameter	Description
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %VARBUF parameter and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax.
	%VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFFWOL call to retrieve records without locks. The find specification is based on a specified combination of retrieval conditions and is identical to the one that is used with the IFFIND call.

Completion return code (RETCODE)

A completion code of 4 indicates an IFFWOL error. If the IFFWOL call is unsuccessful, Model 204 returns a completion code of 4 for the following error conditions:

- An error in an IFFWOL specification. Note that Model 204 does not save the compilation and does not define the compilation name.
- A field name variable is replaced by a non-existent field. In this case, Model 204 does not execute the IFFWOL, but does save the compilation is if the name parameter is specified.
- An error is encountered in a %variable parameter.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5)  COMP SYNC.
    05  QUAL-1   PIC X(75) VALUE 'SEX=FEMALE;
                           OCCUPATION=DOCTOR OR DENTIST;
                           CITY=BOSTON;NAME LIKE "PAT*";END;'.
•
•
•
PROCEDURE DIVISION.
```

```
.
.
.
  CALL "IFFWOL" USING RETCODE, QUAL-1.
```

IFGERR call *-mc,sc*

Function

The IFGERR call (GET ERROR) returns the text of the most recent error message issued by Model 204 for the current thread.

Full syntax (29)

```
IFGERR(RETCODE,MSG_AREA)
```

Compile-only syntax

A compile-only form of IFGERR is not available.

Execute-only syntax

An execute-only form of IFGERR is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value.
MSG_AREA	[O,c,r] The message area is a required output parameter which specifies the address of the user's data area. The work area contains the text of the error message that is returned by IFGERR. Specify a character string.

Notes and tips

Use the IFGERR call to retrieve the text of the latest error message issued by Model 204 for the current thread. IFGERR returns either the latest call cancellation message or the latest counting error message. Note that a counting error message is written to the audit trail as an ER line and is refreshed only when a new message is generated or when the thread is ended.

Note that IFGERR truncates messages that are longer than 80 bytes. Using an IFAM2 or IFAM4 thread, IFGERR also returns the completion code of the previous call.

For more information about Model 204 messages, see the Rocket Model 204 messages documentation.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE               PIC 9(5)  COMP SYNC.
    05  M204-ERROR-MESSAGE    PIC X(80) VALUE SPACES.
•
•
•
01  ERROR-REPORT.
    05  CONTROL-REPORT-CODE   PIC X.
    05  CONTROL-REPORT-DATA   PIC X(132).
•
•
```

```
•
WRITE-ERROR-REPORT.
•
•
•
  CALL 'IFGERR' USING RETCODE, M204-ERROR-MESSAGE.
  WRITE ERROR-REPORT FROM M204-ERROR-MESSAGE.
•
•
•
```

IFGET call -sc

Function The IFGET call (GET) processes the next logical record and returns specified data to the user.

Full syntax (15)
```
IFGET(RETCODE,BUFFER,EDIT_SPEC,GET_NAME,%VARBUF,
%VARSPEC,ORD_SPEC)
```

Compile-only syntax (48)
```
IFGETC(RETCODE,EDIT_SPEC,GET_NAME,ORD_SPEC)
```

Execute-only syntax (49)
```
IFGETE(RETCODE,BUFFER,GET_NAME,%VARBUF,%VARSPEC)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
BUFFER	[O,c,r] The buffer location is a required output parameter which specifies the address of the user's data area. The buffer contains the data returned by IFGET for the fields that are defined by the EDIT_SPEC parameter, described below. Specify a character string. **Note:** The data is placed in the data area from left to right. If parameters are being passed without dope vectors (IFSTRT language indicator = 2), the data is placed in the area as specified. No length checking is attempted because the Host Language Interface does not know the length of the data area.

EDIT_SPEC	[I,c,r] The edit specification is a required input parameter which defines the fields that are to be returned from the specified record. The specification describes the format of the data which is returned at the buffer location (see BUFFER above). Specify a character string using one of the following LIST, DATA, or EDIT format options:
	`LIST (fieldname list);`
	`DATA (fieldname list);`
	`DATA;`
	`EDIT`
	`EDIT`
	`(fieldname list2) (edit format2);`
	where:
	fieldname list is required and specifies a field name or names. Specify elements in the field name list using one of the following options:

Parameter	Description
	• fieldname • fieldname(n) • fieldname(*) • fieldname(%variable) where: • *fieldname* retrieves the first occurrence of the named field. • *fieldname(n)* retrieves the nth occurrence of the named field. • *fieldname(*)* retrieves all occurrences of the named field in the order of occurrence. • *fieldname(%variable)* retrieves the occurrence of the field specified by the %VARSPEC and %VARBUF parameters. *edit format* is required in the EDIT specification and specifies a code or codes which indicate(s) the format of the data to be returned for the named field in the field name list-edit format pair. See page 194 for a detailed description of the EDIT format codes that are used with IFGET. See Chapter 7 for a description of LIST, DATA, and EDIT formatting.
GET_NAME	[I,s,r/o] The name of the IFGET compilation is an input parameter that is only required if using the Compiled IFAM facility (IFGETC and IFGETE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string. Any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. A null value is equivalent to omitting the name parameter.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Parameter	Description

ORD_SPEC	[I,c,o] The order specification is an optional input parameter that provides ordered index functionality. This parameter works with multicursor IFAM only. The order specification specifies the ORDERED index field that is used to order the return of the records. Specify the order using an IN ORDER clause with the following format:
	`IN [ASCENDING | DESCENDING] ORDER`
	`[BY [EACH] fieldname]`
	`[FROM value1] [TO value2]`
	`[[NOT] LIKE pattern]`

	where: *ASCENDING* and *DESCENDING* are keywords that indicate the order in which the record set will be processed. ASCENDING order is the default. *fieldname* specifies the name of the field to be used for ordering the records. The *FROM* and *TO* clauses specify the range of values for fieldname, where *value1* specifies the beginning value, and *value2* specifies the ending value. You may specify the range using both FROM and TO, or using only FROM (for all values greater than or equal to), or only TO (for all values less than or equal to). Specifying a range limits the selection of records to be processed. If a range is specified, records that do contain the field are not processed. *pattern* specifies a field value in the form of a character string that is used to match against the record. Enclose the pattern string inside single quotation marks. See the Rocket Model 204 documentation wiki for a description of the valid pattern characters and examples of their use: http://m204wiki.rocketsoftware.com/index.php/Record_loops Specifying a pattern limits the selection of records to be processed. Only records which meet the pattern matching criteria are processed. **Note:** Order specification works with multicursor programs only. Change your program to a multicursor program, is necessary. For more information about the order specification, see the Rocket Model 204 documentation wiki: http://m204wiki.rocketsoftware.com/index.php/Record_loops

Notes and tips Use IFGET to retrieve the next record from the current set. Only fields that you specify are returned to the application program.

The call to IFGET specifies a data area into which the Host Language Interface places the retrieved data. IFGET also specifies the needed fields and the manner in which they are to be formatted in the data area.

Records from a sorted file are returned from the current set in sort key sequence. As each record is retrieved from the current set, its physical record number is stored in the parameter CURREC and it becomes the current record.

Note that execution of an IFEPRM call on a sorted set returns the record number of the database record. The record is then removed from the current set and the record set count is decreased by one.

Call IFCOUNT and IFLIST before the first IFGET from the current set. Otherwise, IFCOUNT and IFLIST do not reflect the entire set of records selected with IFFIND.

Completion return code (RETCODE)

Note the following special conditions about three of the possible completion codes for IFGET:

Code	Condition
1	Conversion of a nonnumeric value was attempted for a B, P, Z, or F edit format, or the integer portion of a numeric value was too large for the output area. The value returned in the data area is binary (HIGH-VALUES in COBOL). Processing of the field name list continues, with the result that more than one value may be in error.
2	Indicates the end of the found set.
4	If a nonexistent field name is encountered, the IFGET specification is executed and saved if appropriate, with the nonexistent field being ignored. Any other type of error prevents the specification from being executed or saved.

Coding example (Assembler)

Suppose that a particular record contains the following pairs:

```
SCHOOL=HARVARD,SCHOOL=YALE,SCHOOL=MICHIGAN STATE
```

The application program might include the following:

```
CALL IFGET,(ERR,WORKAREA, EDIT LIST),VL
•
•
•
WORKAREA DS CL120
EDITLIST (DC) CL20 'EDIT(SCHOOL(*)) (M);'
```

After the IFGET function completed, WORKAREA would contain the following:

```
X'0307C8C1D9E5C1D9C404E8C1D3C50AD6C8C9D640E2E3C1E3C5'
```

Coding example (COBOL)

```
01  RETCODE            PIC 9(5) COMP SYNC.
01  WORK-REC.
    05 WORK-SSN        PIC 9(9).
    05 WORK-NAME       PIC X(30).
    05 WORK-BDATE      PIC 9(6).
    05 WORK-SCDATE     PIC 9(6).
    05 FILLER          PIC X(2).
    05 WORK-GRADE      PIC 9(2).
    05 WORK-STEP       PIC 9(2).
•
•
01 EDITLIST-1          PIC X(68) VALUE
     'EDIT(SSN,NAME,BDATE,SCDATE,GRADE,STEP)(A(9),
  -    A(30),2A(6),'X(2),2J(2));'.
•
•
CALL 'IFGET' USING RETCODE, WORK-REC, EDITLIST-1.
•
•
```

IFGETV call *-sc*

Function The IFGETV call (GET VALUE) extracts the next field value from the current value set and returns the specified data to the user.

Full syntax (77)
```
IFGETV(RETCODE,BUFFER,EDIT_SPEC,GETV_NAME)
```

Compile-only syntax (78)
```
IFGTVC(RETCODE,EDIT_SPEC,GETV_NAME)
```

Execute-only syntax (79)
```
IFGTVE(RETCODE,BUFFER,GETV_NAME)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
BUFFER	[O,c,r] The buffer location is a required output parameter which specifies the address of the user's data area. The buffer contains the data returned by IFGETV as defined by the EDIT_SPEC parameter, described below. Specify a character string.

EDIT_SPEC	[I,c,r] The edit specification is a required input parameter which defines the format of the data to be returned from the specified value. The specification describes the format of the data which is returned at the buffer location (see BUFFER above). Specify a character string using either of the following LIST or EDIT format options:
	`LIST;`
	`EDIT (edit format);`
	where:
	edit format is required in the EDIT specification and specifies a code or codes which indicate(s) the format of the data to be returned for the current value. See page 199 for a detailed description of the EDIT format codes that are used with IFGETV.
	See Chapter 7 for a description of LIST, DATA, and EDIT formatting.

GETV_NAME	[I,s,r/o] The name of the IFGETV compilation is an input parameter that is only required if using the Compiled IFAM facility (IFGTVC and IFGTVE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string. Any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. A null value is equivalent to omitting the name parameter.

Notes and tips

Use IFGETV to retrieve the next value from the current set.

IFGETV is used to extract one field value from the value set built by IFFDV or IFSRTV.

Completion return code (RETCODE)

IFGETV must be called after IFFDV is called. Otherwise, Model 204 returns an error code of 2, which indicates that no value exists in the current set. Note that an error code of 2 may also indicate that the current set has been exhausted.

Coding example (Assembler)

The following z/OS ASSEMBLER example provides an IFGETV call. In the example, IFFDV has been called first to find all the values of field COLOR:

```
•
•
•
  CALL IFGETV,(RETCODE,DATA,SPEC),VL
•
•
•
RETCODE  DC F'0'
DATA     DC CL256' '
*
SPEC     DC C'LIST;'
*
```

In this example, if you specify LIST; the retrieved data appears in the data area with single quotation marks, for example, ‘RED'. Or, if you specify EDIT as shown in the example below, the retrieved data appears in the data area without quote marks, that is, as RED.

```
*
SPEC DC C'EDIT (A(3));'
```

IFGETX call *-sc*

Function

The IFGETX call (GET EXCLUSIVE) retrieves the next logical record from the current set, enqueuing on the record in exclusive mode, and returns the specified fields.

Full syntax (36)

```
IFGETX(RETCODE,BUFFER,EDIT_SPEC,TIME,GETX_NAME,
%VARBUF,%VARSPEC,ORD_SPEC)
```

Compile-only syntax

A compile-only form of IFGETX is not available; you can use IFGETC; see page 194.

Execute-only syntax (50)

```
IFGETXE|IFGTXE(RETCODE,BUFFER,TIME,GETX_NAME,%VARBUF,%VARSPEC)
```

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
BUFFER	[O,c,r] The buffer location is a required output parameter which specifies the address of the user's data area. The buffer contains the data returned by IFGETX for the fields that are defined by the EDIT_SPEC parameter, described below. Specify a character string.

EDIT_SPEC	[I,c,r] The edit specification is a required input parameter which defines the fields that are to be returned from the specified record. The specification describes the format of the data which is returned at the buffer location (see BUFFER above).
	Specify a character string using one of the following LIST, DATA, or EDIT format options:
	`LIST (fieldname list);`
	`DATA (fieldname list);`
	`DATA;`
	`EDIT (fieldname list) (edit formats);`
	`EDIT (fieldname list1) (edit format1) (fieldname list2) (edit format2);`
	where:
	fieldname list is required and specifies a field name or names. Specify elements in the field name list using one of the following options:
	• fieldname
	• fieldname (n)
	• fieldname(*)
	• fieldname(%variable)

Parameter	Description
	where: *fieldname* retrieves the first occurrence of the named field. • *fieldname(n)* retrieves the nth occurrence of the named field. • *fieldname(*)* retrieves all occurrences of the named field in the order of occurrence. • *fieldname(%variable)* retrieves the occurrence of the field specified by the %VARBUF and %VARSPEC parameters. *edit format* is required in the EDIT specification and specifies a code or codes which indicate(s) the format of the data to be returned for the named field in the fieldname list-edit format pair. See page 201 for a detailed description of the EDIT format codes that are used with IFGETX. See Chapter 7 for a description of LIST, DATA, and EDIT formatting.
TIME	[I,i,r] The time is a required input parameter. Specify an integer value which is the number of times to try the retrieval in the event of an enqueuing conflict; the wait time is in three-second periods.
GETX_NAME	[I,s,r/o] The name of the IFGETX compilation is an input parameter that is only required if using the Compiled IFAM facility (IFGETXE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string. Any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. A null value is equivalent to omitting the name parameter.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Parameter	Description
ORD_SPEC	[I,c,o] The order specification is an optional input parameter that provides ordered index functionality. The order specification specifies the ORDERED index field that is used to order the return of the records. Specify the order using an IN ORDER clause, such as is used with IFGET. See ORD_SPEC on page 196 for a description of the order specification used for IFGET that is also valid for IFGETX. **Note:** The order specification is interpreted only on the first IFGETX call following an IFFIND. Thereafter, it is only necessary to provide the first seven parameters on subsequent IFGETX calls for the same IFFIND. If you use the order specification parameter, all eight parameters must be coded, even if some contain dummy parameters. For more information about the order specification, see the Rocket Model 204 documentation wiki: http://m204wiki.rocketsoftware.com/index.php/Record_loops

Notes and tips

Use IFGETX to retrieve the next record from the current set. Only fields that you specify are returned to the application program.

IFGETX performs the same operations as IFGET. However, the record is enqueued upon in exclusive rather than in share status.

Use IFGETX if several users share a file and extensive processing is to be done between the IFGET which retrieves the record and an update operation on the record.

If a call to IFGETX results in an enqueuing conflict, Model 204 waits at most three seconds and then tries again, for as many time as specified in the time parameter (the wait time is in 3-second periods).

After trying unsuccessfully for the number of times specified in the time parameter, Model 204 returns a completion code of 3 to the HLI program. For more information, see the Rocket Model 204 documentation wiki:

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

Coding example (COBOL)

```
01  RETCODE             PIC 9(5) COMP SYNC.
01  WORK-REC.
    05 WORK-SSN         PIC 9(9).
    05 WORK-NAME        PIC X(30).
    05 WORK-BDATE       PIC 9(6).
    05 WORK-SCDATE      PIC 9(6).
    05 FILLER           PIC X(2).
    05 WORK-GRADE       PIC 9(2).
    05 WORK-STEP        PIC 9(2).
•
•
```

```
•
01  TIME               PIC 9(5) COMP SYNC VALUE 5.
01  EDITLIST-1        PIC X(68) VALUE
     'EDIT(SSN,NAME,BDATE,SCDATE,GRADE,STEP)(A(9),
  -    A(30),2A(6),'X(2),2J(2));'.
•
•
•
  CALL 'IFGETX' USING RETCODE, WORK-REC, EDITLIST-1, TIME.
•
•
•
```

IFHNGUP call *-di*

Function The IFHNGUP call (HANGUP) ends an IFDIAL or IFDIALN connection.

Full syntax (8)

```
IFHNGUP | IFHNGP(RETCODE,USER_RETCODE,URC_TYPE)
```

Compile-only syntax A compile-only form of IFHNGUP is not available.

Execute-only syntax An execute-only form of IFHNGUP is not available.

Parameters

Parameter	Specifies
RETCODE [O,i,r]	Model 204 return code, the required output parameter. The code is a binary integer value.
USER_RETCODE [O,i,o]	Model 204 return code for the current user. The code is a binary integer value and is output only.
URC_TYPE [I,c,o]	Type of return code to return in USER_RETCODE. Return values are: • N for return code = 0 • O for highest Online value • B for highest batch value

Notes and tips USER_RETCODE and URC_TYPE parameters are optional, however, if you enter one, you must enter both.

If the connection to Model 204 was lost prior to the call to IFHNGUP, the RETCODE value returned is 0. In this case, the USER_RETCODE value returns -1, indicating that it is unknown. If an IFDIAL application explicitly includes the LOGOUT command in the CCAIN stream, the connection is disconnected and a subsequent call to IFHNGUP returns an unknown USER_RETCODE.

Use the IFHNGUP call only with an IFDIAL connection. IFHNGUP ends a connection to the Model 204 SOUL facility that was established by a call to IFDIAL or IFDIALN.

IFHNGUP issues an IFATTN call and then sends a LOGOUT command to ensure that the caller's session is terminated.

Completion return code (RETCODE) If the IFHNGUP call is unsuccessful, Model 204 returns an error code of 100 which indicates that the connection was lost prior to the call.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5)  COMP SYNC.
•
•
•
PROCEDURE DIVISION.
OPEN-FILES.
  OPEN INPUT...
  CALL "IFDIAL"...
•
•
•
CLOSE-FILES.
•
•
•
  CALL "IFHNGUP" USING RETCODE.
```

IFINIT call *-mc,sc*

Function The IFINIT call (INITIALIZE) initializes a Model 204 file. IFINIT deletes all records in the file and reformats all corresponding Model 204 file tables.

Full syntax (23)
```
IFINIT(RETCODE,FIELD_DESC,FILE_SPEC)
```

Compile-only syntax A compile-only form of IFINIT is not available.

Execute-only syntax An execute-only form of IFINIT is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.

Parameter	Description
FIELD_DESC	[I,c,r] The field description is a required input parameter which names a field, or fields, and lists the field attributes. You may specify a sort or hash key field description, or a record security field description, or both, for the file that is being initialized. Specify a character string using the following format: `[{SORT | HASH} RECSCTY] field description;` where: *SORT* is the keyword which specifies that a sorted file is being initialized. A sorted file cannot be a hash key file (SORT and HASH are mutually exclusive). *HASH* is the keyword which specifies that a hash key file is being initialized. A hash key file cannot be a sorted file (SORT and HASH are mutually exclusive). *RECSCTY* is the keyword which specifies that a file having record security in effect is being initialized. A sorted or hash key file can also be initialized with record security. *field description* is the name of the field, followed by a list of field attributes enclosed in parentheses. Use commas or blanks to separate attributes. Model 204 automatically supplies the following field attributes: • Sort field: NON-CODED, VISIBLE, STRING • Hash key field: NON-CODED, VISIBLE, STRING, NON-KEY • Record security field: KEY You cannot specify the UPDATE option for sort or hash key field attributes. For sort or hash key fields, if a key is required in every record, specify OCCURS 1 LENGTH m, otherwise, if keys are not required in each record, you cannot specify the OCCURS attribute. For a record security field that is defined with the LENGTH m option, specify the value of m to be larger than the length allowed for LOGIN accounts.
	Note: If no attributes are specified, Model 204 defines the field assigning all of the default attributes for the file that is being initialized. For a description of field attributes for the DEFINE command in SOUL, see the Rocket Model 204 documentation wiki: http://m204wiki.rocketsoftware.com/index.php/Field_design [•••] indicates that a second field description may be optionally specified. If you specify both a sort or hash key field description and a record security field description, enter the record security description second. Separate each description with a comma (,). End the field description with a semicolon (;). **Note:** The field description parameter is required; specify a semicolon (;) for no field description.

Parameter	Description
FILE_SPEC	[I,s,o] The file specification is an optional input parameter for use only with a multiple cursor IFSTRT thread for specifying the name of the Model 204 file that is to be initialized. Specify the Model 204 file name as a short character string. `IN [FILE] filename` The specified file must be open on the thread, otherwise the call is unsuccessful and Model 204 returns a completion code of 4.

Notes and tips

Use the IFINIT call to clear a file and prepare it for use. Note that IFINIT cannot be used in a group context.

The IFINIT call is permitted on *all* types of IFSTRT threads. On a single cursor IFSTRT thread, IFINIT always initializes the current file (that is, the last file opened). On a multiple cursor IFSTRT thread, the file context can change. And if you do not specify a particular file using the FILE_SPEC parameter, IFINIT initializes the default file (that is, the last file opened).

The IFINIT call is the equivalent of the Model 204 INITIALIZE command in the host language environment and follows the same basic rules for use. See the Rocket Model 204 documentation wiki for information about using the INITIALIZE command:

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

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE   PIC 9(5)  COMP SYNC.
    05  INITVALS  PIC X(28) VALUE 'SORT ITEMNO,
                  RECSCTY RECSEC;'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFINIT" USING RETCODE, INITVALS.
```

IFLIST call -sc

Function The IFLIST call (LIST) places the current set of found records on a list.

Full syntax (17)
```
IFLIST(RETCODE,LIST_NAME)
```

Compile-only syntax A compile-only form of IFLIST is not available.

Execute-only syntax An execute-only form of IFLIST is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LIST_NAME	[I,c,r] The list name is a required input parameter which specifies the name of a list. Specify the name as a character string, up to 255 alphanumeric characters in length. Blank characters are not valid.

Notes and tips Use IFLIST to place records on a list. The list name can then be used in an IFFIND with the LIST$ specification.

If a list name is reused in a call to IFLIST, the old list is cleared and the original set of records on the list is no longer available. Otherwise, lists are available until the file is closed.

Lists built by IFLIST contain only pointers to records; they do not contain the actual data records. In a group context, the list can contain records from all files in the group.

Restrictions on using IFLIST

The following restrictions apply to using IFLIST:

- Lists cannot be shared across threads.
- You cannot use IFLIST with sorted record sets.

Coding example (COBOL)
```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE   PIC 9(5) COMP SYNC.
    05 LISTSAVE  PIC X(5) VALUE ‘SAVE;'.
•
•
•
PROCEDURE DIVISION.
```

```
.
.
.
  CALL "IFFIND" USING RETCODE, SPEC-1.
  CALL "IFLIST" USING RETCODE, LISTSAVE.
  CALL "IFFIND" USING RETCODE, SPEC-2.
  CALL "IFLIST" USING RETCODE, LISTSAVE.
```

In this example, only records from the second IFFIND are on the list named SAVE after the last IFLIST call is executed.

IFLOG call *-mc,sc*

Function The IFLOG call (LOGIN) identifies the user to Model 204 if a login is required.

Full syntax

```
IFLOG(RETCODE,LOGIN)
```

Compile-only syntax A compile-only form of IFLOG is not available.

Execute-only syntax An execute-only form of IFLOG is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value. See “Notes and tips” on the next page.
LOGIN	[I,c,r] The login information is a required parameter which supplies a valid Model 204 user ID and password that permit entry to the system. Specify the login as a character string using the following format: `userid [account]; password;` where: *userid* is required and is a character string that identifies the user who is logging into Model 204. *account* is an optional character string, from one to ten characters in length, that supplies an account under which the user is logging into Model 204. *password* is a character string that allows the specified user to access Model 204, and is required unless an external security package is being used. See Notes on the next page.

Notes and tips Use the IFLOG call to provide login information for an IFAM1 HLI application, as necessary. IFLOG is required in an IFAM1 program where the user authorization is to be validated by a security interface.

Note: IFLOG is available for use only in IFAM1 using an IFSTRT thread.

When an external security interface is performing login validation, Rocket recommends the following:

- Do not specify a user ID in the login for User 0.
- If a user ID is supplied on the LOGIN command, it must match the user ID of the owner of the address space, or this user ID must exist on CCASTAT. Otherwise, the login fails.

When using an external security interface, do not code the password in the host language program unless the user ID exists on CCASTAT. In this case, if a password is encountered, Model 204 interprets IFLOG as an invalid command.

Completion return code (RETCODE)

Model 204 returns either of the following completion codes for IFLOG:

Code	Condition
0	Indicates that the login was successful.
100	Indicates that the login failed.

Coding example (COBOL)

The COBOL coding example below specifies the following IFLOG parameter arguments: login account name USERABC, and login password ECP.

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE    PIC 9(5) COMP SYNC.
    05 LOGIN      PIC X(12) VALUE 'USERABC;ECP;'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFLOG" USING RETCODE, LOGIN.
  IF RETCODE IS NOT EQUAL TO ZERO, GO TO ERROR-ROUTINE.
•
•
•
```

IFMORE call *-sc*

Function The IFMORE call (MORE) continues the IFGET function to retrieve more data from the current record.

Full syntax (16)
```
IFMORE(RETCODE,BUFFER,EDIT_SPEC,MORE_NAME,
%VARBUF,%VARSPEC)
```

Compile-only syntax (51)
```
IFMOREC|IFMREC(RETCODE,EDIT_SPEC,MORE_NAME)
```

Execute-only syntax (52)
```
IFMOREE|IFMREE(RETCODE,BUFFER,MORE_NAME,%VARBUF,%VARSPEC)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
BUFFER	[O,c,r] The buffer location is a required output parameter which specifies the address of the user's data area. The buffer contains the data returned by IFMORE for the fields that are defined by the EDIT_SPEC parameter, described below. Specify a character string. **Note:** The data is placed in the data area from left to right. If parameters are being passed without dope vectors (IFSTRT language indicator = 2), the data is placed in the area as specified. No length checking is attempted because the Host Language Interface does not know the length of the data area.

EDIT_SPEC	[I,c,r] The edit specification is a required input parameter which defines the fields that are to be returned from the specified record. The specification describes the format of the data which is returned at the buffer location (see BUFFER above).
	Specify a character string using one of the following LIST, DATA, or EDIT format options:
	`LIST (fieldname list);`
	`DATA (fieldname list);`
	`DATA;`
	`EDIT (fieldname list) (edit formats);`
	`EDIT (fieldname list1) (edit format1) (fieldname list2) (edit format2);`
	where:
	fieldname list is required and specifies a field name or names. Specify elements in the fieldname list using one of the following options:

Parameter	Description
	• fieldname • fieldname (n) • fieldname(*) • fieldname(%variable) where: • *fieldname* retrieves the first occurrence of the named field. • *fieldname(n)* retrieves the nth occurrence of the named field. • *fieldname(*)* retrieves all occurrences of the named field in the order of occurrence. • *fieldname(%variable)* retrieves the occurrence of the field specified by the %VARBUF and %VARSPEC parameters. *edit format* is required in the EDIT specification and specifies a code or codes which indicate(s) the format of the data to be returned for the named field in the fieldname list-edit format pair. See page 214 for a detailed description of the EDIT format codes that are used with IFMORE. See Chapter 7 for a description of LIST, DATA, and EDIT formatting.
MORE_NAME	[I,s,r/o] The name of the IFMORE compilation is an input parameter that is only required if using the Compiled IFAM facility (IFMOREC and IFMOREE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string. Any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. A null value is equivalent to omitting the name parameter.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use IFMORE to get more data from the current record after using IFGET. IFMORE does not affect the current set and does not change the current record. You can use IFMORE until you have completed all necessary operations on the current record.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE      PIC 9(5) COMP SYNC.
    05 WORK-AREA-1  PIC x(256).
    05 WORK-AREA-2  PIC X(256).
•
•
•
01  SPEC-1          PIC X(64) VALUE
    EDIT (SSN,NAME,BDATE,SCDATE,GRADE,STEP)
    (A(9),A(30),2A(6),2J(2));'.
01  SPEC-2          PIC X(34) VALUE
    'EDIT (STATUS,SALARY)(A(11),A(12));'.
•
•
•

PROCEDURE DIVISION.
•
•
•
  CALL "IFGET" USING RETCODE, WORK-AREA-1, SPEC-1
  IF GRADE IS EQUAL TO 9 THEN PERFORM GETMORE.
•
•
•
GETMORE.
  CALL "IFMORE" USING RETCODE, WORK-AREA-2, SPEC-2.
```

IFMOREX call *-sc*

Function

The IFMOREX call (MORE EXCLUSIVE) continues the IFGETX function to retrieve more data from the current record, enqueuing on the record in exclusive mode.

Full syntax (37)

```
IFMOREX|IFMREX(RETCODE,BUFFER,EDIT_SPEC,TIME,
MOREX_NAME,%VARBUF,%VARSPEC)
```

Compile-only syntax

A compile-only form of IFMOREX is not available. You can use IFMOREC; see page 214.

Execute-only syntax (53)

```
IFMORXE|IFMRXE(RETCODE,BUFFER,MOREX_NAME,%VARBUF,%VARSPEC)
```

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
BUFFER	[O,c,r] The buffer location is a required output parameter which specifies the address of the user's data area. The buffer contains the data returned by IFMORE for the fields that are defined by the EDIT_SPEC parameter, described below. Specify a character string. **Note:** The data is placed in the data area from left to right. If parameters are being passed without dope vectors (IFSTRT language indicator = 2), the data is placed in the area as specified. No length checking is attempted because the Host Language Interface does not know the length of the data area.

Parameter	Description

EDIT_SPEC	[I,c,r] The edit specification is a required input parameter which defines the fields that are to be returned from the specified record. The specification describes the format of the data which is returned at the buffer location (see BUFFER above).
	Specify a character string using one of the following LIST, DATA, or EDIT format options:
	`LIST (fieldname list);`
	`DATA (fieldname list);`
	`DATA;`
	`EDIT (fieldname list) (edit formats);`
	`EDIT (fieldname list1) (edit format1) (fieldname list2) (edit format2);`
	where:
	fieldname list is required and specifies a field name or names. Specify elements in the field name list using one of the following options:

	• fieldname • fieldname (n) • fieldname(*) • fieldname(%variable) where: • *fieldname* retrieves the first occurrence of the named field. • *fieldname(n)* retrieves the nth occurrence of the named field. • *fieldname(*)* retrieves all occurrences of the named field in the order of occurrence. • *fieldname(%variable)* retrieves the occurrence of the field specified by the %VARBUF and %VARSPEC parameters. *edit format* is required in the EDIT specification and specifies a code or codes which indicate(s) the format of the data to be returned for the named field in the fieldname list-edit format pair. See page 217 for a detailed description of the EDIT format codes that are used with IFMOREX. See Chapter 7 for a description of LIST, DATA, and EDIT formatting.
TIME	[I,i,r] The time is a required input parameter. Specify an integer value which is the number of times to try the retrieval in the event of an enqueuing conflict; the wait time is in three-second periods.

Parameter	Description
MOREX_NAME	[I,s,r/o] The name of the IFMOREX compilation is an input parameter that is only required if using the Compiled IFAM facility (IFMORXE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string. Any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. A null value is equivalent to omitting the name parameter.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

IFMOREX performs the same operations as IFMORE. However, the record is enqueued upon in exclusive rather than in share status.

If a call to IFMOREX results in an enqueuing conflict, Model 204 waits at most three seconds and then tries again, for as many times as specified in the time parameter (wait time is in three-second periods).

After trying unsuccessfully for the number of times specified, Model 204 returns a completion code of 3 to the HLI program. For more information, see the description of the ENQRETRY parameter in the Rocket Model 204 documentation wiki:

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

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE        PIC 9(5) COMP SYNC.
    05 WORK-AREA-1    PIC x(256).
    05 WORK-AREA-2    PIC X(256).
•
•
•
01  SPEC-1            PIC X(64) VALUE
    'EDIT (SSN,NAME,BDATE,SCDATE,GRADE,STEP)
```

```
    (A(9),A(30),2A(6),2J(2));'.
01  SPEC-2            PIC X(34) VALUE
    'EDIT (STATUS,SALARY)(A(11),A(12));'.
01  TIME              PIC 9(5) COMP SYNC VALUE 5.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFGET" USING RETCODE, WORK-AREA-1, SPEC-1
  IF GRADE IS EQUAL TO 9 THEN PERFORM GETMORE.
•
•
•
GETMORE.
  CALL "IFMOREX" USING RETCODE, WORK-AREA-2, SPEC-2, TIME.
```

IFNFLD call *-mc,sc*

Function The IFNFLD call (RENAME FIELD) renames a field that has been defined as VISIBLE for a Model 204 file. IFNFLD requires Model 204 file manager privileges.

Full syntax (63)

```
IFNFLD(RETCODE,FIELD_NAMES,FILE_SPEC)
```

Compile-only syntax A compile-only form of IFNFLD is not available.

Execute-only syntax An execute-only form of IFNFLD is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FIELD_NAMES	[I,c,r] This is a required input parameter which specifies the name of an existing field and its new name. Specify the names as a character string using the following format: `oldname,newname;` where: *oldname* is required and specifies the existing name that is defined for the field. *newname* is required and specifies the replacement name for the field. The name must be unique within the context of the file where it will be stored. Specify the name as a character string, up to 255 characters in length. The name must begin with a letter and it can contain any alphanumeric character except the following: • At sign (@) • Pound sign (#) • Semi-colon (;) • Double question marks (??) • Question mark followed by a dollar sign (?$) • Question mark followed by an ampersand (?&) See the Rocket Model 204 documentation wiki for the detailed list of rules that apply to field names: http://m204wiki.rocketsoftware.com/index.php/Field_names

Parameter	Description
FILE_SPEC	[I,s,o] The file specification is an optional input parameter for use only with a multiple cursor IFSTRT thread for specifying the name of the Model 204 file that contains the field to be renamed. Specify the name of the file as a short character string variable using the following format: `IN [FILE] filename`
	The specified file must be open on the thread, otherwise the call is unsuccessful and Model 204 returns a completion code equal to 4.

Notes and tips

Use the IFNFLD call to rename a field in a Model 204 file, only for a field which is defined as VISIBLE. The IFNFLD call is valid only in file context, not for a group. The IFNFLD call is valid on *all* types of IFSTRT threads.

When FOPT=X'10' and the date/time stamp feature is installed, the IFNFLD function is supported for DTS files.

Note: The file context can change on a multiple cursor thread and, if the file specification parameter (FILE_SPEC) is omitted, IFNFLD renames the field for the default file on the thread.

If the field being renamed has a security level, the Model 204 file manager must have field level security (FLS) access privileges.

Completion return code (RETCODE)

The IFNFLD call is not successful under either of the following conditions:

- The old field name that is specified does not exist.
- The new field name that is specified cannot be added to the Model 204 dictionary for either of the following reasons:
  - The new name that is specified already exists.
  - Space cannot be allocated.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5)  COMP SYNC.
    05  NAMES    PIC X(20) VALUE 'SALARY,COMPENSATION;'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFNFLD" USING RETCODE, NAMES.
```

IFOCC call *-mc*

Function

The IFOCC call (COUNT OCCURRENCES) counts the number of occurrences of the specified field, or fields, in the current record and returns a count value for each field in an output parameter. IFOCC specifies the cursor for the current record.

Full syntax (122)

```
IFOCC(RETCODE,BUFFER,CURSOR_NAME,FIELD_LIST,OCC_NAME,
%VARBUF,%VARSPEC)
```

Compile-only syntax (123)

```
IFOCCC(RETCODE,CURSOR_NAME,FIELD_LIST,OCC_NAME)
```

Execute-only syntax (124)

```
IFOCCE(RETCODE,BUFFER,OCC_NAME,%VARBUF,%VARSPEC)
```

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
BUFFER	[O,c,r] The buffer location is a required output parameter which specifies the address of the user's data area. Specify a character string. The buffer contains the occurrence counter, or counters, returned by IFOCC for each field that is defined by the FIELD_LIST parameter, described on page 223. For each field specified, a four byte value is returned. Multiple count values are positioned in order corresponding to the location of the fields in the field list. If a specified field does not occur in the record, its counter is set to zero.
CURSOR_NAME	[I,c,r] Is a required input parameter, which specifies the name of the cursor that points to the current record for which the field count(s) is (are) performed. This is a character string, the name previously assigned to the cursor in a corresponding IFOCUR call. See CURSOR_NAME on page 229 for a description of the cursor name for the IFOCUR call.
FIELD_LIST	[I,c,r] The field specification is a required input parameter which defines the field, or fields, that are to be counted in the current record. Specify the name of a field as a character string. You must list at least one field to be counted in the current record. You may list additional fields by separating field names with a comma.
OCC_NAME	[I,s,o] The name of the IFOCC compilation is an optional input parameter. If specified, Model 204 saves the compilation using this name.

Parameter	Description
	Specify the name as unique, and as a short character string (maximum 32 characters). The first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). A null value is equivalent to omitting the name parameter, and is not valid.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFOCC call to count occurrences of fields in a record.

IFOCC requires a cursor to specify the current record, but otherwise operates similarly to the single cursor IFCTO call. See the IFCTO call.

The IFOCC call is the equivalent of the COUNT OCCURRENCES statement in SOUL in the host language multiple cursor environment. See the Rocket Model 204 documentation wiki for information about the COUNT OCCURRENCES statement:

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

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE      PIC 9(5) COMP SYNC.
    05  BUFFER       PIC X(8).
    05  CURSOR-NAME  PIC X(5) VALUE "CUR1;".
    05  FIELD-LIST   PIC X(12) VALUE "COLOR, MAKE;".
•
•
•
PROCEDURE DIVISION.
•
•
```

-
```
CALL "IFOCC" USING RETCODE, BUFFER, CURSOR-NAME,
                   FIELD-LIST.
```

IFOCUR call *-mc*

Function

The IFOCUR call (OPEN CURSOR) opens a cursor on the thread to a previously established found set.

Full syntax (95)

```
IFOCUR(RETCODE,CURSOR_SPEC,CURSOR_NAME,%VARBUF,%VARSPEC,
FILE_SPEC)
```

Compile-only syntax (106)

```
IFOCURC|IFOCRC(RETCODE,CURSOR_SPEC,CURSOR_NAME,FILE_SPEC)
```

Execute-only syntax (107)

```
IFOCURE|IFOCRE(RETCODE,CURSOR_NAME,%VARBUF,%VARSPEC)
```

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.

Parameter	Description

CURSOR_SPEC	[I,c,r] The cursor specification is a required input parameter which indicates an existing found set of records or values that was established by a previously compiled call. Specify the found set as a character string using one of the following formats:
	`{IN setname [ordering clause] | ON [list] listname [ordering clause] | OF fieldname [value set specification]}`
	where:
	IN setname specifies the compilation name for the previously compiled IFFIND, IFFNDX, IFFWOL, IFFAC, IFSORT, IFFDV, or IFSRTV call which established the set.
	ON listname specifies the name of a list of records. If the list does not exist, it is created in context of the default file or group.
	ordering clause is optional and specifies ordering criteria and is valid only for cursors opened against an unsorted record set. Ordering criteria may *not* be specified for cursors opened against a value set, or against a record set that is sorted (with IFSORT).
	Specify the ordering clause using only the IN ORDER clause for sorted file ordering, or the following format line in full for B-tree ordering:
	`IN [ASCENDING | DESCENDING] ORDER [BY [EACH} fieldname] [(FROM value1) (TO value2)] [LIKE pattern] [BY [ASCENDING | DESCENDING] RECORD]`
	where:
	ASCENDING and *DESCENDING* are mutually exclusive keywords that indicate the order in which the record set will be processed. ASCENDING order is the default.
	fieldname specifies the name of the field to be used for ordering the records.

Parameter	Description
	The *FROM* and *TO* clauses specify the range of values for fieldname, where *value1* specifies the beginning value, and *value2* specifies the ending value. You may specify the range using both FROM and TO, or using only FROM, for all values greater than or equal to, or only TO, for all values less than or equal to). **Note:** Specifying a range limits the selection of records to process. If a range is specified, records that do contain the field are not processed. *pattern* specifies a field value as a character string that is used to match against the record. Enclose the pattern string inside single quotation marks. **Note:** Specifying a pattern limits the selection of records to process. Only records which meet the pattern matching criteria are processed. See the Rocket Model 204 documentation wiki for a description of the valid pattern characters and examples of their use: http://m204wiki.rocketsoftware.com/index.php/Record_loops The *BY ASCENDING|DESCENDING RECORD* clause specifies how records within each Ordered Index value are processed by IFFTCH. IF BY DESCENDING RECORD is not specified, IFFTCH processes records in ascending order. Ordering clause options are similar to the ordering options available in a FOR EACH RECORD statement in SOUL. For IFOCUR, the WHERE|WITH retrieval conditions clause is not supported. See the Rocket Model 204 documentation wiki for information about the FOR EACH RECORD IN ORDER statement: http://m204wiki.rocketsoftware.com/index.php/Operations_on_multiply_occurring_fields *OF fieldname* specifies a value set specification that is compiled and saved as a part of the IFOCUR/IFOCURC compilation. At execution time (IFOCUR or IFOCURE) this specification is executed and a value set is created. A cursor then opens against this value set. Subsequent IFFTCH or IFFTCHE calls fetch the values from the value set. When the cursor is closed using IFCCUR, the value set empties. IFOCUR with OF fieldname is similar to the Model 204 FOR EACH VALUE OF statement. *value set specification* is optional and allows you to specify selection or ordering criteria against a value set. `[FROM value1] [TO value2] [[NOT] LIKE pattern] [IN [ASCENDING | DESCENDING] [NUMERIC | CHARACTER | RIGHT-ADJSTED] ORDER]`

Parameter	Description
	where: *FROM* and *TO* clauses are optional and specify a minimum (greater than or equal to) value (FROM), a maximum (less than or equal to) value (TO), or a range of values (FROM and TO) for selection criteria. *LIKE*|*NOT LIKE* clause is optional and specifies a string pattern for selection criteria. See the Rocket Model 204 documentation wiki for a description of the valid pattern characters and examples of their use: http://m204wiki.rocketsoftware.com/index.php/Record_loops *ASCENDING* and *DESCENDING* are mutually exclusive keywords that indicate the order in which the value set will be processed. ASCENDING order is the default. CHARACTER, RIGHT-ADJUSTED, and NUMERIC are mutually exclusive keywords. *CHARACTER* specifies values sorted in standard EBCDIC collating sequence. *RIGHT-ADJUSTED* specifies that values are temporarily right-justified before sorting so that shorter fields sort first. *NUMERIC* specifies a sort of number values with the usual numeric order relationships.
CURSOR_NAME	[I,s,r] The name to be assigned to the cursor is a required input parameter. Specify the cursor name as a short character string variable, from 1 to 32 characters in length. The cursor name must be unique, and must begin with a letter (A–Z or a–z) followed by one or more of the following characters: a letter, a digit (0–9), a period (.), or an underscore (_).
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified. **Note:** For IFOCUR, use these parameters to assign values to the %variables in the ordering clause before executing the call.

Parameter	Description
FILE_SPEC	[I,c,o/r] The file specification is a required input parameter only when using the *OF fieldname* form of the IFOCUR call. FILE_SPEC identifies the Model 204 file that will be updated to contain the new record. Specify the file as a character string using a standard Model 204 IN clause. See the Rocket Model 204 documentation wiki for information about the IN clause: http://m204wiki.rocketsoftware.com/index.php/Basic_SOUL_statements_and_commands#IN_clauses

Notes and tips

Use the IFOCUR call to open a cursor to a record or a value in a set which has been named on a thread. You may open more than one cursor against the same named set to maintain different positions within the set. You may also open several cursors against several different record sets.

When executing an IFOCURC (compile-only) call, Model 204 maps the cursor to the specified named set. The initial cursor position is located prior to the first record or value in the set. If ordering criteria is specified, Model 204 also validates the ordering criteria.

When executing an IFOCURE (execute-only) call, Model 204 performs the following actions:

1. Resets the cursor position prior to the first record or value in the set.
2. Assigns percent variables (%variable) in the ordering clause, for any FROM/TO conditions or ordered field names that are specified.

Using *OF fieldname* to create value set specifications

The *OF fieldname* form of IFOCUR allows you to use IFOCUR to create a value set without first calling either IFFDV or IFSRTV. If *OF fieldname...* is specified, an IFOCUR/IFOCURC call does not reference a previously compiled HLI call. Instead, a value set specification is compiled and saved as a part of the IFOCUR/IFOCURC compilation. At execution time this specification is executed and a value set is created. A cursor is then opened against this value set. Subsequent IFFTCH or IFFTCHE calls will fetch the values from the value set. When the cursor is closed using IFCCUR, the value set is emptied.

Choosing IFOCUR or IFFDV and IFSRTV

IFOCUR with the OF fieldname clause performs much the same function as IFFDV. The main difference is that IFFDV creates a value set (and if IFSRTV is called, sorts it) and stores it in CCATEMP while IFOCUR does not use CCATEMP space to save the compilation. We suggest the following:

- The IFFDV call is more efficient if you are fetching through the same value set multiple times. This is especially true if you have a large value set or are sorting the values using the IFSRTV call.

- The IFOCUR call with the *OF fieldname* clause is more efficient if you use this value set only once.

The IFOCUR call with the OF fieldname clause is similar to the Model 204 FOR EACH VALUE OF statement. IFFDV and IFSRTV are similar to the FDV (FIND ALL VALUES) and SORT VALUES statements. See the IFFDV call on page 161 and the IFSRTV call on page 285 for more information about those calls.

See the Rocket Model 204 documentation wiki for information about doing efficient value retrievals:

http://m204wiki.rocketsoftware.com/index.php/Basic_SOUL_statements_and_commands#Find_statement

Record number processing for ORDERED fields

With IFOCUR, Ordered Index records can be processed in either ascending or descending order. Use the BY ASCENDING | DESCENDING RECORD option on the IFOCUR call to specify how you want records within each Ordered Index value to be processed by IFFTCH.

- The first two columns of Table 6-4 show how IFFTCH processes records within the Ordered Index value when IN DESCENDING ORDER is specified and no record order is specified (or BY ASCENDING RECORD is specified).

  Note: If no record order is specified, BY ASCENDING RECORD is assumed.

- The last two columns of Table 6-4 show how IFFTCH processes the records within Ordered Index value when IN DESCENDING ORDER and BY DESCENDING RECORD are both specified.

Table 6-4. IFFTCH processing of records within an Ordered Index

When either no record order or BY ASCENDING ORDER is specified ...		When IN DESCENDING ORDER and BY DESCENDING RECORD both are specified ...	
LAST_NAME	**RECNO**	**LAST_NAME**	**RECNO**
SMITH	08	SMITH	09
SMITH	09	SMITH	08
MARTIN	05	MARTIN	07
MARTIN	06	MARTIN	06
MARTIN	07	MARTIN	05
JONES	01	JONES	04
JONES	02	JONES	03

Table 6-4. IFFTCH processing of records within an Ordered Index

When either no record order or BY ASCENDING ORDER is specified ...		When IN DESCENDING ORDER and BY DESCENDING RECORD both are specified ...	
LAST_NAME	**RECNO**	**LAST_NAME**	**RECNO**
JONES	03	JONES	02
JONES	04	JONES	01

Rules for cursor processing

The following rules apply to cursor processing:

- You can only open a cursor to a record, value set, or list that has been established (named) by the saved compilation of one of the following calls: IFFIND, IFFNDX, IFFWOL, IFFAC, IFSORT, IFFDV, or IFSRTV.
- You cannot modify a record set that is established by an IFFIND, IFFNDX, IFFWOL, IFFAC, IFFDV, IFSORT, or IFSRTV while a cursor is open against it.

  See the IFFIND, IFFAC, IFSORT, IFFDV, and IFSRTV calls.
- You can issue calls that modify a list while a cursor is open in it. Note, however, that a record that is added to a list while a cursor is open may or may not be processed, depending on its position relative to the current cursor location.
- You can successfully issue the IFOCUR call to be executed only for a cursor that is in a “closed” state. A cursor is in a closed state before it is first allocated in response to an IFOCUR call and thereafter, after an IFCCUR call is executed.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE      PIC 9(5) COMP SYNC.
    05  FDSPEC       PIC X(12) VALUE "MAKE='FORD';".
    05  DNAME        PIC X(7) VALUE "FDFORD;".
    05  CURSPEC      PIC X(10) VALUE "IN FDFORD;".
    05  CURSOR-NAME  PIC X(7) VALUE "CRFORD;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFFIND" USING RETCODE, FDSPEC, FDNAME.
  CALL "IFOCUR" USING RETCODE, CURSPEC, CURSOR-NAME.
```

Note: This example illustrates a key concept in multiple cursor processing. In the example, IFOCUR opens a cursor (named CRFORD) to a found set (saved as FDFORD) that was established by the IFFIND call. Alternatively, the found set could have been established by an IFFNDX, IFFWOL, IFFDV, IFFAC, IFSORT, or IFSRTV call. In any case, a cursor can only reference an existing found set or a list, one that was established by a previously compiled call.

IFOPEN call *-mc,sc*

Function The IFOPEN call (OPEN) opens the specified file or group.

Full syntax (11) `IFOPEN(RETCODE,FILE_SPEC)`

Compile-only syntax A compile-only form of IFOPEN is not available.

Execute-only syntax An execute-only form of IFOPEN is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value. See page 237.

Parameter	Description

FILE_SPEC	[I,c,r] The file specification is a required parameter which specifies the file or group to be opened. Specify a character string using the following format:
	`[FILE | GROUP] name [,deferred ddname];`
	`[deferred ddname2];`
	`[password[:newpassword]];`
	where:
	FILE or *GROUP* is the keyword which specifies that either a file or a group is to be opened. The keyword is optional. If you do not specify FILE or GROUP, Model 204 attempts to find a group with the specified name before searching for a file.
	name is required and specifies the name of the file or group to be opened. This name corresponds to a single file name or to a permanent group definition.
	deferred ddname is optional and specifies the name of the sequential data set used for deferred updates. If specified, this name must correspond to a DD statement in the JCL of the Host Language Interface/Model 204 service program. When you specify a deferred ddname, to select the deferred form of file maintenance, file maintenance functions are divided into two phases, as described in “Notes and tips” below. If the deferred ddname is specified, insert a comma separator immediately following the file name.
	Note: The deferred update option is not available when a group open is performed. See “Notes and tips” below for more information about opening files in deferred update mode.
	deferred ddname2 is optional and specifies the name of a second data set in which deferred updates to the file are to be stored. This data set is for variable-length records.
	password is optional and specifies the user’s identification which allows access to the file or group. You can omit the password for public and semipublic files and groups.

	Note: The update indicator in the IFSTRT function for IFAM2 or IFAM4 takes precedence over the password in IFOPEN. If the IFSTRT update indicator is 0, no updates are allowed, regardless of the password. If the IFSTRT update indicator is 1, the password indicates the privileges given. *new password* is an optional character string that changes the login password for the specified user, for future logins. Specify a new password if you need to replace the existing password. Note that changing passwords at open time requires special login privileges. If the new password is specified, insert a colon separator immediately following the password.

Notes and tips Use the IFOPEN call before any data records are accessed to open a file or group. When IFOPEN is called, the named file or group becomes the current file or group.

On a single cursor IFSTRT thread, a file or group named in a previous call to IFOPEN is no longer available to the thread. On a multiple cursor IFSTRT thread, files or groups named in a previous IFOPEN call remain accessible.

If the file or group has already been opened by another thread or user, complete open processing is not performed by Model 204 and the file or group is made available to the new user.

Files and groups that are no longer needed should be closed explicitly with IFCLOSE to lessen memory requirements. However, IFCLOSE causes all server tables to be reinitialized, and saved compilations are lost.

Opening files in deferred update mode

When you specify a deferred data set name to select the deferred form of file maintenance, file maintenance functions are divided into the following two phases:

1. The first phase updates records in the data area and is completed immediately.
2. Information for the second phase, the index update, is written onto the deferred update data set. Subsequent job steps must be run to complete phase two.

   See the Rocket Model 204 documentation wiki for information about job steps required for phase two:

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

Note: The deferred update mode can produce unexpected results if retrieval is attempted before phase two has been completed because the record updates are not yet reflected in the index. Deferred updates are used to speed processing when major file maintenance is being performed.

The deferred update option is not available when a group open is performed. However, prior to the group open, the HLI program can open selected files individually in deferred update mode, and the following conditions apply:

- All updates to those files, even in group context, are written to the appropriate deferred data sets.
- Each deferred update file in the group must have its own deferred update data set, and phase two must be run for each file.

Completion return code (RETCODE)

IFOPEN returns the value (RETCODE) of the file's FISTAT parameter. In a group context, the file, or files, from which a nonzero FISTAT is returned cannot be determined. The settings of FISTAT are listed below.

Code	Condition
0	File is normal.
1	File is not initialized.
2	File is physically inconsistent.
8	File is full.
16	File has been recovered.
32	File is in deferred update mode.
64	File may be logically inconsistent.
260	The file or group could not be opened. A journal message further explains the problem.
325	IQB not large enough. Adjust the MODEL 204 parameters LIBUFF, LOBUFF, and IFAMBS as necessary.

Note: If two or more FISTAT values are appropriate simultaneously, Model 204 returns the sum of these values.

Note the following conditions when evaluating IFOPEN completion codes:

- A nonzero completion code does not necessarily indicate a problem with the file, for example:
  - A code of 0 or 16 indicates that file or group processing can proceed without further conditional testing. Any other return code indicates a potential problem and should be tested further.
  - Codes 32 and 16 do not indicate broken files.
- The logically inconsistent condition (code 64) does not prevent use of the file. It is set by a soft restart of an updating request and indicates that data relationships that are maintained by the application may not be complete.
- We suggest that if you get a completion code that is not listed above, you first check Chapter 8 to see if the code is listed. If it is not, the completion code is the sum of the codes listed above.

FISTAT codes are examined in subsequent calls

The Host Language Interface examines the FISTAT setting before any function is allowed to operate. After IFOPEN, any operation on a broken file except IFRPRM, results in a completion code of 61, which indicates that the function was rejected. Note that although privileged users may reset the FISTAT parameter, subsequent operations on the file may cause further damage or unpredictable results.

Opening a file or group during system recovery

During system recovery, a file or group that is not participating in the system recovery can be opened.

However, updates are not allowed, and the file or group is opened with read-only privileges. The IFOPEN returns a normal completion code, but the first function call that tries to update the file or group returns a completion code of 40. To update, reissue the IFOPEN after system recovery has been completed. You can call IFEPRM to test the CURPRIV parameter for update privileges before attempting the update.

Coding examples (COBOL)

The COBOL coding example below opens a file and a group using the IFOPEN call:

- A file called FILEA with a password of MYSECRET. (In this example, the group name FILEA does not exist.)
- A group called WEEK without a password.

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE       PIC 9(5) COMP SYNC.
    05 FILE-INFO     PIC X(15) VALUE ‘FILEA;MYSECRET;'.
    05 GROUP-INFO    PIC X(12) VALUE ‘GROUP WEEK;;'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFOPEN" USING RETCODE, FILE-INFO.
•
•
•
  CALL "IFOPEN" USING RETCODE, GROUP-INFO.
•
•
•
```

IFOPENX call *-mc,sc*

Function

The IFOPENX call (OPEN EXCLUSIVE) opens the specified file or group, enqueuing in exclusive mode.

Full syntax (35)

```
IFOPENX|IFOPNX(RETCODE,FILE_SPEC,TIME)
```

Compile-only syntax

A compile-only form of IFOPENX is not available.

Execute-only syntax

An execute-only form of IFOPENX is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value.
FILE_SPEC	[I,c,r] The file specification is a required parameter which specifies the file or group to be opened. Specify a character string. See FILE_SPEC on page 235 for a detailed description of the file specification used for IFOPEN that is also valid for IFOPENX.
TIME	[I,i,r] The time is a required parameter which specifies the wait time (in seconds). Specify an integer value.

Notes and tips

Use the IFOPENX call before any data records are accessed to open a file or group. When IFOPENX is called, the named file or group becomes the current file or group. IFOPENX performs the same operations as IFOPEN, however, IFOPENX enqueues on the file in exclusive rather than in share status.

IFOPENX prevents any other user from accessing the file until the current user closes it. If you issue IFOPENX for a group, all files in the group are enqueued upon in exclusive status and the group itself is enqueued upon in share status. See the Rocket Model 204 documentation wiki for more information about enqueuing:

http://m204wiki.rocketsoftware.com/index.php/Transaction_back_out#Concurrency_control_and_locking_mechanisms

If the first attempt to enqueue on a file fails, Model 204 waits the number of seconds specified in the time parameter, then tries again. If the second attempt fails, Model 204 returns a completion code of 3 to the HLI program.

Coding examples (COBOL)

The COBOL coding example below opens a file using the IFOPENX call. The file is called FILEB and requires a password of OURSECRET. (In this example, the group name FILEB does not exist.)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE      PIC 9(5) COMP SYNC.
    05 FILE-INFO    PIC X(15) VALUE ‘FILEB;OURSECRET;'.
   05 TIME          PIC 9 COMP SYNC VALUE 3.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFOPENX" USING RETCODE, FILE-INFO, TIME.
•
•
•
```

IFPOINT call *-sc*

Function

The IFPOINT call (POINT) changes the current record number and, in group context, optionally changes the current file.

Full syntax (44)

```
IFPOINT|IFPNT(RETCODE,REC_NUM,FILE_SPEC)
```

Compile-only syntax

A compile-only form of IFPOINT is not available.

Execute-only syntax

An execute-only form of IFPOINT is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
REC_NUM	[I,c,r] The record number is a required input parameter which specifies the number of a record from a previously found set. Specify the number as a character string. Upon successful completion of IFPOINT, the record number specified becomes the current record number.
FILE_SPEC	[I,s,r/o] The file specification is an input parameter that is required in group context to specify the name of a file which is a member of the open group. In individual file context, the file name is optional and, if specified, must specify the name of the current file. You may specify the following $functions in the place of a known file name string constant: $CURFILE for the current file of the group, or $UPDATE for the group update file.

Notes and tips

Use IFPOINT to change the current record. Upon successful completion, the record number specified becomes the current record number and is the next record to be operated upon by an IFMORE or IFPUT function.

IFPOINT does not affect the current record set and does not enqueue the new current record. The new current record is enqueued when it is operated on by IFMORE and IFPUT. The old current record is dequeued by the next IFMORE, IFPUT, or IFGET.

In a group context, IFPOINT can change both the current record and the current file. Be sure that the record number and the file name are consistent.

In files that contain record security, IFPOINT does not allow you to retrieve records lacking an appropriate record security field value.

Coding examples (COBOL)

The COBOL example below, which takes place in an individual file context, sets CURREC to 175.

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE    PIC 9(5) COMP SYNC.
    05 RECNO      PIC X(4) VALUE ‘175;’.
•
•
•
PROCEDURE DIVISION.
•
•
•
    CALL "IFPOINT" USING RETCODE, RECNO.
```

The COBOL example below, which takes place in a file group context, sets CURREC to 0 (first record) for file PAYROLL.

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE   PIC 9(5) COMP SYNC.
    05 RECNO     PIC XX VALUE ‘0;’.
    05 FILE-NAM  PIC X(8) VALUE ‘PAYROLL;’.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFPOINT" USING RETCODE, RECNO, FILE-NAM.
```

The COBOL example below, which takes place in a group context, sets CURREC to 0 which is the first record.

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE   PIC 9(5) COMP SYNC.
    05 RECNO     PIC XX VALUE ‘0;’.
    05 FILE-NAM  PIC X(8) VALUE ‘$UPDATE;’.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFPOINT" USING RETCODE, RECNO, FILE-NAM.
```

IFPROL call *-mc,sc*

Function

The IFPROL call (PLACE RECORD ON LIST) places the current record on the specified list.

Full syntax (33)

```
IFPROL(RETCODE,LIST_NAME,CURSOR_NAME)
```

Compile-only syntax

A compile-only form of IFPROL is not available.

Execute-only syntax

An execute-only form of IFPROL is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LIST_NAME	[I,c,r] The list name is a required input parameter which specifies the name of the list to be used. If the list does not already exist, Model 204 creates a new list using this name. Specify a character string variable which is the name of a list, existing or new.
CURSOR_NAME	[I,s,r] The name of the cursor is an input parameter that is available only for use with a multiple cursor IFSTRT thread and is required for specifying the current record to be placed on the list. Specify the cursor name as a short character string (maximum 32 characters), using the name previously assigned to the cursor in a corresponding IFOCUR call. See CURSOR_NAME on page 226 for a description of the cursor name for the IFOCUR call. **Note:** The cursor name is not a valid parameter for use with a single cursor IFSTRT thread.

Notes and tips

Use the IFPROL call to add a record to a list if the record is not already on the list. You may use IFPROL to add records from a sorted set to a list.

The IFPROL call is valid on *all* types of IFSTRT threads. On a multiple cursor IFSTRT thread, you must specify the cursor whose current record is to be placed on the list. On a single cursor IFSTRT thread, IFPROL places the current record on the list.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE    PIC 9(5) COMP SYNC.
    05 LISTNAME   PIC X(11) VALUE "NEWEMPLOYS;".
•
•
```

```
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFPROL" USING RETCODE, LISTNAME.
```

In this example, processing is done on a single cursor IFSTRT thread, and the IFPROL call is preceded by IFFIND and IFGET calls (not shown).

IFPROLS call *-mc*

Function

The IFPROLS call (PLACE RECORDS ON LIST) places a set of found records on a list. IFPROLS creates a list or references an existing list on the current thread.

Full syntax (109)

```
IFPROLS|IFPRLS(RETCODE,LIST_SPEC)
```

Compile-only syntax

A compile-only form of IFPROLS is not available.

Execute-only syntax

An execute-only form of IFPROLS is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LIST_SPEC	[I,c,r] The list specification is a required input parameter which specifies an existing set of found records to be added to a list. Specify the records and the list as a character string using the following format: `set qualifier ON [LIST] listname` where: *set qualifier* is required; specify the setname using the IN label clause, where label is the name of a saved IFFIND, IFFNDX, IFFWOL, IFFAC, IFSORT, IFFDV, or IFSRTV compilation from a previously compiled call. *listname* is required and specifies the name of a particular list. If the list does not exist, it is created.

Notes and tips

Use the IFPROLS call to add records to a list.

The IFPROLS call is the equivalent of the PLACE RECORDS ON LIST statement in SOUL in the host language multiple cursor environment. See the Rocket Model 204 documentation wiki for information about the PLACE RECORDS ON LIST statement:

http://m204wiki.rocketsoftware.com/index.php/DML_statements_in_Parallel_Query_Option/204

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE    PIC 9(5) COMP SYNC.
    05  LISTSPEC   PIC X(20) VALUE "IN PREVFD ON LIST L;".
```

```
•
•
•
PROCEDURE DIVISION.

•
•
•
   CALL "IFPROLS" USING RETCODE, LISTSPEC.
```

IFPUT call *-sc*

Function The IFPUT call (PUT) updates the current record.

Full syntax (18)
```
IFPUT(RETCODE,BUFFER,EDIT_SPEC,PUT_NAME,%VARBUF,%VARSPEC)
```

Compile-only syntax (54)
```
IFPUTC(RETCODE,EDIT_SPEC,PUT_NAME)
```

Execute-only syntax (55)
```
IFPUTE(RETCODE,BUFFER,PUT_NAME,%VARBUF,%VARSPEC)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
BUFFER	[I,c,r] The buffer location is a required input parameter which specifies the address of the data that will be used to update the fields defined in the EDIT_SPEC parameter. Specify a character string variable.

EDIT_SPEC	[I,c,r] The edit specification is a required input parameter which defines the fields that are to be returned from the specified record. The specification describes the format of the data which is returned at the buffer location (see BUFFER above).
	Specify a character string using one of the following LIST, DATA, or EDIT format options:
	`LIST (fieldname list);`
	`DATA;`
	`EDIT (fieldname list) (edit formats);`
	where:
	fieldname list is required for the LIST or EDIT specification and specifies a field name or names. Specify elements in the field name list using one of the following options:
	• fieldname
	• fieldname(n)
	• fieldname(*)
	• fieldname(%variable)
	• fieldname(+n)
	• fieldname(+%variable)
	where:
	• *fieldname* updates the first occurrence of the named field. Note that this is equivalent to fieldname(1). If the field does not occur in the current record, IFPUT adds it.

Parameter	Description
	• *fieldname(n)* updates the nth occurrence of the named field for a multiply occurring field. If the nth occurrence does not exist in the current record, IFPUT adds it. • *fieldname(*)* adds the named field to the current record. If the field already exists in the current record, IFPUT adds another occurrence of the field. • *fieldname(%variable)* retrieves the occurrence of the field specified by the %VARBUF and %VARSPEC parameters. If the nth occurrence does not exist in the current record, IFPUT adds it. • *fieldname(+n)* inserts a new occurrence of the named field to the current record. This is analogous to the SOUL INSERT statement and is useful for adding new occurrences of a field where the order of the values is important. Insert the new occurrence as the nth occurrence. • *fieldname (+%variable)* inserts a new occurrence of the field into the current record. The occurrence number is retrieved from the %VARBUF and %VARSPEC parameters. **Note:** If there is a current nth (or %variable) occurrence, make it the one after the nth occurrence. If n is greater than the current number of occurrences, add the new occurrence at the end. If the field does not occur in the current record, add it. If n is 0 or is not specified, treat it as though n=1 and insert the field as the first occurrence. *edit formats* is required in the EDIT specification and specifies a code or codes which indicate(s) the format of the data to be returned for the named field in the field name list-edit format pair. See “Using EDIT format codes for an updating call” on page 328 for a detailed description of the EDIT format codes that are used with IFPUT. **Note:** See Chapter 7 for a description of LIST, DATA, and EDIT formatting.
PUT_NAME	[I,s,r/o] The name of the IFPUT compilation is an input parameter that is only required if using the Compiled IFAM facility (IFPUTC and IFPUTE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string. Any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. A null value is equivalent to omitting the name parameter.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation

Parameter	Description
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use IFPUT to update the current record. IFPUT changes or deletes fields in an existing record and adds new fields to an existing or newly created record. Typically, you issue the IFPUT call after an IFGET, IFBREC, or IFPOINT function.

When FOPT=X'10' and the date/time stamp feature is installed, the IFPUT function is *not* supported for DTS files.

You cannot update the sort key field in a sorted file or the hash key field in a hashed file. In these cases, the old record must be deleted and a new record built. The new record contains the updated sort or hash key field.

You can use IFPUT to add INVISIBLE fields to the current record. To change an INVISIBLE field, first call IFDVAL to delete the field. Then, use IFPUT to add the new value. Also, use IFDVAL to delete an INVISIBLE field or to delete a particular occurrence of a field when the occurrence order is not known. You can also use IFPUT with the G edit format to delete field occurrences. See "Using EDIT format codes for an updating call" on page 328 for information about using the G edit format.

To use IFPUT to delete fields, update the field with a null value.

Completion return code (RETCODE)

If the IFPUT call is unsuccessful, Model 204 returns the following completion codes:

Code	Error condition
4	An undefined field is encountered in a precompiled field name list. The IFPUT updating operation for the current call stops (error message begins with CANCELLING CALL). **Note:** If the IFPUT specification is not precompiled, an undefined field name prevents all updating for the current call.
10	Model 204 encountered invalid data values for BINARY and FLOAT numeric field for a file having FILEMODL set to NUMERIC VALIDATION.
200	A uniqueness violation has occurred (field level constraint).
202	An AT_MOST_ONE violation has occurred (field level constraint).

See the Rocket Model 204 documentation wiki for information about field level constraints:

http://m204wiki.rocketsoftware.com/index.php/Field_design#Field_and_content_constraints

http://m204wiki.rocketsoftware.com/index.php/Field_attributes#AT-MOST-ONE_versus_UNIQUE_attributes

Also see the documentation wiki for information about BINARY and FLOAT field values:

http://m204wiki.rocketsoftware.com/index.php/Data_maintenance#Storing_data_in_fields

Updating with UPDATE IN PLACE fields

Field values are read from left to right from the data area to correspond to names in the field name list. Changes to existing fields are applied to a record one field at a time. If the UPDATE IN PLACE attribute has been specified for a field (see IFDFLD), then the order of the fields is not affected when the contents of the field are changed.

When many occurrences of an UPDATE IN PLACE field are being changed, a subscripted field name should appear in the IFPUT name list for each value to be changed. If a field name appears more than once with the same or no subscript, the specified occurrence is changed many times and all the other occurrences remain unchanged.

Updating with UPDATE AT END fields

If you specify UPDATE AT END for a field, as each occurrence is updated the old occurrence is deleted and the new one is added following other occurrences of the same field. New fields are added to the end, regardless of the update option in effect. Deletions always preserve the order of the remaining occurrences.

For example, suppose that you specify UPDATE AT END and a record appears as:

```
LAST NAME=SMITH CHILD=TOM CHILD=DICK CHILD=HARRY
```

Then, if the field name list specifies (CHILD(1), CHILD(2)), the system first updates CHILD=TOM and the record appears as:

```
LAST NAME=SMITH CHILD=DICK CHILD=HARRY CHILD=THOMAS
```

The field list specification then directs Model 204 to change the second occurrence of CHILD, which is now CHILD=HARRY. The result is:

```
LASTNAME=SMITH CHILD=DICK CHILD=THOMAS CHILD=HAROLD
```

Specifying (CHILD,CHILD), which is equivalent to (CHILD(1), CHILD(1)), would have changed CHILD=TOM and then CHILD=DICK in the original record.

Coding example (COBOL)

If the current record contains the following information:

```
NAME=SMITH,SALARY=15000,SCHOOL=HIGH
```

and the work area contains the following information (in most applications, the values would be moved into the work area and not set up by VALUE clauses):

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05  RETCODE    PIC 9(5) COMP SYNC.
01  WORKAREA.
    05  NAME1       PIC X(5) VALUE ‘SMITH’.
    05  SCHOOL1     PIC X(7) VALUE ‘COLLEGE’.
    05  SALARY1     PIC 9(5) VALUE ’16000’.
    05  AGE         PIC 9(2) VALUE ‘29’.
•
•
•
    05  EDITLIST    PIC X(54) VALUE
        ‘EDIT(NAME,SALARY,SCHOOL(*),AGE)
        (A(20),J(6),A(7),J(2));’.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFPUT" USING RETCODE, WORKAREA, EDITLIST.
```

The current record is updated. If SALARY is UPDATE IN PLACE, the current record appears after the changes as:

```
NAME=SMITH,SALARY=16000,SCHOOL=HIGH,SCHOOL=COLLEGE,AGE=29
```

IFREAD call *-di*

Function The IFREAD call (READ) obtains a line of output from Model 204.

Full syntax (9)

```
IFREAD(RETCODE,LINE_AREA,MSG_DESC,LINE_LEN)
```

Compile-only syntax A compile-only form of IFREAD is not available.

Execute-only syntax An execute-only form of IFREAD is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required output parameter. The code is a binary integer value.
LINE_AREA	[O,c,r] The line area is a required output parameter which is set to the output line from Model 204. **Note:** If Model 204 generates a logical line of output which is longer than a connection's line size, the logical line is broken into two or more physical lines that do not exceed the connection's line size. A hyphen (-) is placed in the continuation character position of each continued line and a completion code of 2 is returned until the entire logical line has been transmitted. For example, suppose that the line size was set to 4 and Model 204 is to print the string ABCDEFGH. Three calls to IFREAD are required, as summarized below: Programs using a language indicator of 2, for example, COBOL and FORTRAN, must blank out the line area before each call to IFREAD.

Parameter	Description	
MSG_DESC	[O,i,o] The message descriptor is an optional output parameter. Model 204 returns the descriptor as an integer value in the following format:	
	Bytes 0 and 1	Contain the message length +4, in binary.
	Byte 2	Is a bit string that describes the message returned in LINE_AREA.
	Byte 3	Reserved
	Note: Since the bits can be used in combination, host language programs should test the bits individually. The bits for byte 2 are summarized below.	
	Bit 0 (X'80')	If one, output is a class E message.
	Bit 1 (X'40')	If one, output begins a new page.
	Bit 2 (X'20')	If one, user is being restarted.
	Bit 3 (X'10')	If one, output is a nondisplay input prompt.
	Bit 4 (X'08')	If one, output is a $READ prompt.
	Bit 5 (X'04')	If one, output is a class I message or any other special message such as a broadcast or file open message.
	Bits 6 and 7	Reserved
LINE_LEN	[I,i,o] The line length is an optional input parameter which specifies the transfer length for IFREAD. This parameter determines the maximum line length for the IFREAD call. If this parameter is present, it overrides any value specified on IFDIAL or IFDIALN. For PL/1, the length is the minimum of this value plus the string length. See page 254 for more information about the IFREAD line length.	

Notes and tips

Use the IFREAD call only with an IFDIAL connection to receive data from Model 204.

See the *Rocket Model 204 Host Language Interface Programming Guide* for an coding example of a subroutine that translates the message descriptor (MSG_DESC) values returned by Model 204.

You may specify a different buffer length with each call by specifying the line length in IFREAD. See the next page for detailed information about the IFREAD line length.

Completion return code (RETCODE)

Code your IFDIAL application to check the return code for the following values:

Code	Required action
1	Call IFWRITE next to provide Model 204 with input.
2	Call IFREAD next to get more output from Model 204.

See the *Rocket Model 204 Host Language Interface Programming Guide* for more information about coding IFDIAL applications.

If the IFREAD call is unsuccessful, Model 204 returns an error code for either of the following error conditions:

Code	Condition
12	No output ready from Model 204 call IFWRITE to provide more input. In this case, the answer area is not altered. IFREAD call not accepted (IFWRITE call expected).
100	No current Model 204 connection exists or the connection is lost.

Transfer length for output from Model 204

The parameters in effect during the execution of the IFREAD call determine the length of data transferred from Model 204 and the actions taken when the receiving area is a different size than the Model 204 output line.

The following factors determine the length:

The first factor is PL/1 string length; for PL/1 compilers (F-level, Optimizer and Checkout) use a dope vector when passing character string arguments. This dope vector contains the maximum length of the string and its address. For strings declared as VARYING, it also contains the current length.

Additionally, the transfer length for output from Model 204; this value is based on the following order of precedence, from highest to lowest:

1. The optional length parameter in the IFREAD call. The HLI application program may optionally specify a buffer length in the IFREAD call (see the LINE_LEN parameter on page 253) which is different from the CRAM buffer size returned by the IFDIAL or IFDIALN call.

   For PL/1, if this length is greater than the maximum string length, the maximum string length is used.

   Note: This value is in effect only for this specific IFREAD call.

2. The optional default length parameter in the IFDIAL call; this new default remains in effect until IFHNGUP is called.

For PL/1, if this length is greater than the maximum string length, the string length is used.

3. The standard default length, which is set relative to the host language (that is, the language indicator parameter specified in the IFDIAL or IFDIALN call); Model 204 uses either of the following lengths:
   – 252 for COBOL, FORTRAN, and Assembler
   – PL/1 maximum string length (dope vector)

Note that the maximum length of a data area that can be transferred over an IFDIAL thread is 32763 bytes. See the Rocket Model 204 documentation wiki for more information about buffer size parameters:

http://m204wiki.rocketsoftware.com/index.php/Defining_the_Runtime_Environment_(CCAIN)

When transfer length differs from output line size

The maximum length of data that can be transferred is a result of the transfer length. If this value is different than the actual Model 204 output line size, truncation or padding may occur.

Note: For all languages, when the output transfer length is less than the Model 204 output line size, Model 204 truncates the data.

However, if the transfer length is greater than the output line size, Model 204 performs either of the following actions:

- For COBOL, FORTRAN, and PL/1 fixed string areas, pads the remaining area with spaces.
- For PL/1, Model 204 sets the current string length to the output line size. The remaining answer area is unchanged.

Overview of IFREAD data transfer

Table summarizes the relationship between the parameters that determine the IFREAD data transfer length and the actions taken by Model 204.

Table uses the following codes:

- *Lang=n*, which is the language indicator specified in the IFDIAL or IFDIALN call.
- *LENGTH1*, which is the default length parameter in the IFDIAL call.
- *LENGTH2*, which is the length parameter in the IFREAD call.
- *FIXED*, which is a PL/1 string argument that is declared as fixed.
- *VARYING*, which is a PL/1 string argument that is declared as varying.
- *MAXLEN*, which is the maximum length of the PL/1 string.

- *Pad*, which indicates that Model 204 pads the remaining answer area with spaces.
- *Set length*, which indicates that PL/1 string length field (CURRLEN) is set to the Model 204 output line size, and the remaining answer area is unchanged.

Table 6-5. IFREAD data transfer length

Parameters in effect				Transfer length	Action
IFDIAL		**IFREAD**			
Lang=1	LENGTH1	LENGTH2		min(LENGTH2,MAXLEN)	Set length
	LENGTH1	¬LENGTH2		min(LENGTH1,MAXLEN)	
	¬LENGTH1	LENGTH2		min(LENGTH2,MAXLEN)	
	¬LENGTH1	¬LENGTH2		MAXLEN	
Lang=2	LENGTH1	LENGTH2		LENGTH2	Pad
	LENGTH1	¬LENGTH2		LENGTH1	
	¬LENGTH1	LENGTH2		LENGTH2	
	¬LENGTH1	¬LENGTH2		252	
Lang=3	LENGTH1	LENGTH2,	FIXED	min(LENGTH2,MAXLEN)	Pad
	LENGTH1	¬LENGTH2,	FIXED	min(LENGTH1,MAXLEN)	
	¬LENGTH1	LENGTH2,	FIXED	min(LENGTH2,MAXLEN)	
	¬LENGTH1	¬LENGTH2	FIXED	MAXLEN	
Lang=3	LENGTH1	LENGTH2,	VARYING	min(LENGTH2,MAXLEN)	Set length
	LENGTH1	¬LENGTH2,	VARYING	min(LENGTH1,MAXLEN)	
	¬LENGTH1	LENGTH2,	VARYING	min(LENGTH2,MAXLEN)	
	¬LENGTH1	¬LENGTH2,	VARYING	MAXLEN	

Legend:

LENGTHn	Parameter was specified
¬LENGTHn	Parameter was not specified
min(l,m)	Transfer length is determined to be the minimum value of l and m

Coding example (Assembler)

```
•
•
•
  CALL IFDIAL...
•
•
•
  CALL IFWRITE...
•
•
•
  MVC     RLEN(4),=F'30'
  CALL    IFREAD,(RETCODE,INPUT,MDF,RLEN),VL
  CLI     MDF+2,X'14'  DO WE NEED A PASSWORD
```

```
•
•
•
RETCODE    DC   F'0'
INPUT      DC   CL30' '
MDF        DS   0F
MDFLEN     DS   H
MDFTYPE    DS   BL2      X'80'=CLASS E ERROR MESSAGE
*                        X'40'=NEW PAGE
*                        X'20'=RESTART
*                        X'10'=PASSWORD PROMPT
*                        X'08'=$READ PROMPT
*                        X'04'=CLASS I MESSAGE
RLEN        DC  F'0'
  END
```

IFRELA call *-mc*

Function

The IFRELA call (RELEASE ALL RECORDS) releases all record sets held by the current thread.

Full syntax (104)

```
IFRELA(RETCODE)
```

Compile-only syntax

A compile-only form of IFRELA is not available.

Execute-only syntax

An execute-only form of IFRELA is not available.

Parameters

Enclose the parameter inside parentheses, as shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the only required parameter. The code is a binary integer value.

Notes and tips

Use the IFRELA call to explicitly release *all* found sets of records.

When executing an IFRELA call, Model 204 performs an IFRELR function on each record set and list that is held by the current thread. See the IFRELR call.

The IFRELA call is the equivalent of the RELEASE ALL RECORDS statement in SOUL in the host language multiple cursor environment. See the Rocket Model 204 documentation wiki for information about the RELEASE ALL RECORDS statement:

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

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5) COMP SYNC.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFRELA" USING RETCODE.
```

IFRELR call *-mc*

Function

The IFRELR call (RELEASE RECORDS) releases a particular record set held by the current thread.

Full syntax (103)

```
IFRELR(RETCODE,SET_QUAL)
```

Compile-only syntax

A compile-only form of IFRELR is not available.

Execute-only syntax

An execute-only form of IFRELR is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
SET_QUAL	[I,c,r] The set qualifier is required for specifying the record set or list whose records will be released. Specify the set qualifier as a character string using either one of the following formats: `[IN label | ON [LIST] listname]` where: *label* is the name of a saved IFFIND, IFFNDX, IFFWOL, IFFAC, IFSORT, IFFDV, or IFSRTV compilation from a previously compiled call. *listname* specifies the name of a list which contains the found set.

Notes and tips

Use the IFRELR call to explicitly release *a single* found set of records.

If you issue an IFRELR call on a sorted set, Model 204 frees the records in the scratch file. If you issue an IFRELR call on a list, Model 204 performs a CLEAR LIST function. See the IFCLST (CLEAR LIST) call.

The IFRELR call is the equivalent of the RELEASE RECORDS statement in SOUL in the host language multiple cursor environment. See the Rocket Model 204 documentation wiki for information about the RELEASE RECORDS statement:

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

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE   PIC 9(5) COMP SYNC.
    05  REL-SPEC  PIC X(10) VALUE "IN PREVFD;".
•
•
•
PROCEDURE DIVISION.


•
•
•
  CALL "IFRELR" USING RETCODE, REL-SPEC.
```

IFRFLD call *-mc,sc*

Function The IFRFLD call (REDEFINE FIELD) redefines one or more fields that have been previously defined for a Model 204 file. IFRFLD requires Model 204 file manager privileges.

Full syntax (59) `IFRFLD(RETCODE,FIELD_DESC,FILE_SPEC)`

Compile-only syntax A compile-only form of IFRFLD is not available.

Execute-only syntax An execute-only form of IFRFLD is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.

FIELD_DESC	[I,c,r] The field description is a required input parameter which specifies the name of the field to be redefined and a list of its new attributes. You may specify more than one field to be redefined. Specify the field description as a character string using the following format:
	`fieldname (attribute •••)`
	`[fieldname (attribute •••)];`
	where:
	fieldname is required and specifies the name of the Model 204 field that is being redefined. Specify the name as a character string, up to 255 characters in length.
	attribute is required and specifies a particular characteristic that controls how the field is used, stored, or accessed. You may specify more than one attribute, separating each by a comma or a blank. Specify only the attributes to be changed.
	The attributes that may be specified using the IFRFLD call are identical to those that are used with the Model 204 REDEFINE command. See the Rocket Model 204 documentation wiki for a description of field attributes used with the REDEFINE command:
	http://m204wiki.rocketsoftware.com/index.php/REDEFINE_command

Parameter	Description
FILE_SPEC	[I,s,o] The file specification is an optional input parameter for use only with a multiple cursor IFSTRT thread for specifying the name of the Model 204 file that contains the field to be redefined. Specify the name of the file as a short character string variable using the following format: `IN [FILE] filename` The specified file must be open on the thread, otherwise the call is unsuccessful and Model 204 returns a completion code of 4.

Notes and tips

Use the IFRFLD call to redefine a previously defined field in an open Model 204 file. Using the IFRFLD call allows a field definition to be altered without requiring that the file be reinitialized or reloaded. The IFRFLD call is valid only in file context, not for a group. The IFRFLD call is valid on *all* types of IFSTRT threads.

When FOPT=X'10' and the date/time stamp feature is installed, the IFRFLD function is supported for DTS files.

Note: The file context can change on a multiple cursor thread and, if the file specification parameter (FILE_SPEC) is omitted, IFRFLD redefines the field for the default file on the thread.

If the field being redefined has a security level, the Model 204 file manager must have field level security (FLS) access privileges. Note that changing a field's security level to 0 desecures the field. See the Rocket Model 204 documentation wiki for information about field security:

http://m204wiki.rocketsoftware.com/index.php/Security#Field-level_security

Note also that IFRFLD follows the same basic rules for specifying field attribute definitions as the Model 204 REDEFINE FIELD command. See the Rocket Model 204 documentation wiki for information about using the REDEFINE FIELD command:

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

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5)  COMP SYNC.
    05  MODIFY   PIC X(20) VALUE 'FIELDA (UP,LEVEL 0);'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFRFLD" USING RETCODE, MODIFY.
```

IFRNUM call *-mc*

Function

The IFRNUM call (EXTRACT RECORD NUMBER) returns the number of the current record in the specified cursor in an output parameter.

Full syntax (121)

```
IFRNUM(RETCODE,CURSOR_NAME,RECNUM)
```

Compile-only syntax

A compile-only form of IFRNUM is not available.

Execute-only syntax

An execute-only form of IFRNUM is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
CURSOR_NAME	[I,s,r] Is a required input parameter which specifies the name of the cursor whose record number is to be returned. This is a short character string, the name previously assigned to the cursor in a corresponding IFOCUR call. See CURSOR_NAME on page 226 for a description of the cursor name for the IFOCUR call.
RECNUM	[O,c,r] Is a required output parameter which returns the number of the current record for the specified cursor. Specify a character string variable. Model 204 returns a number in display format, up to eleven characters in length.

Notes and tips

Use the IFRNUM call to access the internal number of the record, that is, the database record number, at the location where the specified cursor is currently positioned.

The IFRNUM call is the equivalent of the $CURREC function in SOUL in the multiple cursor environment.

See the Rocket Model 204 documentation wiki for information about the $CURREC function:

http://m204wiki.rocketsoftware.com/index.php/$Currec

IFRNUM returns a -1 for the current record number (RECNUM) to indicate that there is no current record for the specified cursor. This result occurs if no fetch is issued or if an IFRELA call was issued to release the record set.

Note: To successfully issue the IFRNUM call, you must specify a cursor that meets *both* of the following requirements:

- The cursor was declared in a previously compiled IFOCUR call on the thread. See the IFOCUR (open cursor) call.
- The cursor is open.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5) COMP SYNC.
    05  RECNUM   PIC X(11).
    05  CURNAME  PIC X(4) VALUE "CR1;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFRNUM" USING RETCODE, CURNAME, RECNUM.
```

IFRPRM call *-mc,sc*

Function The IFRPRM call (RESET PARAMETER) resets the value of one or more specified Model 204 parameters.

Full syntax (27) `IFRPRM(RETCODE,PARM_LIST,FILE_SPEC)`

Compile-only syntax A compile-only form of IFRPRM is not available.

Execute-only syntax An execute-only form of IFRPRM is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
PARM_LIST	[I,c,r] The parameter list is a required input parameter, which specifies the name and value pair for each Model 204 parameter whose value is to be reset. Specify a character string using the following format: `parm1=value1 [,parm2=value2 •••];` where: *parm1* is the name of the Model 204 parameter to be reset, and parm2 is the name of a second parameter to be reset. Additional parameters may be specified in a name-value pair. Specify the keyword name of the Model 204 system, file, or user parameter. *value1* is the new value for the specified parameter in the first pair, and value2 is the new value for the specified parameter in the second pair. A value is required for each name that is specified in the list. Values may be specified in decimal form, such as 193, in hexadecimal form, such as X'C1', or in character form, such as C'A'. For example, the specification OPENCTL=128 is equivalent to OPENCTL=X'80'. You may specify more than one name=value pair, separating each by a comma or a blank.

Parameter	Description
FILE_SPEC	[I,s,o] The file specification is an optional input parameter for use only with a multiple cursor IFSTRT thread for specifying the name of the file for which the Model 204 file parameter will be reset. Specify the Model 204 file name as a short character string using the following format: `IN [FILE] filename;` The specified file must be open on the thread, otherwise the call is unsuccessful and Model 204 returns a completion code equal to 4.

Notes and tips

Use the IFRPRM call to reset certain Model 204 system, file, or user parameters. The IFRPRM call is valid for resetting individual file parameters only in file context, not for a group. IFRPRM follows the same basic rules for specifying parameter settings as the Model 204 RESET PARAMETER command. See the Rocket Model 204 documentation wiki for information about using the RESET command to set file parameters:

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

The IFRPRM call is equivalent to the IFSPRM call.

The IFRPRM call is valid on *all* types of IFSTRT threads. Note that the file context can change on a multiple cursor thread. If a Model 204 file parameter is specified for PARM_LIST and the file specification parameter (FILE_SPEC) is omitted, IFRPRM resets the value for the default file on the thread.

Note: Use IFRPRM with caution to avoid resetting sensitive parameters that may affect the entire operating environment. See the Rocket Model 204 documentation wiki for information about Model 204 parameters:

http://m204wiki.rocketsoftware.com/index.php/List_of_Model_204_parameters

Resetting TBO or non-TBO files

If a file is the only file open for all threads on a job, you may reset the file parameters to change the file from transaction backout (TBO) to non-TBO or vice versa. Note, however, that changing a file from TBO to non-TBO (or non-TBO to TBO) will cause a file discontinuity for backout and recovery. Therefore, if you use the IFRPRM call to change TBO status, Rocket suggests that you reset the TBO parameters as soon as possible as described in the steps below. In this example, the job starts with TBO files in use.

1. Before you start, commit all current updates and close all open files
2. Open just the file you want to change, reset that file from TBO to non-TBO, and close the file.
3. Open the non-TBO files that you need, and perform whatever work needs to be done with the file set as non-TBO. When you are done, commit the

updates and close all files.

4. As soon as possible, reopen the file that you originally changed, set it back to TBO, and close the file.
5. At this point, you can reopen files and continue working.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5)  COMP SYNC.
    05  RESET    PIC X(12) VALUE 'OPENCTL=128;'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFRPRM" USING RETCODE, RESET.
```

IFRRFL call *-mc,sc*

Function

The IFRRFL call (REMOVE RECORD FROM LIST) removes the current record from the specified list.

Full syntax (34)

```
IFRRFL(RETCODE,LIST_NAME,CURSOR_NAME)
```

Compile-only syntax

A compile-only form of IFRRFL is not available.

Execute-only syntax

An execute-only form of IFRRFL is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LIST_NAME	[I,c,r] The list name is a required input parameter which specifies the name of the list to be used. Specify a character string variable which is the name of an existing list.
CURSOR_NAME	[I,s,r] The name of the cursor is an input parameter that is available only for use with a multiple cursor IFSTRT thread and is required for specifying the current record to be removed. Specify the cursor name as a short character string, using the name previously assigned to the cursor in a corresponding IFOCUR call. See CURSOR_NAME on page 226 for a description of the cursor name for the IFOCUR call. **Note:** The cursor name is not a valid parameter for use with a single cursor IFSTRT thread.

Notes and tips

Use the IFRRFL call to remove a record from a list.

The IFRRFL call is valid on *all* types of IFSTRT threads. On a multiple cursor IFSTRT thread, you must specify the cursor name for the current record. On a single cursor IFSTRT thread, IFRRFL processes the current record using the set that is current for the file or group most recently opened.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05  RETCODE    PIC 9(5) COMP SYNC.
    05 LISTNAME    PIC X(5) VALUE "SAVE;".
•
•
•
PROCEDURE DIVISION.
```

```
.
.
  CALL "IFRRFL" USING RETCODE, LISTNAME.
```

In this example, processing is done on a single cursor IFSTRT thread, and the IFRRFL call is preceded by IFFIND, IFLIST, and IFGET calls (not shown).

IFRRFLS call *-mc*

Function

The IFRRFLS call (REMOVE RECORDS FROM LIST) removes records from a particular list on the current thread.

Full syntax (110)

```
IFRRFLS|IFRFLS(RETCODE,LIST_SPEC)
```

Compile-only syntax

A compile-only form of IFRRFLS is not available.

Execute-only syntax

An execute-only form of IFRRFLS is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LIST_SPEC	[I,c,r] The list specification is a required input parameter which specifies an existing set of found records to be matched against and removed from a particular list. Specify the record set and the list as a character string using the following format: `set qualifier FROM [LIST] listname` where: *set qualifier* is required and specifies the name of a previously established found set or list using the IN label format, where label is the name of a saved IFFIND, IFFNDX, IFFWOL, IFFAC, IFSORT, IFFDV, or IFSRTV compilation from a previously compiled call. *listname* is required and specifies the name of a particular list from which records (having a match in the found set) will be removed.

Notes and tips

Use the IFRRFLS call to delete from a particular list any records that match those in the specified found set or list.

The IFRRFLS call is the equivalent of the REMOVE RECORDS FROM LIST statement in SOUL in the host language multiple cursor environment. See the Rocket Model 204 documentation wiki for information about the REMOVE RECORDS FROM LIST statement:

http://m204wiki.rocketsoftware.com/index.php/DML_statements_in_Parallel_Query_Option/204

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE    PIC 9(5) COMP SYNC.
    05  LISTSPEC   PIC X(22) VALUE "IN PREVFD FROM LIST L;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFRRFLS" USING RETCODE, LISTSPEC.
```

IFSETUP call *-di*

Function The IFSETUP call (SETUP) initiates contact with Model 204 and sets the PARM parameters and the CCAIN control statements for the IFDIAL IFAM1 Host Language Interface job.

Full syntax (86)

```
IFSETUP|IFSETP(RETCODE,LANG_IND,EXEC,PROLOGUE)
```

Compile-only syntax A compile-only form of IFSETUP is not available.

Execute-only syntax An execute-only form of IFSETUP is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value.
LANG_IND	[I,i,r] The language indicator is a required input parameter, which establishes the calling sequence convention to be used corresponding to the host language. The indicator specifies the format of parameters that are passed in subsequent calls. Specify one of the following integer values: 1 = PL/1 F-level, and BAL languages 2 = COBOL, FORTRAN, and BAL languages 3 = PL/1 with +Optimizer/Checkout compilers, VS/FORTRAN, and BAL languages **Note:** Any convention may be specified for use with the BAL language, and the BAL programmer must adhere to the convention that is specified when coding parameters.
EXEC	[I,c,o] The EXEC specification is an optional input parameter, which specifies the value of the PARM parameter of the EXEC JCL statement for the IFAM1 job. Specify a character string, up to 100 bytes in length, and separate parameters with a comma (,). If the language indicator (LANG_IND) setting is 2, append a semicolon (;) to the end of the input string. A null string indicates that no PARM values are to be set. **Note:** If the LIBUFF and LOBUFF parameters are to be set, include them in the parameter list. Do not specify the following parameters in the PARM list: ALTIODEV, NUSERS, and NSERVS.

Parameter	Description
PROLOGUE	[I,c,o] The prologue specification is an optional input parameter which is the first logical line of the CCAIN input stream that is used to specify User 0 parameters. Specify a character string, and separate parameters with a comma (,). If the language indicator (LANG_IND) setting is 2, append a semicolon (;) to the end of the input string. A null string indicates that no Model 204 parameters are to be set.
	Note: Do not specify the following parameters in the User 0 line: ALTIODEV, NUSERS, and NSERVS.

Notes and tips

Use the IFSETUP call to set the job control statements for a Host Language Interface IFAM1 IFDIAL job. Issue the IFSETUP call before IFDIAL in the IFAM1 job.

Completion return code (RETCODE)

If the IFSETUP call is unsuccessful, Model 204 returns an error code of 400 for an invalid language code (LANG_IND).

Coding example (PL/1 Optimizer)

```
DCL NERRFIXED BIN(31)INIT(0);
DCL LANG_CODEFIXED BIN(31)INIT(3); /* LANG=3 for PL/1 Opt */
DCL EXEC_PARMCHAR(100)INIT('SYSOPT=144,LIBUFF=500;');
DCL CCAIN_PARMCHAR(256) INIT('SPCORE=5000;');
DCL IFSETUPENTRY(FIXED BIN(31), FIXED BIN(31),
CHAR(*), CHAR(*));
•
•
•
CALL IFSETUP (NERR, LANG_CODE, EXEC_PARM, CCAIN_PARM);
•
•
•
```

IFSKEY call *-mc,sc*

Function The IFSKEY call (SORT KEYS) sorts the records in a found set or list in the specified order using the record keys.

Full syntax (83)
```
IFSKEY(RETCODE,SORT_SPEC,SKEY_NAME,%VARBUF,%VARSPEC)
```

Compile-only syntax (84)
```
IFSKYC(RETCODE,SORT_SPEC,SKEY_NAME)
```

Execute-only syntax (85)
```
IFSKYE(RETCODE,SKEY_NAME,%VARBUF,%VARSPEC)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
SORT_SPEC	[I,c,r] The sort specification is required to specify ordering criteria for a found record set. Specify the ordering clause as a character string. See SORT_SPEC on page 278 for a detailed description of the sort specification used with IFSORT that is also valid for IFSKEY. **Note:** The SORT_SPEC options that are available in an IFSKEY call are similar to the sort options available in a SOUL SORT RECORD KEYS statement. For IFSKEY, the EACH term is not supported. See the Rocket Model 204 documentation wiki for information about sorting: http://m204wiki.rocketsoftware.com/index.php/Sorting
SKEY_NAME	[I,s,r/o] The name of the IFSKEY compilation is an input parameter that is required for use with a multiple cursor IFSTRT thread, and is only required for a single cursor IFSTRT thread if using the Compiled IFAM facility (IFSKYC and IFSKYE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). On a single cursor IFSTRT thread, any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. On a multiple cursor IFSTRT thread, the first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). **Note:** A null value is equivalent to omitting the name parameter, and is not valid for a multiple cursor thread.

Parameter	Description
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area that accommodates up to 255 bytes of data per value.
	The buffer contains values that are defined by the %VARSPEC parameter to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter and lists the names of %variables to be assigned. %VARSPEC specifies the contents of the variable buffer. Specify a character string that follows LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFSKEY call to sort a found set; the record number of the Model 204 record is added to the temporary sort record. On a single cursor IFSTRT thread, the sorted set replaces the IFFIND or IFFAC set and becomes the current set.

The sorted records contain only the keys and the Table B record numbers, so the records are very small. The sorted set acts as an index into Table B. Any reference to the sorted record directly accesses the Table B record.

IFSKEY conserves the amount of CCATEMP space that is used and provides relatively fast sort processing. In comparison to IFSORT, IFSKEY processing is faster and more efficient. See the IFSORT call.

The IFSKEY call is valid on *all* types of IFSTRT threads. You must specify the found set of records that are to be sorted on a multiple cursor IFSTRT thread. On a single cursor IFSTRT thread, IFSKEY sorts records using the current IFFIND or IFFAC set.

Record enqueuing

IFSKEY does not lock the original Table B records. Another Model 204 user can update them while the sorted records are being processed.

Completion return code (RETCODE)

If the IFSKEY call is unsuccessful, Model 204 returns an error code of 4 for either of the following error conditions:

- General syntax error
- Attempt to sort an already sorted set

Processing sorted records

Once records are sorted, you can issue subsequent retrieval calls, such as IFGET on a single cursor IFSTRT thread or IFFTCH on a multiple cursor thread, to retrieve fields from the sorted records. On a single cursor IFSTRT thread, you can also use the IFMORE call or the IFCTO call after IFSKEY, and you can use IFOCC on a multiple cursor IFSTRT thread.

On either a single cursor or multiple cursor IFSTRT thread, you can use IFPROL or IFRRFL to update a list with records from a sorted set. Because lists are named sets of record numbers that exist as bit patterns, lists that contain records from sorted sets are not maintained in sorted order. However, you can explicitly sort any list.

Restrictions on processing sorted records

You cannot use IFSKEY with a record set that is already sorted. On a single cursor IFSTRT thread, a sorted set cannot be resorted without an intervening call to IFFIND or IFFAC to rebuild the original set.

On a single cursor IFSTRT thread, any call to IFFIND replaces the sorted set and discards any unprocessed sorted records. This means that an HLI program that uses single cursor IFSTRT threads and finds new records within a loop on sorted records must use one thread for the sorted records and another thread for the inner IFFINDs.

You cannot use updating calls, such as IFPUT on a single cursor IFSTRT thread or IFUPDT on a multiple cursor thread, with sorted records. You cannot use IFLIST, IFDSET, and IFFILE to process a sorted set.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE      PIC 9(5) COMP SYNC.
    05 SORT-SPEC    PIC X(53) VALUE "CUSTOMER NAME AND
      PRODUCT CODE VALUE DESCENDING;".
    05 SKYNAME      PIC X(8) VALUE 'SAVSORT;'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFSKEY" USING RETCODE, SORT-SPEC, SKYNAME.
```

This example operates similarly to the example shown for IFSORT on page 282.

IFSORT call *-mc,sc*

Function The IFSORT call (SORT) sorts the records in a found set in the specified order and creates a record set.

Full syntax (68)
```
IFSORT(RETCODE,SORT_SPEC,SORT_NAME,%VARBUF,%VARSPEC)
```

Compile-only syntax (69)
```
IFSRTC(RETCODE,SORT_SPEC,SORT_NAME)
```

Execute-only syntax (70)
```
IFSRTE(RETCODE,SORT_NAME,%VARBUF,%VARSPEC)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.

Parameter	Description
SORT_SPEC	[I,c,r] The sort specification is required to specify ordering criteria for a found record set. Specify the ordering clause using the following format line: `[set qualifier] BY key1 [AND key2 ••• AND keyn] [VALUE [ASCENDING	DESCENDING]]` where: *set qualifier* is available only for use with a multiple cursor IFSTRT thread and it is required for specifying the record set or list whose records will be sorted. Note that the set qualifier is not a valid parameter for use with a single cursor IFSTRT thread. Specify the set qualifier as a character string using one of the following formats: `[IN label	ON [LIST} listname]` where • *label* is the name of a saved IFFIND, IFFNDX, IFFWOL, or IFFAC compilation from a previously compiled call. • *listname* specifies the name of a list. *BY* clause specifies the key, or keys, for ordering records. Each key is the name of a field to be used for ordering the records, key1 is the highest order for the sort, key2 is next highest, and so on, keyn is the lowest sort level. Separate keys with the keyword AND. *ASCENDING* and *DESCENDING* are mutually exclusive keywords that indicate the order in which the record set will be processed. ASCENDING order is the default. **Note:** The sort specification is required for a record set. To specify default sorted order (ASCENDING) on a multiple cursor IFSTRT thread, specify the IN label or ON listname clause followed by a semicolon (;). On a single cursor IFSTRT thread, specify a semicolon for default ordering.
	The SORT_SPEC options that are available in an IFSORT call are similar to the sort options available in a SORT statement in SOUL. See the Rocket Model 204 documentation wiki for information about sorting: http://m204wiki.rocketsoftware.com/index.php/Sorting

Parameter	Description
SORT_SPEC	[I,c,r] The sort specification is required to specify ordering criteria for a found record set. Specify the ordering clause using the following format line: `[set qualifier] BY key1 [AND key2 ••• AND keyn] [VALUE [ASCENDING	DESCENDING]]` where: *set qualifier* is available only for use with a multiple cursor IFSTRT thread and it is required for specifying the record set or list whose records will be sorted. Note that the set qualifier is not a valid parameter for use with a single cursor IFSTRT thread. Specify the set qualifier as a character string using one of the following formats: `[IN label	ON [LIST} listname]` where • *label* is the name of a saved IFFIND, IFFNDX, IFFWOL, or IFFAC compilation from a previously compiled call. • *listname* specifies the name of a list. *BY* clause specifies the key, or keys, for ordering records. Each key is the name of a field to be used for ordering the records, key1 is the highest order for the sort, key2 is next highest, and so on, keyn is the lowest sort level. Separate keys with the keyword AND. *ASCENDING* and *DESCENDING* are mutually exclusive keywords that indicate the order in which the record set will be processed. ASCENDING order is the default. **Note:** The sort specification is required for a record set. To specify default sorted order (ASCENDING) on a multiple cursor IFSTRT thread, specify the IN label or ON listname clause followed by a semicolon (;). On a single cursor IFSTRT thread, specify a semicolon for default ordering.
	The SORT_SPEC options that are available in an IFSORT call are similar to the sort options available in a SORT statement in SOUL. See the Rocket Model 204 documentation wiki for information about sorting: http://m204wiki.rocketsoftware.com/index.php/Sorting

Parameter	Description
SORT_NAME	[I,s,r/o] The name of the IFSORT compilation is an input parameter that is required for use with a multiple cursor IFSTRT thread, and is only required for a single cursor IFSTRT thread if using the Compiled IFAM facility (IFSRTC and IFSRTE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). On a single cursor IFSTRT thread, any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. On a multiple cursor IFSTRT thread, the first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z), which can be followed by a letter, a digit (0–9), a period (.), or an underscore (_). **Note:** A null value is equivalent to omitting the name parameter, and is not valid for a multiple cursor thread.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area, which accommodates up to 255 bytes of data per value. The buffer contains values that are defined by the %VARSPEC parameter to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter and lists the names of %variables to be assigned. %VARSPEC specifies the contents of the variable buffer. Specify a character string that follows LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFSORT call to sort a found set of records. The sorted records are temporary copies of the original records. The record number of the Model 204 record is added to the temporary sort record.

On a single cursor IFSTRT thread, the sorted set replaces the IFFIND or IFFAC set and becomes the current set. On a multiple cursor IFSTRT thread, the IFFIND or IFFAC set is not replaced and may be accessed again.

IFSORT makes a copy of the original Table B record in CCATEMP and sorts the copies. All references to fields on the sorted records are references to CCATEMP.

Because of the space limitations of CCATEMP and the performance characteristics of Model 204 sort processing, Rocket does not recommend

using IFSORT to sort very large numbers of records. The IFSKEY call provides an efficient alternative to IFSORT processing. See the IFSKEY call.

The IFSORT call is valid on *all* types of IFSTRT threads. You must specify the found set of records that are to be sorted on a multiple cursor IFSTRT thread. On a single cursor IFSTRT thread, IFSORT sorts records using the current IFFIND or IFFAC set.

Completion return code (RETCODE)

If the IFSORT call is unsuccessful, Model 204 returns an error code of 4 for either of the following error conditions:

- General syntax error
- Attempt to sort an already sorted set

Record enqueuing

On a single cursor IFSTRT thread, enqueuing on the found set is released when the sort is performed. IFSORT does not lock the original Table B records. Another Model 204 user can update the original records while the sorted records are being processed, but the sorted records are not updated.

Processing sorted records

Once records are sorted, you can issue subsequent retrieval calls, such as IFGET on a single cursor IFSTRT thread or IFFTCH on a multiple cursor IFSTRT thread, to retrieve fields from the sorted records. On a single cursor IFSTRT thread, you can also use the IFMORE call or the IFCTO call after IFSORT, and you can use IFOCC on a multiple cursor IFSTRT thread.

On either a single cursor or a multiple cursor thread, you can use IFPROL or IFRRFL to update a list with records from a sorted set. Since lists are named sets of record numbers that exist as bit patterns, lists that contain records from sorted sets are not maintained in sorted order. However, you may explicitly sort any list.

Restrictions on processing sorted records

Note that on a single cursor IFSTRT thread, you cannot use IFSORT with a record set that is already sorted. On a single cursor IFSTRT thread, a sorted set cannot be resorted without an intervening call to IFFIND or IFFAC to rebuild the original set.

On a single cursor IFSTRT thread, any call to IFFIND replaces the sorted set and discards any unprocessed sorted records.

Therefore, a host language program that uses single cursor IFSTRT threads and finds new records within a loop on sorted records must use one thread for the sorted records and another thread for the inner IFFINDs.

You cannot use updating calls, such as IFPUT on a single cursor IFSTRT thread or IFUPDT on a multiple cursor IFSTRT thread, with sorted records. You cannot use the IFLIST, IFDSET, and IFFILE to process a sorted set.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05 RETCODE      PIC 9(5) COMP SYNC.
    05 SORT-SPEC    PIC X(53) VALUE "CUSTOMER NAME AND
    PRODUCT CODE VALUE DESCENDING;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFSORT" USING RETCODE, SORT-SPEC.
```

This example takes the found record set and generates a sorted set of temporary records. Each element of the sorted set contains a customer name and a product code, and the records are ordered alphabetically by customer name. When there are multiple product codes for a given customer name, the record with the higher product code precedes the one with the lower code.

IFSPRM call *-mc,sc*

Function The IFSPRM call (SET PARAMETER) sets the value of one or more specified Model 204 parameters.

Full syntax (26) `IFSPRM(RETCODE,PARM_LIST,FILE_SPEC)`

Compile-only syntax A compile-only form of IFSPRM is not available.

Execute-only syntax An execute-only form of IFSPRM is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
PARM_LIST	[I,c,r] The parameter list is a required input parameter that specifies the name and value pair for each Model 204 parameter whose value is to be set. Specify a character string using the following format: `parm1=value1 [,parm2=value2•••];` where: *parm1* is the name of the Model 204 parameter to be set, and parm2 is the name of a second parameter to be set. Additional parameters may be specified in a name-value pair. Specify the keyword name of the Model 204 system, file or user parameter. *value1* is the new value for the specified parameter in the first pair, and value2 is the new value for the specified parameter in the second pair. A value is required for each name that is specified in the list. Values may be specified in decimal form, such as 193, in hexadecimal form, such as X'C1', or in character form, such as C'A'. For example, the specification OPENCTL=128 is equivalent to OPENCTL=X'80'. You may specify more than one name=value pair, separating each by a comma or a blank.

Parameter	Description
FILE_SPEC	[I,s,o] The file specification is an optional input parameter for use only with a multiple cursor IFSTRT thread for specifying the name of the file for which the Model 204 file parameter is set. Specify the Model 204 file name as a short character string using the following format: `IN [FILE] filename;` The specified file must be open on the thread, otherwise the call is unsuccessful and Model 204 returns a completion code equal to 4.

Notes and tips

Use the IFSPRM call to set or, in effect, to reset certain Model 204 system, file or user parameters. The IFSPRM call is valid for resetting individual file parameters only in file context, not for a group. The IFSPRM call is equivalent to the IFRPRM call and IFSPRM follows the same basic rules for specifying parameter settings as the Model 204 RESET PARAMETER command.

See the Rocket Model 204 documentation wiki for information about using the RESET command to set file parameters:

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

The IFSPRM call is valid on *all* types of IFSTRT threads. Note that the file context can change on a multiple cursor thread. If a Model 204 file parameter is specified for PARM_LIST and the file specification parameter (FILE_SPEC) is omitted, IFSPRM sets the value for the default file on the thread.

Note: Use IFSPRM with caution to avoid resetting sensitive parameters that may affect the entire operating environment. See the Rocket Model 204 documentation wiki for information about Model 204 parameters:

http://m204wiki.rocketsoftware.com/index.php/List_of_Model_204_parameters

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5)  COMP SYNC.
    05  SET      PIC X(12) VALUE 'OPENCTL=128;'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFSPRM" USING RETCODE, SET.
```

IFSRTV call *-mc,sc*

Function The IFSRTV call (SORT VALUES) sorts the values in a found set in the specified order and creates a sorted value set.

Full syntax (80)
```
IFSRTV(RETCODE,SORT_SPEC,SRTV_NAME)
```

Compile-only syntax (81)
```
IFSTVC(RETCODE,SORT_SPEC,SRTV_NAME)
```

Execute-only syntax (82)
```
IFSTVE(RETCODE,SRTV_NAME)
```

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
SORT_SPEC	[I,c,r] The sort specification is required to specify ordering criteria and is valid for an unsorted or a sorted value set. To indicate default ordering for the value set, specify a semicolon (;). Specify the ordering clause using the following format line with an IN ORDER clause: `[set qualifier] [IN [ASCENDING | DESCENDING] [NUMERICAL | CHARACTER | RIGHT-ADJUSTED] ORDER]` where: *set qualifier* is available only for use with a multiple cursor IFSTRT thread and it is required for specifying the value set whose values will be sorted. **Note:** The set qualifier is not a valid parameter for use with a single cursor IFSTRT thread. Specify the set qualifier as a character string using the IN label clause, where label is the name of a saved IFFDV or IFSRTV compilation from a previously compiled call that established the value set. *ASCENDING* and *DESCENDING* are mutually exclusive keywords that indicate the order in which the value set is processed. ASCENDING order is the default. CHARACTER, RIGHT-ADJUSTED and NUMERIC are mutually exclusive keywords. • *CHARACTER* specifies values sorted in standard EBCDIC collating sequence. • *RIGHT-ADJUSTED* specifies that values are temporarily right-justified before s • *NUMERICAL* specifies a sort of number values with the usual numeric order relationships.

Parameter	Description
	Note: The sort specification is required for a value set. To specify default sorted order (ASCENDING) on a multiple cursor IFSTRT thread, specify the IN label clause followed by a semicolon (;). On a single cursor IFSTRT thread, specify a semicolon for default ordering. See the Rocket Model 204 documentation wiki for more information about sorting: http://m204wiki.rocketsoftware.com/index.php/Sorting
SRTV_NAME	[I,s,r/o] The name of the IFSRTV compilation is an input parameter that is required for use with a multiple cursor IFSTRT thread, and is only required for a single cursor IFSTRT thread if using the Compiled IFAM facility (IFSTVC and IFSTVE). Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). On a single cursor IFSTRT thread, any characters *except* the following are valid in the name: blank, comma, parenthesis, equal sign, or semicolon. On a multiple cursor IFSTRT thread, the first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). **Note:** A null value is equivalent to omitting the name parameter, and is not valid for a multiple cursor thread.

Notes and tips

Use the IFSRTV call to sort a value set. On a single cursor IFSTRT thread, the sorted set replaces the IFFDV or IFSRTV set and becomes the current set. On a multiple cursor IFSTRT thread, the IFFDV or IFSRTV set is not replaced and may be accessed again.

You can issue more than one IFSRTV call and you can sort a value set that is already sorted. Note that if you issue IFSRTV after some values have been extracted using IFGETV on a single cursor IFSTRT thread, Model 204 sorts the values that are remaining in the set.

The IFSRTV call is valid on *all* types of IFSTRT threads. You must specify the found value set that is to be sorted on a multiple cursor IFSTRT thread. On a single cursor IFSTRT thread, IFSRTV sorts values using the current IFFDV or IFSRTV set.

If the value set does not exist, Model 204 does not perform the sort and returns an error completion code.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  ARGS-FOR-CALL.
    05  RETCODE     PIC 9(5) COMP SYNC.
    05  SORT-SPEC   PIC X(26) VALUE
        "IN FDV IN ASCENDING ORDER;".
```

```
    05 NAME         PIC X(8) VALUE "SAVEVAL;".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFSORT" USING RETCODE, SORT-SPEC.
```

IFSTHRD call *-mc,sc*

Function

The IFSTHRD call (SWITCH THREAD) deactivates the current thread and activates the specified thread.

Full syntax (2)

```
IFSTHRD|IFSTRD(RETCODE,NEW_ID,OLD_ID)
```

Compile-only syntax

A compile-only form of IFSTHRD is not available.

Execute-only syntax

An execute-only form of IFSTHRD is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value.
NEW_ID	[I,i,r] The new thread identifier is a required input parameter which identifies the thread to be made current. This is the thread identifier previously assigned by the IFSTRT or IFSTRTN call which started the thread. (See the THRD_ID parameter for IFSTRT on page 300.) Specify an integer value.
OLD_ID	[O,i,r] The old thread identifier is a required output parameter which identifies the thread that is being deactivated. Model 204 returns the integer value which identifies the current thread.

Notes and tips

Use the IFSTHRD call to switch from the current thread to another, while holding the connection with the old thread. IFSTHRD involves low overhead. Note that you cannot use IFSTHRD to deactivate the current thread without specifying a new thread.

You can use the IFSTHRD call on any type of IFSTRT thread except for IFAM1. IFSTHRD is not valid for use with an IFAM1 thread.

The IFSTHRD call is useful for switching threads in a multithreaded IFAM2 or IFAM4 application using single cursor IFSTRT threads, for parallel processing of several sets of records, or for cross-referencing between records in different files.

See Appendix B for an example of a multithreaded application. See the *Rocket Model 204 Host Language Interface Programming Guide* for information about multithreaded IFSTRT transactions.

Completion return code (RETCODE)

If the IFSTHRD call is unsuccessful, Model 204 returns an error code of 95 if a nonexistent new thread is specified and ignores the call.

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  START-ARGS.
    05  COBOL-IND         PIC 9(8) VALUE 2.
    05  LOGIN-INFO        PIC X(12) VALUE 'USERX;PASSW;'.
    05  ACCESS-MODE       PIC 9(8) VALUE 0.
01  THREAD-NBR.
    05  FILEA-THREAD-NBR  PIC 9(8).
    05  FILEB-THREAD-NBR  PIC 9(8).
    05  THREAD-NBR        PIC 9(8).
01  WORK-ARGS.
    05  RETCODE           PIC 9(5) COMP SYNC.
    05  FILE-NAME         PIC X(6).
•
•
•
PROCEDURE DIVISION.
BEGIN-RTN.
  MOVE "APPLES" TO FILE-NAME.
  PERFORM START-THREAD.
  IF (RETCODE = ZERO) OR (RETCODE =16) THEN
    MOVE THREAD-NBR TO FILEA-THREAD-NBR.
  MOVE "BANANA" TO FILE-NAME
  PERFORM START-THREAD.
  IF (RETCODE = ZERO) OR (RETCODE =16) THEN
        MOVE THREAD-NBR TO FILEB-THREAD-NBR.
START-THREAD.
  CALL "IFSTRT" USING RETCODE, COBOL-IND, LOGIN-INFO,
             ACCESS-MODE, THREAD-NBR.
•
•
•
GET-FILEA-REC.
  CALL "IFSTHRD" USING RETCODE, FILEA-THREAD-NBR, THREAD-NBR.
•
•
•
GET-FILEB-REC.
  CALL "IFSTHRD" USING RETCODE, FILEB-THREAD-NBR, THREAD-NBR.
•
•
•
```

IFSTOR call *-mc*

Function

The IFSTOR call (STORE RECORD) creates a new record and adds it to the specified file. IFSTOR specifies the data fields that comprise the new record. Note that, for a saved compilation, IFSTOR allocates a cursor that points to the stored record.

Full syntax (112)

```
IFSTOR(RETCODE,FILE_SPEC,BUFFER,EDIT_SPEC,STOR_NAME,
%VARBUF,%VARSPEC,RECNUM)
```

Compile-only syntax (113)

```
IFSTRC(RETCODE,FILE_SPEC,EDIT_SPEC,STOR_NAME)
```

Execute-only syntax (114)

```
IFSTRE(RETCODE,BUFFER,STOR_NAME,%VARBUF,%VARSPEC,RECNUM)
```

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
FILE_SPEC	[I,c,r] The file specification is a required input parameter which identifies the Model 204 file that will be updated to contain the new record. Specify the file as a character string using a standard Model 204 IN clause. See the Rocket Model 204 documentation wiki for information about the IN clause: http://m204wiki.rocketsoftware.com/index.php/Files,_groups,_and_reference_context#IN_clause If a group is specified, the member clause may be used to specify a particular file within the group. Note that if a group name is specified in the IN clause and the member name is not coded, the group update file is specified by default.
BUFFER	[I,c,r] The buffer location is a required input parameter that specifies the address of the user's data area. Specify a character string variable. The buffer supplies the data, the actual values, for the fields that are defined by the EDIT_SPEC parameter, described on page 291.

Parameter	Description

EDIT_SPEC	[I,c,r] The edit specification is a required input parameter which defines the fields that are to be added to form the new record. The EDIT_SPEC describes the format of the data which is read at the buffer location, described on page 290.
	Note: If the file is a sorted or hash key file and the key is required, the first field name in the EDIT_SPEC must be the key field.
	Specify a character string using one of the following LIST, DATA, or EDIT format options:
	`LIST (fieldname list);`
	`DATA;`
	`EDIT (fieldname list) (edit formats);`

Parameter	Description
	where: *fieldname list* is required for the LIST or EDIT specification and specifies a field name or names. Specify elements in the field name list using one of the following options: • fieldname • fieldname(n) • fieldname(*) • fieldname(%variable) • fieldname(+n) • fieldname(+%variable) where: • *fieldname* updates the first occurrence of the named field. Note that this is equivalent to fieldname(1). If the field does not occur in the current record, IFSTOR adds it. • *fieldname(n)* updates the nth occurrence of the named field for a multiply occurring field. If the nth occurrence does not exist in the current record, IFSTOR adds it. • *fieldname(*)* adds the named field to the current record. If the field already exists in the current record, IFSTOR adds another occurrence of the field. • *fieldname(%variable)* retrieves the occurrence of the field specified by the %VARBUF and %VARSPEC parameters. If the nth occurrence does not exist in the current record, IFSTOR adds it. • *fieldname(+n)* inserts a new occurrence of the named field to the current record. This is analogous to the INSERT statement in SOUL and is useful for adding new occurrences of a field where the order of the values is important. Insert the new occurrence as the nth occurrence. • *fieldname (+%variable)* inserts a new occurrence of the field into the current record. The occurrence number is retrieved from the %VARBUF and %VARSPEC parameters. **Note:** If there is a current nth (or %variable) occurrence, make it the one after the nth occurrence. If n is greater than the current number of occurrences, add the new occurrence at the end. If the field does not occur in the current record, add it. If n is 0 or is not specified, treat it as though n=1 and insert the field as the first occurrence. *edit formats* is required in the EDIT specification and specifies a code or codes which indicate(s) the format of the data to be returned for the named field in the field name list-edit format pair. See “Using EDIT format codes for an updating call” on page 328 for a detailed description of the EDIT format codes that are used with IFSTOR. **Note:** See Chapter 7 for a description of LIST, DATA, and EDIT formatting.

Parameter	Description
STOR_NAME	[I,s,o] The name of the IFSTOR compilation is an optional input parameter. If specified, Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). The first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). A null value is equivalent to omitting the name parameter, and is not valid. **Note:** Model 204 allocates a cursor as part of the saved IFSTOR compilation. The cursor points to the stored record. You can reference this cursor using the STOR_NAME in an IFUPDT call to add additional fields to the record.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area, which accommodates up to 255 bytes of data per value. The buffer contains values, which are defined by the %VARSPEC parameter to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer. Specify a character string that follows LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.
RECNUM	[O,i,o] Record number is an optional output parameter that returns the Model 204 internal record number. The number is displayed as an integer.

Notes and tips

Use the IFSTOR call to create and store a record. IFSTOR optionally opens a cursor to the new record which allows it to be operated on by subsequent single record functions such as the IFUPDT call.

When FOPT=X'10' and the date/time stamp feature is installed, the IFSTOR function is *not* supported for DTS files.

The IFSTOR call is the equivalent of the STORE RECORD statement in SOUL and replaces the single cursor IFBREC and IFPUT call sequence in the multiple cursor environment.

See the IFBREC call, and the IFPUT call.

See the Rocket Model 204 documentation wiki for information about the STORE RECORD statement:

http://m204wiki.rocketsoftware.com/index.php/Data_maintenance#STORE_RECORD_statement

Completion return code (RETCODE)

If the IFSTOR call is unsuccessful, Model 204 returns an error code for either of the following error conditions:

Code	Error condition
10	Model 204 encountered invalid data values for BINARY and FLOAT numeric field for a file having FILEMODL set to NUMERIC VALIDATION.
200	A uniqueness violation has occurred (field level constraint).
202	An AT-MOST-ONE violation occurred (field level constraint).

See the Rocket Model 204 documentation wiki for information about BINARY and FLOAT field values:

http://m204wiki.rocketsoftware.com/index.php/Data_maintenance#Storing_data_in_fields

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  WORK-REC.
    05  WORK-SSN   PIC 9(9).
    05  WORK-NAME  PIC X(30).
•
•
•
O1  CALL-ARGS.
    05  RETCODE     PIC 9(5) COMP SYNC.
    05  FILESPEC    PIC X(13) VALUE "IN FILE EMPS;".
    05  EDITSPEC    PIC X(28) VALUE "EDIT (SSN,NAME)
                        (A(9),A(30));".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFSTOR" USING RETCODE, FILESPEC, WORK-REC, EDITSPEC.
```

IFSTRT call (IFAM1) *-mc,sc*

Function

The form of the IFSTRT call (START THREAD) that starts an IFAM1 thread connection to Model 204 performs the following actions:

- Allocates a single active thread
- Specifies the calling protocol to be used for the host language
- Sets certain Model 204 job parameters
- Establishes either a single or a multiple cursor type thread

Full syntax

```
IFSTRT(RETCODE,LANG_IND,PARM_LIST,PRO_LIST,THRD_TYP)
```

Compile-only syntax

A compile-only form of IFSTRT is not available.

Execute-only syntax

An execute-only form of IFSTRT is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LANG_IND	[I,i,r] The language indicator is a required input parameter that establishes the calling sequence convention to be used corresponding to the host language. The indicator specifies the format of parameters that are passed in subsequent calls. Specify one of the following integer values: 1 = PL/1 F-level, and BAL languages 2 = COBOL, FORTRAN, and BAL languages 3 = PL/1 with +Optimizer/Checkout compilers, VS/FORTRAN, and BAL languages **Note:** Any convention may be specified for use with the BAL language, and the BAL programmer must adhere to the convention that is specified when coding parameters.

Parameter	Description
PARM_LIST	[I,c,r] The parameter list is a required input parameter which specifies PARM entries that are set on the EXEC statement for the IFAM1 job. Specify a character string and append a semicolon (;), or specify a semicolon if no parameters are to be set. See the Rocket Model 204 documentation wiki for a description of the Model 204 parameters: http://m204wiki.rocketsoftware.com/index.php/List_of_Model_204_parameters **Note:** If the LIBUFF and LOBUFF parameters are to be set, include them in the parameter list. Do not specify the following parameters in the PARM list: ALTIODEV, NUSERS, and NSERVS.
PRO_LIST	[I,c,r] The prologue list is a required input parameter which specifies Model 204 User 0 parameters, such as page size and lengths of various work areas. Specify a character string and append a semicolon (;), or specify a semicolon if no parameters are to be set. (This entry corresponds to the first line of the SYSIN data set in a BATCH204 run.) See “Notes and tips” in the following section for restrictions that apply to User 0 login entries. See the Rocket Model 204 documentation wiki for a description of the Model 204 User 0 parameters: http://m204wiki.rocketsoftware.com/index.php/Defining_the_runtime_environment_(CCAIN)#Structure_of_CCAIN
THRD_TYP	[I,i,o] The thread type indicator is an optional parameter, which indicates either a single cursor or a multiple cursor IFSTRT thread. Specify either of the following integer values: 0 = Single cursor thread (default) 2 = Multiple cursor thread **Note:** If not specified, the thread indicator defaults to 0; you must specify a value of 2 to use a multiple cursor IFSTRT thread.

Notes and tips

Use the IFSTRT call to establish a connection to the Host Language Interface Model 204 service program.

Note that an IFAM1 job is single-threaded, and only one IFSTRT thread can be started in the job. You can specify a multiple cursor IFSTRT thread or use the default single cursor IFSTRT thread.

The IFSTRT protocol allows applications written in a host language to process against the Model 204 database using a particular class of HLI calls. See Chapter 5 for an overview of IFSTRT calls.

User 0 login restrictions

Certain restrictions apply for specifying User 0 login parameters (the PRO_LIST parameter). Rocket recommends the following actions when using a security subsystem, such as Security Server (formerly RACF), that performs login validation:

- Do not specify a user ID in the login for User 0. Note that if you do supply a user ID in the login, it must match the user ID of the owner of the address space; otherwise, the login fails.
- When the IFSTRT call processes the login parameter, do not code the password in the host language program. Model 204 interprets a password that is passed in the IFSTRT call as an invalid command.

Completion return code (RETCODE)

If the IFSTRT call is unsuccessful, Model 204 returns an error code for either of the following error conditions:

Code	Error condition
4	No IFSTRT thread was started; do not attempt to issue any other HLI calls. The HLI program code should check the return code from IFSTRT and continue processing only if the call was successful; for a return code of 4, either reissue IFSTRT until it is successful or stop job processing and give an error message.
80	No current thread. (Action: Call IFSTRT.)
90	An illegal IFSTRT call was made when a thread already exists.

Coding example (PL/1 Optimizer)

```
•
•
•
DCL NERR FIXED BIN (31) INIT (0);
DCL PARM CHAR (80) INIT('SYSOPT=144,LIBUFF=500,LOBUFF=500;');
DCL IFSTRT ENTRY(FIXED BIN(31), FIXED BIN(31),CHAR(*),CHAR(*));
•
•
•
CALL IFSTRT (NERR,3,PARM,'PAGESZ=6184,SPCORE=5000;');
•
•
•
```

IFSTRT call (IFAM2/IFAM4) *-mc,sc*

Function

The form of the IFSTRT call (START THREAD) that starts an IFAM2 or IFAM4 thread connection to Model 204 performs the following actions:

- Allocates a thread, making it currently active in the job
- Returns the thread identifier for use by IFDTHRD or IFSTHRD
- Specifies the calling protocol to be used for the host language
- Performs a user login
- Establishes either a single cursor or a multiple cursor type thread
- For a single cursor thread, determines whether retrieval or updating privileges are allowed

Full syntax (1)

```
IFSTRT(RETCODE,LANG_IND,LOGIN,THRD_TYP,THRD_ID)
```

Compile-only syntax

A compile-only form of IFSTRT is not available.

Execute-only syntax

An execute-only form of IFSTRT is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LANG_IND	[I,i,r] The language indicator is a required input parameter which establishes the calling sequence convention to be used corresponding to the host language. The indicator specifies the format of parameters that are passed in subsequent calls. Specify one of the following integer values: 1 = PL/1 F-level, and BAL languages 2 = COBOL, FORTRAN, and BAL languages 3 = PL/1 with +Optimizer/Checkout compilers, VS/FORTRAN, and BAL languages **Note:** Any convention may be specified for use with the BAL language, and the BAL programmer must adhere to the convention that is specified when coding parameters.

Parameter	Description

LOGIN	[I,c,r] The login information is a required parameter which supplies a valid Model 204 user ID and password that permit entry to the system. Specify the login as a character string using the following format:
	`userid [account];`
	`password[:new password];`

	where: *userid* is a character string that identifies the user who is logging into Model 204. *account* is an optional character string that supplies an account under which the user is logging into Model 204. *password* is a character string that allows the specified user to access Model 204. *new password* is an optional character string that changes the login password for the specified user, for future logins. See page 300 for the restrictions that apply to login entries when using a security subsystem, such as Security Server (formerly RACF), to perform login validation. See the Rocket Model 204 documentation wiki for a description of the login command: http://m204wiki.rocketsoftware.com/index.php/LOGIN_or_LOGON_command
THRD_TYP	[I,i,r] The thread type indicator is a required parameter that specifies the type of IFSTRT thread to be allocated. Specify one of the following integer values: 0 = Single cursor thread with read-only privileges 1 = Single cursor thread with update privileges 2 = Multiple cursor thread **Note:** The 0(read) and 1 (update) settings are valid for a single cursor IFSTRT thread which can be used in an multithreaded application. A thread type indicator of 0 allows a single cursor IFSTRT thread to be used only for retrieval, regardless of the file or group password that is used in a particular call. File updating by passing data from a retrieval-only Host Language Interface thread to an update thread can lead to logical inconsistencies. To prevent inconsistencies, start the thread as an update thread (1) and use a retrieval-only password to open a file. This provides share-mode (SHR) enqueuing and prevents updating from the thread. Files that are opened this way are also prevented from being marked physically inconsistent with a user restart or system crash.

Parameter	Description
THRD_ID	[O,i,r] The thread identifier is a required output parameter. Specify an integer variable. Model 204 returns a value that may be referenced using the IFDTHRD and IFSTHRD calls for thread switching in multithreaded applications.

Notes and tips

Use the IFSTRT call to establish a connection to the Host Language Interface Model 204 service program. For an IFAM2 connection, IFSTRT assumes a default channel name of IFAMPROD.

Note that an IFAM2 or IFAM4 job can be multithreaded. You can call IFSTRT more than once in a job to establish multiple threads, but only one IFSTRT thread is currently active and, for single cursor IFSTRT threads, each thread has its own current file or group, current record set, and current record. You can start single cursor and multiple cursor IFSTRT threads in the same job.

The IFSTRT protocol allows applications written in a host language to process against the Model 204 database using a particular class of HLI calls. See Chapter 5 for an overview of IFSTRT calls.

On a single cursor IFSTRT thread, IFSTRT together with IFFNSH initiates CPSORT checkpointing. See the *Rocket Model 204 Host Language Interface Programming Guide* for more information about CPSORT.

Login restrictions

Certain restrictions apply for specifying login information (the LOGIN parameter). Rocket recommends the following actions when using a security subsystem, such as Security Server, that performs login validation:

- Do not specify a user ID in the login for User 0. Note that if you do supply a user ID in the login, it must match the user ID of the owner of the address space, otherwise, the login fails.
- When the IFSTRT call processes the login parameter, do not code the password in the host language program. Model 204 interprets a password that is passed in the IFSTRT call as an invalid command.

Completion return code (RETCODE)

If the IFSTRT call is unsuccessful, Model 204 returns an error code for any of the following error conditions:

Code	Error condition
4	No IFSTRT thread was started; do not attempt to issue any other HLI calls. The HLI program code should check the return code from IFSTRT and continue processing only if the call was successful; for a return code of 4, either reissue IFSTRT until it is successful or stop job processing and give an error message.

Code	Error condition
90	A Model 204 is not yet available for user processing. Recovery may still be in progress. (Action: Call IFSTRT again.)
100	LOGIN failed.
400	Invalid language code (LANG_IND).

Coding example (COBOL)

This COBOL coding example specifies the following IFSTRT parameters:

- COBOL calling convention (language indicator is 2)
- Login account name USERABC
- Login password ECP
- Single cursor thread with read-only access (thread type is 0)

```
WORKING-STORAGE SECTION.
01  LOGIN-INFO.
    05 LOGIN       PIC X(12) VALUE 'USERABC;ECP;'.
01  CALL-PARMS  COMP SYNC.
    05 RETCODE     PIC 9(5).
    05 LANG-IND    PIC 9(5) VALUE 2.
    05 MODE        PIC 9(5) VALUE 0.
    05 THRD-NO     PIC 9(5).
•
•
•
PROCEDURE DIVISION.
INITIALIZATION.
  OPEN OUTPUT...
  CALL "IFSTRT" USING RETCODE, LANG-IND, LOGIN, MODE, THRD-NO.
  IF RETCODE IS NOT EQUAL TO ZERO
     GO TO ERROR-ROUTINE.
•
•
•
```

IFSTRTN call (IFAM2) *-mc,sc*

Function

The IFSTRTN call (START THREAD) starts an IFAM2 thread connection to Model 204 with a specified Host Language Interface Model 204 service program through the named channel.

Full syntax (4)

```
IFSTRTN |  IFSTRN
  (RETCODE,LANG_IND,LOGIN,THRD_TYP,THRD_ID,[SBSN:]CHAN)
```

Compile-only syntax

A compile-only form of IFSTRTN is not available.

Execute-only syntax

An execute-only form of IFSTRTN is not available.

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
LANG_IND	[I,i,r] The language indicator is a required input parameter which establishes the calling sequence convention to be used corresponding to the host language. The indicator specifies the format of parameters that are passed in subsequent calls. Specify one of the following integer values: 1 = PL/1 F-level, and BAL languages 2 = COBOL, FORTRAN, and BAL languages 3 = PL/1 with +Optimizer/Checkout compilers, VS/FORTRAN, and BAL languages **Note:** Any convention may be specified for use with the BAL language, and the BAL programmer must adhere to the convention that is specified when coding parameters.

Parameter	Description

LOGIN	[I,c,r] The login information is a required parameter which supplies a valid Model 204 user ID and password that permit entry to the system. Specify the login as a character string using the following format:
	`userid [account];`
	`password[:new password];`
	where:
	userid is a character string that identifies the user who is logging into Model 204.
	account is an optional character string that supplies an account under which the user is logging into Model 204.
	password is a character string that allows the specified user to access Model 204.
	new password is an optional character string that changes the login password for the specified user, for future logins.

	See page 304 for the restrictions that apply to login entries when using a security subsystem, such as Security Server, to perform login validation. See the Rocket Model 204 documentation wiki for a description of the login command: http://m204wiki.rocketsoftware.com/index.php/LOGIN_or_LOGON_command
THRD_TYP	[I,i,r] The thread type indicator is a required parameter that specifies the type of IFSTRT thread to be allocated. Specify one of the following integer values: 0 = Single cursor thread with read-only privileges 1 = Single cursor thread with update privileges 2 = Multiple cursor thread **Note:** The 0 (read) and 1 (update) settings are valid for single cursor IFSTRT threads and can be used for a multithreaded application. A thread type indicator of 0 allows a single cursor IFSTRT thread to be used only for retrieval, regardless of the file or group password that is used in a particular call. File updating by passing data from a retrieval-only Host Language Interface thread to an update thread can lead to logical inconsistencies to the updated file during a roll forward. To prevent inconsistencies, start the thread as an update thread (1) and use a retrieval-only password to open a file. This provides share-mode enqueuing and prevents updating from the thread. Files that are opened this way are also prevented from being marked physically inconsistent with a user restart or system crash.
THRD_ID	[O,i,r] The thread identifier is a required output parameter. Specify an integer variable. Model 204 returns a value that may be referenced using the IFDTHRD and IFSTHRD calls for thread switching in multithreaded applications.

Parameter	Description
[SBSN:]CHAN	[I,c,r] The channel (CHAN) name is a required input parameter which specifies the CRAM, IUCV, or VMCF communications channel name for a particular service program. Specify the name as an eight-character string. The subsystem name (SBSN:) is optional. You can use it when you want to override the default. Specify the name as a four-character string, plus colon (:). **Note:** Do *not* append a semicolon. If the host language is PL/1, pass the address of the string using a based variable that overlays the original parameter.

Notes and tips

Use the IFSTRTN call to establish an IFAM2 connection using a specific Host Language Interface Model 204 service program. IFSTRTN performs the same function as IFSTRT. The IFSTRTN call includes a sixth parameter, which is not available with IFSTRT that is used to specify the communications channel name for the service program.

For more information about CRAM (Cross Region Access Method) and Host Language Interface Model 204 service programs, see Chapter 3 and the *Rocket Model 204 Host Language Interface Programming Guide*.

Note that an IFAM2 job can be multithreaded. You can call IFSTRTN more than once in a job to establish multiple threads, but only one thread is currently active and, for single cursor IFSTRT threads, each thread has its own current file or group, current record set, and current record.

IFSTRTN uses IFSTRT protocols, and allows applications written in a host language to process against the Model 204 database using a the same class of HLI calls that are available with IFSTRT. You can start a single cursor or multiple cursor IFSTRT thread using the IFSTRTN call.

IFSTRT, together with IFFNSH, initiates CPSORT checkpointing. See the *Rocket Model 204 Host Language Interface Programming Guide* for a detailed description of CPSORT checkpointing.

Login restrictions

Certain restrictions apply for specifying login information (the LOGIN parameter). Rocket recommends the following actions when using a security subsystem, such as Security Server, that performs login validation:

- Do not specify a user ID in the login for User 0. Note that if you do supply a user ID in the login, it must match the user ID of the owner of the address space, otherwise, the login fails.
- When the IFSTRT call processes the login parameter, do not code the password in the host language program. Model 204 interprets a password that is passed in the IFSTRT call as an invalid command.

Coding example (COBOL)

This COBOL coding example specifies the following IFSTRT parameters:

- COBOL calling convention (language indicator is 2)
- Login account name USERABC
- Login password ECP
- Single cursor thread with read-only access (thread type is 0)
- Channel name M204CHNB

```
WORKING-STORAGE SECTION.
01  LOGIN-INFO.
    05 LOGIN      PIC X(12) VALUE 'USERABC;ECP;'.
01  CALL-ARGS.    COMP SYNC.
    05 RETCODE    PIC 9(5).
    05 LANG-IND   PIC 9(5) VALUE 2.
    05 MODE       PIC 9(5) VALUE 0.
    05 THRD-NO    PIC 9(5).
    05 CHAN-NAME  PIC X(13) VALUE "SSN1:IFAMCHNL".
*   05 CHAN-NAME  PIC X(8) VALUE "IFAMCHNL".
•
•
•
PROCEDURE DIVISION.
INITIALIZATION.
    OPEN OUTPUT...
    CALL "IFSTRTN" USING RETCODE, LANG-IND, LOGIN, MODE,
    THRD-NO, CHAN-NAME.
    IF RETCODE IS NOT EQUAL TO ZERO
    GO TO ERROR-ROUTINE.
•
•
•
```

IFUPDT call *-mc*

Function

The IFUPDT call (UPDATE) updates the current record with specified data. IFUPDT specifies the cursor for the current record.

Full syntax (115)

```
IFUPDT(RETCODE,DATA_AREA,CURSOR_NAME,EDIT_SPEC,UPDT_NAME,
%VARBUF,%VARSPEC)
```

Compile-only syntax (116)

```
IFUPDTC|IFUPDC(RETCODE,CURSOR_NAME,EDIT_SPEC,UPDT_NAME)
```

Execute-only syntax (117)

```
IFUPDTE|IFUPDE(RETCODE,DATA_AREA,UPDT_NAME,%VARBUF,%VARSPEC)
```

Parameters

Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required first parameter. The code is a binary integer value.
DATA_AREA	[I,c,r] The data area is a required input parameter which specifies the address of the user's data area. Specify a character string variable. The area contains the data that is used to update the fields which is defined in the EDIT_SPEC parameter.
CURSOR_NAME	[I,s,r] The cursor name is a required input parameter which specifies the name of the cursor whose current record is to be updated. This is a short character string, the name previously assigned to the cursor in a corresponding IFOCUR call. See page 226 for a description of the cursor name for the IFOCUR call.

EDIT_SPEC	[I,c,r] The edit specification is a required input parameter which defines the fields that are to be updated in the current record. The specification describes the format of the data which is read at the data area (described above). Specify a character string using one of the following LIST, DATA, or EDIT format options:
	`LIST (fieldname list);`
	`DATA;`
	`EDIT (fieldname list) (edit formats);`
	where:
	fieldname list is required for the LIST or EDIT specification and specifies a field name or names. Specify elements in the field name list using one of the following options:
	• fieldname
	• fieldname(n)
	• fieldname(*)

Parameter	Description
	• fieldname(%variable) • fieldname(+n) • fieldname(+%variable) where: • *fieldname* updates the first occurrence of the named field. This is equivalent to fieldname(1). If the field does not occur in the current record, IFUPDT adds it. • *fieldname(n)* updates the nth occurrence of the named field for a multiply occurring field. If the nth occurrence does not exist in the current record, IFUPDT adds it. • *fieldname(*)* adds the named field to the current record. If the field already exists in the current record, IFUPDT adds another occurrence of the field. • *fieldname(%variable)* retrieves the occurrence of the field specified by the %VARBUF and %VARSPEC parameters. If the nth occurrence does not exist in the current record, IFUPDT adds it. • *fieldname(+n)* inserts a new occurrence of the named field to the current record. This is analogous to the INSERT statement in SOUL and is useful for adding new occurrences of a field where the order of the values is important. Insert the new occurrence as the nth occurrence. • *fieldname (+%variable)* inserts a new occurrence of the field into the current record. The occurrence number is retrieved from the %VARBUF and %VARSPEC parameters. **Note:** If there is a current nth (or %variable) occurrence, make it the one after the nth occurrence. If n is greater than the current number of occurrences, add the new occurrence at the end. If the field does not occur in the current record, add it. If n is 0 or is not specified, treat it as though n=1 and insert the field as the first occurrence. *edit format* is required in the EDIT specification and specifies a code or codes, which indicate(s) the format of the data to be returned for the named field in the field name list-edit format pair. See “Using EDIT format codes for an updating call” on page 328 for a detailed description of the EDIT format codes that are used with IFUPDT. **Note:** See Chapter 7 for a description of LIST, DATA, and EDIT formatting.

Parameter	Description
UPDT_NAME	[I,s,o] The name of the IFUPDT compilation is an optional input parameter. If specified, Model 204 saves the compilation using this name. Specify the name as unique, and as a short character string (maximum 32 characters). The first character in the name must be alphanumeric, and the name must begin with a letter (A–Z or a–z) which may be followed by a letter, a digit (0–9), a period (.), or underscore (_). A null value is equivalent to omitting the name parameter, and is not valid. **Note:** You may optionally specify the name of a saved IFFTCH compilation for the IFUPDT call. In other words, IFFTCH and IFUPDT may share compilations.
%VARBUF	[I,c,o] The variable buffer is an optional input parameter that addresses a data area which accommodates up to 255 bytes of data per value. The buffer contains values or expressions which are defined by the %VARSPEC parameter, below, to be assigned to %variables. Specify a character string. See the Rocket Model 204 documentation wiki for information about %variables: http://m204wiki.rocketsoftware.com/index.php/Using_variables_and_values_in_computation
%VARSPEC	[I,c,o] The variable specification describes the format of the data that is contained in the %variable parameter, and lists the %variables to be assigned. %VARSPEC specifies the contents of the variable buffer, described above. Specify a character string which follows a LIST, DATA, or EDIT syntax. %VARSPEC is a required input parameter if %VARBUF is specified.

Notes and tips

Use the IFUPDT call to change or delete fields in an existing record, or to add new fields to a record.

When FOPT=X'10' and the date/time stamp feature is installed, the IFUPDT function is *not* supported for DTS files.

Issue the IFUPDT call after an IFFTCH or IFSTOR call. See the IFFTCH and the IFSTOR calls.

The IFUPDT call operates in the multiple cursor environment similarly to the single cursor IFPUT call. See the IFPUT call.

Completion return code (RETCODE)

If the IFUPDT call is unsuccessful, Model 204 returns an error code for either of the following conditions:

Code	Error condition
10	Model 204 encountered invalid data values for BINARY and FLOAT numeric field for a file having FILEMODL set to NUMERIC VALIDATION.
200	A uniqueness violation occurred (field level constraint).
202	An AT-MOST-ONE violation occurred (field level constraint).

See the Rocket Model 204 documentation wiki for information about BINARY and FLOAT field values:

http://m204wiki.rocketsoftware.com/index.php/Data_maintenance#Storing_data_in_fields

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  WORK-REC.
    05  WORK-SSN      PIC 9(9).
    05  WORK-NAME     PIC X(30).
•
•
•
01  CALL-ARGS.
    05  RETCODE       PIC 9(5) COMP SYNC.
    05  CURSOR-NAME   PIC X(5) VALUE "CUR1;".
    05  EDIT-SPEC     PIC X(28) VALUE "EDIT (SSN,NAME)
                          (A(9),A(30));".
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFUPDT" USING RETCODE, WORK-REC, CURSOR-NAME,
        EDIT-SPEC.
```

IFUTBL call *-mc,sc*

Function The IFUTBL call (USER TABLE) resets the specified Model 204 UTABLE (user table) parameters.

Full syntax (64)

```
IFUTBL(RETCODE,PARM_LIST)
```

Compile-only syntax A compile-only form of IFUTBL is not available.

Execute-only syntax An execute-only form of IFUTBL is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is a required output parameter. The code is a binary integer value.
PARM_LIST	[I,c,r] The parameter list is a required input parameter which specifies the name and value pair for each user table parameter whose value is reset. Specify a character string using the following format: `parm1=value1 [,parm2=value2•••];` where: *parm1* is the name of the user table parameter to be reset, and parm2 is the name of a second parameter to be reset. Additional parameters may be specified in a name-value pair. Specify the keyword name of any of the following UTABLE parameters: HTLEN, LNTBL, LVTBL, LFSCB, LPDLST, LXTBL, LFTBL, LQTBL, LGTBL, LSTBL, LITBL, LTTBL, MAXHDR, MAXTRL *value1* is the new value for the specified parameter in the first pair, and value2 is the new value for the specified parameter in the second pair. A value is required for each name that is specified in the list. Values may be specified in decimal form, such as 193, or in hexadecimal form, such as X'C1'. For example, the specification MAXHDR=128 is equivalent to MAXHDR=X'80'. You may specify more than one name=value pair, separating each by a comma or a blank.

Notes and tips Use the IFUTBL call to reset certain user table parameters, which enables you to change the size of Model 204 server tables. The IFUTBL call is not available in IFAM1, but is valid on *all* types of IFSTRT threads in IFAM2 and IFAM4.

Note: If the size of FTBL or XTBL is changed, any open file or group is closed. Changing the size of any table causes any compiled calls and %variables to be flushed.

See the Rocket Model 204 documentation wiki for information about server tables and calculation of server table sizes:

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

Coding example (COBOL)

```
WORKING-STORAGE SECTION.
01  CALL-ARGS.
    05  RETCODE  PIC 9(5)  COMP SYNC.
    05  UTABLE   PIC X(12) VALUE 'MAXHDR=128;'.
•
•
•
PROCEDURE DIVISION.
•
•
•
  CALL "IFUTBL" USING RETCODE, UTABLE.
```

IFWRITE call *-di*

Function The IFWRITE call (WRITE) sends a line of input to Model 204.

Full syntax (10)

```
IFWRITE|IFWRIT(RETCODE,LINE_AREA,LINE_LEN)
```

Compile-only syntax A compile-only form of IFWRITE is not available.

Execute-only syntax An execute-only form of IFWRITE is not available.

Parameters Specify the parameters in the syntax order shown above.

Parameter	Description
RETCODE	[O,i,r] The Model 204 return code is the required output parameter. The code is a binary integer value.
LINE_AREA	[I,c,r] The line area is a required input parameter which is the input line to be sent to Model 204.
LINE_LEN	[I,i,o] The line length is an optional input parameter, which specifies the transfer length for IFWRITE. This parameter determines the maximum line length for the IFWRITE call. If this parameter is present, it overrides any value specified in IFDIAL or IFDIALN. For PL/1, the length is the minimum of this value plus the string length. See page 313 for more information about the input line length.

Notes and tips Use the IFWRITE call only with an IFDIAL thread to transmit data to Model 204.

When using IFWRITE, note that %VAR must be a string equal to TERMINAL and either a PREPARE or IDENTIFY imagename is required prior to writing a new image. See the *Rocket Model 204 Host Language Interface Programming Guide* for more information about coding IFDIAL applications.

You may specify a different buffer length with each call by specifying the line length in IFWRITE. See the next page for detailed information about the IFWRITE data transfer length.

Completion return code (RETCODE) Code your IFDIAL application to check the return code for the following values:

Code	Required action
1	Call IFWRITE next to provide Model 204 with input.
2	Call IFREAD next to get more output from Model 204.

See the *Rocket Model 204 Host Language Interface Programming Guide* for more information about coding IFDIAL applications.

If the IFWRITE call is unsuccessful, Model 204 returns an error code for either of the following error conditions:

Code	Error condition
12	IFWRITE call not accepted (IFREAD call expected).
100	No current Model 204 connection exists or the connection is lost.

Transfer length for input to Model 204

The parameters in effect during the execution of the IFWRITE call determine the length of data transferred to Model 204. Several factors determine the length.

The first factor is the PL/1 string length; for PL/1 compilers (F-level, Optimizer and Checkout) use a dope vector when passing character string arguments. This dope vector contains the maximum length of the string and its address. For strings declared as VARYING, it also contains the current length.

Next, the transfer length for input to Model 204; this value is based on the following order of precedence, from highest to lowest:

1. The optional length parameter in the IFWRITE call.

   For PL/1, if this length is greater than the current string length, the current string length is used.

   Note: This value is in effect only for this specific IFWRITE call.

2. The optional default length parameter in the IFDIAL call. This new default remains in effect until IFHNGUP is called.

   For PL/1, if this length is greater than the current string length, the current string length is used.

3. The standard default length is one of the following:
   - 252 for COBOL, FORTRAN, and Assembler
   - PL/1 current string length (dope vector)

Note that the maximum length of a data area that can be transferred over an IFDIAL thread is 32763 bytes. For all languages, if the transfer length is greater than the CRAM buffer size, the data is truncated and the length adjusted.

See the Rocket Model 204 documentation wiki for more information about buffer size parameters:

http://m204wiki.rocketsoftware.com/index.php/Defining_the_Runtime_Environment_(CCAIN)

Overview of IFWRITE data transfer

Table summarizes the relationship between the parameters that determine the IFWRITE data transfer length.

Table uses the following codes:

- *Lang=n* is the language indicator specified in the IFDIAL or IFDIALN call.
- *LENGTH1* is the default length parameter in the IFDIAL call.
- *LENGTH2* is the length parameter in the IFREAD call.
- *FIXED* is a PL/1 string argument that is declared as fixed.
- *VARYING* is a PL/1 string argument that is declared as varying.
- *CURRLEN* is the current length of the PL/1 string.
- *LENGTHn* signifies that the parameter was specified.
- ¬*LENGTHn* signifies that the parameter was not specified.
- *min(l,m)* is the transfer length, which is the minimum value of l and m.

Table 6-6. IFWRITE data transfer length

Parameters in effect			**Transfer length**
IFDIAL		**IFREAD**	
Lang=1	LENGTH1	LENGTH2	
	LENGTH1	¬LENGTH2	
	LENGTH1	LENGTH2	
	LENGTH1	¬LENGTH2	
Lang=2	LENGTH1	LENGTH2	LENGTH2
	LENGTH1	¬LENGTH2	LENGTH1
	¬LENGTH1	LENGTH2	LENGTH2
	¬LENGTH1	¬LENGTH2	252
Lang=3	LENGTH1	LENGTH2, FIXED	min(LENGTH2,MAXLEN)
	LENGTH1	¬LENGTH2, FIXED	min(LENGTH1,MAXLEN)
	¬LENGTH1	LENGTH2, FIXED	min(LENGTH2,MAXLEN)
	¬LENGTH1	¬LENGTH2, FIXED	CURRLEN
Lang=4	LENGTH1	LENGTH2, VARYING	min(LENGTH2,MAXLEN)
	LENGTH1	¬LENGTH2, VARYING	min(LENGTH1,MAXLEN)
	¬LENGTH1	LENGTH2, VARYING	min(LENGTH2,MAXLEN)
	¬LENGTH1	¬LENGTH2, VARYING	CURRLEN

Coding example (Assembler)

```
•
•
•
  CALL IFDIAL...
•
•
•

  CALL IFREAD...
•
•
•
  MVC  WLEN(4),=F’9’
  CALL IFWRITE,(RETCODE,LOGONMSG,WLEN),VL
  CLC  RETCODE(4),=F’2’
  BNE  END
•
•
•
END    ABEND 999,DUMP
•
•
•
RETCODE       DC  F’0’
LOGONMSG      DC  C’LOGON USR’
WLEN          DC  F’0’
  END
```

7

Field Formatting Options for HLI Calls

Overview

This chapter describes in detail the field formatting options for specifying data that is passed between Model 204 and an HLI application that uses an IFSTRT thread.

Use the information in this chapter to code the edit specification parameter in an IFFTCH, IFGET, IFGETX, IFGETV, IFMORE, IFMOREX, IFUPDT, IFSTOR, or IFPUT call.

The following calls retrieve data:

- IFFTCH
- IFGET
- IFGETX
- IFGETV
- IFMORE
- IFMOREX

The following calls perform updating functions against the database:

- IFUPDT
- iFSTOR

- IFPUT

Note: IFFTCH, IFUPDT, and IFSTOR are used on a multiple cursor IFSTRT thread; IFGET, IFGETX, IFGETV, IFMORE, IFMOREX, and IFPUT are used on a single cursor IFSTRT thread.

For more information

Refer to Chapter 6 for descriptions of the IFFTCH, IFGET, IFGETX, IFGETV, IFMORE, IFMOREX, IFUPDT, IFSTOR, and IFPUT calls.

Using a LIST specification for a retrieval call

An edit specification parameter specifies a LIST format option for an IFFTCH, IFGET, IFGETX, IFMORE, or IFMOREX (retrieval) call using the following syntax:

```
LIST (fieldname list);
```

An edit specification parameter specifies a LIST format option for an IFFTCH call against a value set cursor or an IFGETV call using the following syntax:

```
LIST;
```

The retrieval call returns the value of the fields named in the field name list in the following format:

```
'field value1' 'field value2' ••• 'field value N';
```

where:

- Single quotation marks enclose each value and blanks separate the values.
- Single quotation marks in the data itself are converted to two quotes.
- Fields that are not contained in the record are returned as two single quotes (' '), which is the null value.
- You may use a %variable in the fieldname list in the LIST specification. The value for the %variable is specified in the %VARBUF and %VARSPEC parameters.

Using a DATA specification for a retrieval call

An edit specification parameter specifies a DATA format option for an IFFTCH, IFGET, IFGETX, IFMORE, or IFMOREX call using either of the following syntax forms:

```
DATA (fieldname list);
```

```
DATA;
```

The retrieval call returns the value of the fields named in the field name list in the following format:

```
fn1='value1' fn2='value2'••• fnN='valueN';
```

where fields that are not contained in the record appear as fn=''.

Note that the DATA format option without a field name list retrieves all fields in the record and formats data in the same manner.

Using an EDIT specification for a retrieval call

An edit specification parameter specifies an EDIT format option for an IFFTCH, IFGET, IFGETX, IFMORE, or IFMOREX call using either of the following syntax forms:

```
EDIT (fieldname list) (edit formats);
```

```
EDIT (fieldname list1) (edit format1)
 (fieldname list2)(edit format2);
```

An edit specification parameter specifies an EDIT format option for an IFFTCH call against a value set cursor or an IFGETV call using the following syntax:

```
EDIT (edit format);
```

The retrieval call returns the value of the field(s) named in the field name list in the format that is specified. See page 321 for a listing of edit codes that may be used in an EDIT specification for IFFTCH, IFGET, IFGETV, IFGETX, IFMORE, or IFMOREX.

Guidelines for specifying an EDIT format

When specifying an EDIT format, the following guidelines apply:

- You may insert blanks before and/or after the field name list and edit format entries in the EDIT specification.
- Separate individual entries inside the edit format list with a comma.
- Repetition factors and parentheses are allowed. Blanks following a repetition factor or surrounding commas and parentheses are optional.

  The following examples illustrate repetition factors:

  ```
  2A(10),J(2)
  ```

  is equivalent to:

  ```
  A(10),A(10),J(2)
  ```

  and

  ```
  2(A(10),J(2))
  ```

  is equivalent to:

  ```
  A(10),J(2),A(10),J(2)
  ```

- For each name in the field name list, one edit format is selected. Repetition factors are expanded before the selection occurs. COL, POS, and X are not selected by field names, but are executed as they are encountered in the list of edit formats.
- You can use a %variable in the field name list in the EDIT specification. The value for the %variable is specified in the %VARBUF and %VARSPEC parameters.

  If the edit formats are exhausted before the field name list, selection wraps around to the beginning of the list of formats. Model 204 ignores extra items.

Using the V format

When the V format is specified, the edit format specified immediately after the V determines the format of the data values.

For example, if the EDIT specification contains VA(4), the following hexadecimal data would be stored in the data buffer to represent the three values, ABCD, 1234, and XYZ:

```
00000003C1C2C3C4F1F2F3F4E7E8E940
```

Note that the following edit formats may not be specified after the V:

- M and V, which describe a collection of values.
- COL, POS, and X, which alter the data buffer pointers without manipulating a value.
- A left parenthesis (( ), which indicates a group of formats to be repeated.

V processes field occurrences in the same manner as the M function. The field name corresponding to the V format in the field name list is specified as field name(n), the first n-1 occurrences of the field are omitted.

Note that the VL format is equivalent to the M format, except that the data values are preceded by a 4-byte count rather than by a 1-byte count.

Handling fields that do not occur in the record

Model 204 handles fields that do not occur in the record in the following ways:

- If the first or nth occurrence of a field is specified and the field is not present in the record or fewer than n occurrences exists, then the appropriate pad characters are supplied for A(n), J(n), L(n), and U(n) formats, or zero length is indicated for L and M formats.

  No characters are returned for A and J formats. A numeric zero is returned in the appropriate format for the B, P, E, Z, and F formats.

- If all occurrences are specified and there are no occurrences of the field in the record, no edit formats are used for that field. Place requests for fieldname(*) at the end of a list or in a separate IFFTCH, IFMORE, or IFMOREX call.
- Fields with the INVISIBLE attribute are treated as if the field does not occur in the record.

Examples of numeric edit format conversion

Table 7-1 shows examples of how Model 204 converts data for a numeric edit format specification.

Table 7-1. Examples of numeric edit format conversion

Edit format specification	Model 204 data (character)	User data (hexadecimal)
Z(5,2)	18.21	F0 F1 F8 F2 C1
Z(5,2)	-18.21	F0 F1 F8 F2 D1
P(5,2)	18.21	01 82 1C
P(3)	-372	37 2D
B(31,0)	4095	00000FFF
B(15,2)	10.25	0029 (binary 001010.01)
F(4)	2	41200000

Using EDIT format codes for a retrieval call

Table 7-2 describes the edit format codes that may be specified in the (edit format) with the EDIT option in an IFFTCH, IFGET, IFGETV, IFGETX, IFMORE, or IFMOREX (retrieval) call.

Table 7-2. EDIT format codes used with a retrieval call

Code	Description
A	Place an n-character field value into n characters.
A(n)	Left-justify a field value in an n-character area. This format pads with blanks if the field value is less than n or truncates if the field value is greater than n (the maximum of n is 255).

Table 7-2. EDIT format codes used with a retrieval call (Continued)

Code	Description
B(precision,scale)	The value is returned in two's complement binary form. Precision specifies the number of bits in the numeric value, exclusive of the sign bit. Only two precision types are supported: 15 (halfword) and 31 (fullword). If both precision and scale are omitted, the precision defaults to 15. Scale specifies the number of digits to the right of an implied binary point. The scale must not exceed the precision. If scale is omitted, it defaults to 0, indicating an integer value. A maximum of 12 significant bits of a fraction are returned. If the scale is more than 12, the low-order bits are 0
COL(n) or COLUMN(n)	Adds blanks to character position n in the data area. Specifying a column which has already been passed over is an error.
E(total length,significant digits,fractional digits)	Convert numeric format (FLOAT, BINARY, CODED, NON-CODED) numbers to exponential format. The total length includes the number of significant digits as well as space for E and the exponent, and space for both any decimal point and for any positive and/or negative signs. An insufficient total length results in a truncated representation of the given number. The number of significant digits is the total of whole number digits plus fractional digits. The maximum number of significant digits used in conversions and mathematical operations by Model 204 is 15 (extra digits are rounded to 15).
F(n)	The value is returned as a floating-point number. If n is 16, the value is truncated or expanded to an extended precision, floating-point number (occupies 16 bytes and holds up to 31 significant digits). If n is 8, the character string is truncated or expanded to a long precision, floating-point number (occupies 8 bytes and holds up to 15 significant digits). If n is 4, the character string is first truncated or expanded to a long precision, floating-point number and is then rounded and truncated to a short precision number (occupies 4 bytes and holds up to 6 significant digits). Decimal floating-point fractions are represented internally as imperfect, base 16, floating-point numbers. To provide exact equality when two floating-point values are compared, the numbers are rounded. Also, when a floating-point value is too large or too small for its field or %variable, the value is truncated or expanded so that it correctly fits. Rounding, truncation, or expansion occurs in the following cases: • Rounding occurs after each arithmetic operation involving floating-point numbers. • If the field is KEY, the value to be indexed is rounded. • If the field is used in a direct Table B search in an IFFIND call, the value to be searched for is rounded to long precision (8 bytes), and each field accessed in Table B is likewise rounded before comparison. • If the field will be used in a sort key, the value is rounded before being concatenated to the key. • If the source number is a floating-point number longer than the field or %variable into which it is stored, the source number is rounded and truncated so that it correctly fits. • If the source number is a floating-point number shorter than the field, the source number is rounded to its maximum significant digits and expanded with zeros to the longer precision.
J	Same as A.

Table 7-2. EDIT format codes used with a retrieval call (Continued)

Code	Description
J(n)	Right-justify a field value in an n-character area. Pads with blanks if the field value is less than n, or truncates if the field value is greater than n (the maximum value of n is 255).
L	The first byte is set to the number of data bytes that follow it. For example, 'ABC' would appear as the four-byte string X'03C1C2C3'.
L(n)	Same as J(n) except pads with binary zeros instead of blanks.
M	Collect all the occurrences of a field into a single data area. This area begins with a byte containing the number of occurrences. Each occurrence follows in L format. Specifying fieldname(n) in the field name list causes the first n-1 occurrences of the field to be omitted.
M(n)	Like M but each occurrence is in L(n) format.
P(precision,scale)	The value is returned in packed-decimal form. Precision specifies the number of digits (not bytes) in the numeric value, exclusive of the sign. The maximum precision for a packed-decimal field is 15. If an even precision is specified, one extra high-order 0 digit is returned. If both precision and scale are omitted, the precision defaults to 5. Scale specifies the number of digits to the right of an implied decimal point. The scale must not exceed the precision. If scale is omitted, it defaults to 0, indicating an integer value. No error is indicated if the fractional portion of a value is truncated.
POS(n)	Skip to character position n in the data area, bypassing the original contents from the application program. Specifying a column which has already been passed over is an error.
U	Same as A. The U function can be used with unformatted data such as bit strings or floating-point numbers.
U(n)	Left-justify a field value in an n-character area; pads with binary zeros or truncate on the right as appropriate (n<255).
V	Collect all occurrences of a field into a single data area. The area begins with a four-byte count of the number of data values in the area. The format of the values is determined by the second edit format, which must be specified immediately after the V. An example and a summary of restrictions appears on page 320.
X(n)	Add n blanks in the data area.
Z(precision, scale)	The value is returned in zoned-decimal form. The meaning of precision and scale is identical to that of the P format above except no padding is required for an even precision.

Using a LIST specification for an updating call

An edit specification parameter specifies a LIST format option for an IFUPDT, IFSTOR, or IFPUT call using the following syntax:

```
LIST (fieldname list);
```

The fields and values in the field name list must be specified in the following format:

```
'field value1' 'field value2' ••• 'field value N';
```

where:

- Field values in the data area correspond to the names in the field name list.
- Enclose the values in single quotation marks and separate them with blanks.
- Two consecutive quotation marks in a field value causes a single quotation mark to be retained in the stored data.
- Leading and trailing blanks within the enclosing quotation marks are retained.
- A field value specified as '' (the null value) causes an existing field to be deleted or, if the field does not exist in the record, no action is taken.
- You may use a %variable in the fieldname list in the LIST specification. The value for the %variable is specified in the %VARBUF and %VARSPEC parameters.

Using a DATA specification for an updating call

An edit specification parameter specifies a DATA format option for an IFUPDT, IFPUT, or IFSTOR call using the following syntax:

```
DATA;
```

The value of the fields named in the field name list must be specified in the following format:

```
fieldname1='fieldvalue1' fieldname2='fieldvalue2' •••
fieldnameN='fieldvalueN';
```

where:

- Follow each field name with an equal sign (=) and enclose each new field value in single quotation marks.
- Separate field name=value pairs with blanks.
- Two consecutive single quotation marks in a field value are stored as a single quotation mark in the record.
- The null value, fieldname='', causes the specified field to be deleted from the record. Or, if the field did not previously exist in the record, no action is taken.
- The DATA specification refers to the first occurrence of each field specified.

Using an EDIT specification for an updating call

An edit specification parameter specifies an EDIT format option for an IFUPDT, IFSTOR, or IFPUT call using the following syntax:

```
EDIT (fieldname list) (edit formats);
```

The format of the fields named in the field name list must be specified in the edit format. See page 328 for a list of the edit codes that are used in an EDIT specification for IFUPDT, IFSTOR, or IFPUT. See Chapter 6 for more information about specifying the field name elements.

Guidelines for specifying an EDIT format

When specifying an EDIT format for an IFUPDT, IFPUT, or IFSTOR updating call, the following guidelines apply:

- The value in the data area used with the EDIT specification does not require a final semicolon.
- Each name in the field name list corresponds in sequence to an edit format. Extra edit formats are ignored.
- If the list of edit formats is too short, Model 204 wraps around to the beginning of the list.
- You may use a %variable in the field name list in the EDIT specification. The value for the %variable is specified in the %VARBUF and %VARSPEC parameters.

Specifying significant digits using A, E, J, L, M, and U formats

For numbers passed as parameters in A, E, J, L, M, or U specifications, Model 204 ignores significant digits past 15 and treats them as zeros, regardless of precision.

Specifying a length for the E format

When you specify the E format for an updating function call, Model 204 ignores the significant digits and fractional digits parameters, but the total length specified must be accurate.

If the updating function call receives a total length that does not convert to a floating-point number, Model 204 displays an error message indicating that data is inconsistent. The number is not stored and the HLI call that produced the message receives a return code of 4.

Using the G format

The G (Generic) format allows you to perform complex updates to specific field values when an HLI program is executed. Through the use of a descriptor byte, you can use the G format to:

- Change the value of a field=value pair in a record
- Delete a specific value of the field in a record
- Bypass a specific occurrence of a field
- Store a zero-length string value in a field
- Select the data type at execution time rather than compile time

If you use the G format, Model 204 expects two items in the buffer for each field; the descriptor byte, and the new or updated field value (there are certain exceptions to this, as described below). The descriptor byte contains the following information:

- Bits 0 and 1 describe the format of the value to be stored or searched:

Bit	Meaning
00	DELETE the field specified. If delete is used, Model 204 does not expect a value to follow in the data buffer. If there is another value in the buffer, Model 204 assumes that it belongs to the next edit format.
01	Update or insert this field as an 8-byte FLOAT value.
10	Update or insert this field as a counted STRING value. For a counted string value, the first byte contains the length of the string, followed by the string itself. You can insert a zero-length string into a field using this format.
11	Unused

- Bits 2 to 5 are unused.
- Bit 6 indicates that you want to search for a specific value of a field to update or delete.

  When set, bit 6 indicates that the data buffer contains two values, a value to search for, and a value to store in its place. Each value must be preceded by a descriptor.

  If the descriptor bit for the second value in the buffer is set to '00' with no data following, the field occurrence is deleted.
- Bit 7 is the bypass field occurrence indicator. If this bit is set, Model 204 bypasses the specified field entirely.

Exceptions to G format usage

Although you can use the G format for all updating calls, there are some circumstances for which searching for specific values (using bit 6) or deleting field values (setting all bits to 0) are incompatible. You cannot use the G format to search by value or delete values for the following types of fields:

- Sort key field
- Hash key field
- %variable (%VARBUF and %VARSPEC parameters)
- When adding fields using the field value(*) format

Changing a specific value

For example, if Joan Darcy is transferred from Portland, OR to Pittsburgh, PA, you can use the G format to change the address in her employment record as follows:

```
EDIT(G)(G)

'10000010''8PORTLAND''10000000''APITTSBURGH'
'10000010''2OR''10000000''2PA'
```

If you change the value of the field to a zero-length string, that value is stored in the field.

Deleting a specific value

If, for example, Hadrian Wall is finally old enough to have his own automobile insurance and can be deleted from his parents' insurance policy, you can delete his field as follows:

```
EDIT (G)

'10000010''7HADRIAN'' B'00000000'
```

Note that in this case, the first two bits are 0, which indicates a DELETE, the remaining bits are also 0, and there is no second data value in the buffer.

Bypassing a field occurrence

For example, to bypass the third field in the found set of records, use the G format as follows:

```
EDIT (A)(F)(G)(F)

.
.
.
'00000001'
```

Selecting the data type

To select the data type at execution time, use just the first two bits in the G format. For example, to select only string data:

```
EDIT(G)

'10000000'
```

Specifying the U format with floating-point values

If you specify the U format with floating-point values, Model 204 interprets the values as nonnumeric strings and stores them as such in Table B.

You cannot access these values as numbers in SOUL; however, retrieval through U EDIT specifications works normally.

Specifying V and M formats

The V format processes field occurrences in the same manner as the M format.

If the field name corresponding to the format in the field name list is specified as fieldname(n), n indicates the position of the first occurrence to be changed. If n is represented by an asterisk (*), all of the values are added as new field occurrences.

Note that the VL format is equivalent to the M format, except that the data values are preceded by a four-byte count rather than by a one-byte count.

Updating a FLOAT field using A, J, L, M, or U formats

When the updated field is defined as FLOAT, Model 204 converts exponential format numbers to floating point using A, J, L, M, or U formats.

When the field is not FLOAT, Model 204 leaves exponential format numbers in their original character form.

Using EDIT format codes for an updating call

Table 7-3 describes the edit format codes that can be specified in the edit format with the EDIT option in an IFUPDT, IFSTOR, or IFPUT (updating) call.

Table 7-3. EDIT format codes used with an updating call

Code	Description
A(n) and J(n)	The field value in the data area is n characters long. It is stored with leading and trailing blanks removed. If the final length is 0 after removal of the blanks, the existing field is deleted or is not stored for a new field.

Table 7-3. EDIT format codes used with an updating call (Continued)

Code	Description
B(precision,scale)	The value is in two's-complement binary form and is converted to the appropriate numeric format before being stored. Precision specifies the number of bits in the binary value, exclusive of the sign bit. Only two precisions are supported: 15 (a halfword) and 31 (a fullword). If both precision and scale are omitted, the precision defaults to 15. Scale specifies the number of digits to the right of an implied binary point. The scale must not exceed the precision. If scale is omitted, it defaults to 0, indicating an integer value. If the scale is more than 12, only the high-order 12 fractional bits are converted. Leading integer zeros and trailing fractional zeros are removed from the converted string. An all-zero string is stored as a single zero.
F(n)	The value is in floating-point form. The F specification must be used with FLOAT fields to make floating-point values accessible as meaningful numbers to Model 204. • If n is 16, the value is extended precision (occupies 16 bytes and holds up to 31 significant digits). • If n is 8, the value is long precision (occupies 8 bytes and holds up to 15 significant digits). • If n is 4, the value is short precision (occupies 4 bytes and holds up to 6 significant digits). Values to be stored in FLOAT fields whose lengths differ from the length defined for the FLOAT field are truncated or rounded to the defined length without a cancellation of the call.
COL(n) or COLUMN(n) or POS(n)	Skip to character position n of the data area.
E(total length,significant digits,fractional digits)	Input is in external decimal form and is converted to the appropriate numeric format before being stored. The specification includes the number of bits in the binary value, exclusive of the sign bit. Only two precisions are supported: 15 (a halfword) and 31 (a fullword). If both precision and scale are omitted, the precision defaults to 15. Scale specifies the number of digits to the right of an implied binary point. The scale must not exceed the precision. If scale is omitted, it defaults to 0, indicating an integer value. If the scale is more than 12, only the high-order 12 fractional bits are converted. Leading integer zeros and trailing fractional zeros are removed from the converted string. An all-zero string is stored as a single zero.
G	Allows you to: • Select the data type at execution time • Bypass an occurrence of a specific field at execution time • Change or delete specific field=value pairs The G format requires that the data is preceded by a descriptor byte. The G format is described in detail beginning on page 326.
L	The first byte of the data is a hexadecimal number that represents the length of the data bytes that follow. The value is stored without the length byte. Leading binary zeros are eliminated; leading and trailing blanks are not eliminated. If the final length is 0, the field is deleted; if the field is new, it is not stored. For example, X'03C1C2C3' is equivalent to 'ABC'; X'04C1C2C340' includes the trailing blank in the stored value.
L(n)	The value is n characters long. Leading binary zeros (X'00') are eliminated; leading and trailing blanks are not eliminated. If the final length is 0, the existing field is deleted. If the field is new, it is not stored.
M(n)	The first data byte contains the number of value occurrences. The values that follow are each n characters long. Leading binary zeros are eliminated.

Table 7-3. EDIT format codes used with an updating call (Continued)

Code	Description
	The value is in exponential format and is converted to the appropriate numeric format before being stored. The significant digits and fractional digits specifications are ignored by IFPUT. They are listed so that an application's IFPUT and IFGET edit formats can be shared.
F(n)	The value is in floating-point form. The F specification must be used with FLOAT fields to make floating-point values accessible as meaningful numbers to Model 204. • If n is 16, the value is extended precision (occupies 16 bytes and holds up to 31 significant digits).
	• If n is 8, the value is long precision (occupies 8 bytes and holds up to 15 significant digits). • If n is 4, the value is short precision (occupies 4 bytes and holds up to 6 significant digits). Values to be stored in FLOAT fields whose lengths differ from the length defined for
	the FLOAT field are truncated or rounded to the defined length without a cancellation of the call.
G	Allows you to: • Select the data type at execution time • Bypass an occurrence of a specific field at execution time • Change or delete specific field=value pairs The G format requires that the data is preceded by a descriptor byte. The G format is described in detail beginning on page 326.
L	The first byte of the data is a hexadecimal number that represents the length of the data bytes that follow. The value is stored without the length byte. Leading binary zeros are eliminated; leading and trailing blanks are not eliminated. If the final length is 0, the field is deleted; if the field is new, it is not stored. For example, X'03C1C2C3' is equivalent to 'ABC'; X'04C1C2C340' includes the trailing blank in the stored value.
L(n)	The value is n characters long. Leading binary zeros (X'00') are eliminated; leading and trailing blanks are not eliminated. If the final length is 0, the existing field is deleted. If the field is new, it is not stored.
M(n)	The first data byte contains the number of value occurrences. The values that follow are each n characters long. Leading binary zeros are eliminated.

Table 7-3. EDIT format codes used with an updating call (Continued)

Code	Description
V	Collect all occurrences of a field into a single data area. The area begins with a four-byte count of the number of data values in the area. The format of the values is determined by the second edit format, which must be specified immediately after the V. An example and a summary of restrictions is included below.
X	Same as X(1). See X(n).
X(n)	Skip n characters in the data area.
Z(precision,scale)	The value is in zoned-decimal form. Precision, scale, and conversion rules are identical to the P format.

8

Completion and ABEND Codes

Overview

This chapter describes the completion codes that may be encountered when using the Model 204 HLI facility. There are two broad categories of completion codes:

- Return codes for HLI calls
- Job run ABEND codes

This chapter presents the codes in the following tables:

- Table 8-1 on page 334 lists completion codes that are returned by Model 204 to the host language application program for operations which completed normally, or with a warning. RETCODE is 0, 1, 2, and 3.
- Table 8-2 on page 335 lists completion codes which usually indicate that a call was unsuccessful. RETCODE is greater than or equal to 4.
- Table 8-3 on page 340 lists completion codes that are generated for severe errors that cause the HLI job run to abend.

For more information

For more information about individual completion return codes for conditions that are unique to a specific function call, refer to the usage notes for the particular call in Chapter 6.

Completion return codes 0–3

Table 8-1 describes the completion codes that are returned by Model 204 to the host language application program for operations which completed normally, or with a warning. RETCODE is 0, 1, 2, and 3.

Table 8-1. Completion return codes 0–3

Code	Call(s)	Description
0	IFATTN	IFDIAL connection completed.
	IFCHKPT	Checkpointing status depends on which function code is specified in the call: • Function code is 0: Checkpoint not currently in progress, parameter 3 unaltered. • Function code is 1: Checkpoint successfully initiated. • Function code is 3: Checkpoint successfully completed.
	IFDIAL IFDIALN	IFDIAL connection established.
	IFHNGUP	IFDIAL connection dropped.
	All except IFCHKPT	Function completed successfully.
1	IFCHKPT	Checkpointing status depends on which function code is specified in the call: • Function code is 0: Checkpoint in progress, parameter 3 set to its ID. • Function code is 2: Checkpoint completed successfully, parameter 3 set to its ID.
	IFGET IFFTCH	Nonnumeric value could not be converted, or integer portion of the value is too large for output area. Errors may occur for B, P, F, and Z edit formats.
	IFMORE	Nonnumeric value could not be converted, or integer portion of the value is too large for output area. Errors may occur for B, P, F, and Z edit formats.
	IFREAD	Call IFWRITE next to provide Model 204 with input.
	IFSTRT (IFAM1)	Illegal call. Thread already exists.
	IFWRITE	Call IFWRITE next to provide Model 204 with input.

Table 8-1. Completion return codes 0–3 (Continued)

Code	Call(s)	Description
2	IFCHKPT	Checkpoint request timed out. Checkpoint not taken.
	IFFLS	The file indicator was used to identify the file of the current record and no current record was found.
	IFFTCH	Indicates no more records to fetch in the current set.
	IFREAD	Call IFREAD next to get more output from Model 204.
	IFRPRM and IFSPRM	TBO has been enabled or disabled for a file such that the group of files in the job violates the restriction that TBO files may not open for update while any non-TBO file is open on a read-only thread. The parameter reset was successful, but the M204.0498 error was issued and the file is closed.
	IFWRITE	Call IFREAD next to get more output from Model 204.
	Any other call	Indicates no records in the current set.
3	IFCHKPT	Caller is not allowed to wait for checkpoints. Call ignored.
	All except IFCHKPT	Return code number reserved for future use, or control of necessary resource could not be obtained and call may be reexecuted.

Completion return codes 4 and greater

Table 8-2 describes the completion codes which usually, but not always, indicate that a call was unsuccessful. RETCODE is greater than or equal to 4. Note that the error message is written to the Model 204 journal and is available to the application program using the IFGERR call.

Table 8-2. Completion return codes 4 and greater

Code	Call(s)	Description
4	All calls	Error message has been produced. For more information about conditions that are unique to a specific function call, refer to the Notes for the particular call in Chapter 6.
5	IFCHKPT	Checkpoint not active during recovery. Call ignored.
	Compiled IFAM calls	The name used by an E (execute) call has not been defined.

Table 8-2. Completion return codes 4 and greater (Continued)

Code	Call(s)	Description
6	Compiled IFAM calls	The name used by an E (execute) call or standard compile and execute call has been defined by an incompatible function. IFGET, IFMORE, and IFPUT specs can be shared, but cannot be referred to by IFFIND calls. Similarly, IFFIND compilations cannot be referred to by IFGET-type calls. IFFIND and IFFNDX cannot share specs.
7	Compiled IFAM calls	Not enough space exists to compile the current call. Error message indicates which server table is full. Use IFFLUSH to recover from this condition. Many Host Language Interface calls not directly affected by the Compiled IFAM feature use space in QTBL and VTBL. These functions return a code of 4 if either table fills.
10	IFBREC IFSTOR IFPUT IFUPDT	For an updating call where the FILEMODL file option is set to NUMERIC VALIDATION, Model 204 encountered invalid values for BINARY and FLOAT numeric field types. See the Rocket Model 204 documentation wiki for more information about FILEMODL: http://m204wiki.rocketsoftware.com/index.php/Field_attributes#File_model_feature See the Rocket Model 204 documentation wiki for information about binary and float values: http://m204wiki.rocketsoftware.com/index.php/Data_maintenance#Storing_data_in_fields
	All calls	The function has been cancelled. The transaction is automatically backed out and control is returned to the application program.
12	IFREAD and IFWRITE	IFREAD called when IFWRITE was expected, or IFWRITE called when IFREAD was expected.
	IFREAD	No output ready from Model 204; call IFWRITE to provide more input. The answer area is not altered.
15	IFEFCC and IFERLC	An updating function received an error when there was no conflict to search for.

Table 8-2. Completion return codes 4 and greater (Continued)

Code	Call(s)	Description
40	IFBOUT	An IFBOUT was issued from a read-only thread. IFBOUT is only valid on update threads.
	All calls	Function requires update privileges, but none were granted. Issue IFSTRT with thread update privileges and give file or group password with update privileges.
50	All calls (IFAM2 and IFAM4)	An updating function was issued on a thread which has been quiesced for checkpointing by a previous call to IFCHKPT.
60	All calls	No current file or group. Function cannot complete.
61	All calls	Function attempted against file or group with broken FISTAT setting. Function cannot complete.
80	All calls in IFAM1	No current thread. • For IFSTRT support, call IFSTRT. • For IFDIAL support, call IFSETUP.
90	IFDIAL IFDIALN	An IFDIAL connection already exists for this application program. The attempt to establish more than one simultaneous connection for this application is ignored.
	IFSTRT in IFAM1	Illegal call to IFSTRT. A thread already exists.
	IFSTRT in IFAM2 and IFAM4	Model 204 is not yet available for user processing. Recovery may still be in progress. Call IFSTRT again.
95	IFDTHRD and IFSTHRD	Nonexistent new thread specified. Call ignored.
96	IFDTHRD	New thread is already current thread and thus is not detached.

Table 8-2. Completion return codes 4 and greater (Continued)

Code	Call(s)	Description
100	in IFAM1	LOGIN failed.
	in IFAM2 and IFAM4	No current thread exists. Call IFSTRT.
	IFATTN	The connection was lost.
	IFHNGUP	The connection was lost prior to call.
	IFREAD	No current Model 204 connection exists or the connection is lost.
	IFSTRT in IFAM2 and IFAM4	LOGIN failed.
	IFWRITE	No current Model 204 connection exists or the connection is lost.
101	IFDIAL in IFAM1	Invalid BATCH204 module loaded. Check STEPLIB for correct release.
	IFSTRT in IFAM1	Failure to dynamically load the IFAM1 load module, because the entry point was specified incorrectly when relinking.
200	IFBREC IFPUT IFSTOR IFUPDT	A uniqueness violation has occurred (field level constraint). A UNIQUE violation corresponds to errors handled by ON FCC ON units in SOUL. Refer to the Rocket Model 204 documentation wiki for information about ON units: http://m204wiki.rocketsoftware.com/index.php/Subroutines#On_units_2
202	IFPUT IFSTOR IFUPDT	An AT-MOST-ONE violation occurred (field level constraint). An AT-MOST-ONE violation corresponds to errors handled by ON FCC ON units in SOUL. Refer to the Rocket Model 204 documentation wiki for information about ON units: http://m204wiki.rocketsoftware.com/index.php/Subroutines#On_units_2
260	IFOPEN	A file or group could not be opened.
300	All calls	Thread restarted in Host Language Interface Model 204 service program. Call lost.
325	in IFAM2	IQB not large enough. Adjust the Model 204 parameters LIBUFF, LOBUFF, and IFAMBS as necessary.
350	in IFAM2	Invalid string length. Adjust the Model 204 parameters LIBUFF and LOBUFF as necessary.

Table 8-2. Completion return codes 4 and greater (Continued)

Code	Call(s)	Description
400	IFSTRT and IFSETUP in IFAM2 and IFAM4	Invalid language code.
	Other calls in IFAM2 and IFAM4	Invalid parameter list.
500	IFCALL	Invalid function number passed to IFCALL.
	in IFAM1 using IFDIAL	Invalid function called.
800	in IFAM2 and IFAM4	No Host Language Interface Model 204 threads currently available. Try again later or increase number of defined threads.
	IFDIAL	All Model 204 IFDIAL connections are busy. Check to see that the proper number of IODEV=29 initialization statements were included for the Host Language Interface Model 204 service program.
1000	IFFNSH	Normal return code from IFFNSH. (See 1nnn.)
	in IFAM2 and IFAM4	The application program disconnected from Model 204 normally.
	All other calls	Model 204 is no longer available.
1001	in IFAM2 and IFAM4	Host Languages Interface Model 204 service program is not up, the Host Language interface is halted or drained, or no host language threads were defined in Model 204 This may indicate that the IFAM4 load module was link-edited without the REUS option.
	IFDIAL	Host Language Interface Model 204 is not up, or no IODEV=39 statements were included in initialization, or CRAM channel name does not exist. Connection is impossible.
1002	in IFAM2	Insufficient SQA or CSA space to use CRAM connection.
1003	in IFAM2 and IFAM4	Insufficient memory in the IFAM application to allocate the necessary control blocks.
	IFDIAL IFDIALN	Not enough memory for CRAM open.
1004	in IFAM2	An initial IFCSA call was not made.

Table 8-2. Completion return codes 4 and greater (Continued)

Code	Call(s)	Description
1nnn	IFFNSH in IFAM1	IFFNSH returns 1000 and the highest Model 204 journal error message return code encountered during the run.

Job run ABEND codes

Table 8-3 describes the user ABEND completion codes that are generated when Model 204 detects *severe errors* and abends the HLI application program job run.

Table 8-3. Job run ABEND codes

Code	Job type	Description
100	IFAM1	CCASNAP DD statement is missing.
250	IFAM2 and IFAM4	Invalid parameter list. Completion code could not be set.
1000	IFAM2	Serious CRAM error. Save all output from application and Host Language Interface Model 204 service program for Technical Support personnel.

For more information about how IFAM4 sets the job step return code, see “IFAM4 jobs: Job errors and ABENDs” on page 48.

A

IFAM1 Job Program Samples

Overview

This appendix provides examples of IFAM1 jobs, complete with program code. The sample programs illustrate the use of Host Language Interface functions in an IFAM1 processing environment.

For more information

See Appendix B for additional examples of HLI applications written in COBOL, and job setups that may be run in IFAM2 and IFAM4. Appendix B includes an example of an application that uses a multiple cursor IFSTRT thread.

See Chapter 4 for guidelines on using different programming languages when coding HLI programs. See the *Rocket Model 204 Host Language Interface Programming Guide* for coding examples related to particular aspects of HLI processing.

COBOL example

This section provides a sample program written in COBOL to run in the IFAM1 environment under VSE or CMS using the Model 204 Host Language Interface.

The sample COBOL program that is shown in Figure 1-1 can be run as shown with the application code embedded in the VSE job stream.

Or, the program can be extracted from the VSE JCL and compiled and linked as program IFAM1PG and then run in CMS using the EXECs shown in Figure 1-2 and Figure 1-3.

Using a vehicles file

The IFAM1 COBOL application program accesses a file that contains vehicle data, that is, a Model 204 data file named VEHICLES, and uses the information to produce a report of high risk vehicles.

IFAM1 COBOL example (VSE)

Figure 1-1, starting below and continuing on the following pages, shows an HLI application program that is written for IFAM1 in COBOL which, with the JCL that is shown in the job stream, runs under a VSE operating system.

This application uses a single cursor IFSTRT thread.

Figure 1-1. Sample COBOL program (VSE)

```
* ************************************************************
*                                                            *
* This example COBOL job compiles and catalogs an IFAM1      *
* program into a user's private library and                   *
* is based on the following assumptions:                     *
*                                                            *
* (1) All Model 204 distribution macros, object modules, etc. *
*     are in the M204 library as distributed by Rocket;      *
*                                                            *
* (2) All IBM distributed libraries (such as standard macros, *
*     COBOL compiler, link-time modules, etc.) are defined in *
*     your permanent library search sequence;                *
*                                                            *
* (3) You have a private library for in-house developed      *
*     application programs (systems).                        *
*                                                            *
* ************************************************************
* $$ JOB ...

.
.
.
// JOB CATALOG IFAM1 PROGRAM (WRITTEN IN COBOL)
// DLBL M204LIB, 'M204.PROD.LIBRARY'
// EXTENT SYSnnn,......balance of the EXTENT statement
// DLBL USERLIB,'user.applic.system.library'
// EXTENT SYSnnn,....balance of the EXTENT statement
// LIBDEF *,SEARCH=(M204LIB.V220,USERLIB.sublib)
// LIBDEF PHASE,CATALOG=(USERLIB.sublib)
// OPTION CATAL
   PHASE IFiPGM,* REPLACE=YES
// EXEC FCOBOL,SIZE=(160K)
************************************************************
*                                                          *
*  THIS IS A SAMPLE IFAM1 COBOL PROGRAM.                   *
*                                                          *
*  THIS COBOL PROGRAM USES IFAM1 CALLS.                    *
```

```
*  IT PRODUCES A REPORT OF HIGH RISK VEHICLES. IT ALSO        *
*  CHANGES THE SURCHARGE% ON CERTAIN HIGH RISK                *
*  VEHICLES TO 15%.                                           *
*                                                             *
*                                                             *
***************************************************************


IDENTIFICATION DIVISION.
PROGRAM-ID.
   IFAM1EX1.
AUTHOR.
   JANE DOE.
DATE-WRITTEN.
   MAY 15, 1990.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT REPORT-FILE ASSIGN TO UT-S-REPORT.
DATA DIVISION.
FILE SECTION.
FD REPORT-FILE
   LABEL RECORDS ARE OMITTED
   BLOCK CONTAINS 0 RECORDS
   DATA RECORD IS REPORT-RECORD.
01 REPORT-RECORD             PIC X(133).
WORKING-STORAGE SECTION.
01 CRITICAL-ERROR-SW         PIC X(3)        VALUE "NO ".
   88 NO-CRITICAL-ERROR                      VALUE "NO ".
   88 CRITICAL-ERROR                         VALUE "YES".
01 DONE-PROCESS-SW           PIC X(3)        VALUE "NO ".
   88 DONE-PROCESS                           VALUE "YES".
01 ERROR-FUNCTION            PIC X(8).
01 DISPLAY-STATUS-IND        PIC 9(5)        VALUE ZERO.
01 WS-OUTPUT-REPORT-LINE.
   05 WS-CCTL-CHAR           PIC X.
   05 WS-132-CHAR-LINE       PIC X(132)      VALUE SPACES.
01 WS-LINE-COUNT             PIC 99          VALUE ZERO.
01 NEW-SURCHARGE             PIC X(2)        VALUE "15".


01 M204-INTEGER-CALL-ARGS   COMP SYNC.
   05 STATUS-IND             PIC 9(5).
   05 LANGUAGE-IND           PIC 9(5)        VALUE 2.
   05 UPDATE-IND             PIC 9(5)        VALUE 0.
   05 FIND-COUNT             PIC 9(5).


01 M204-STRING-CALL-ARGS.
   05 EXEC-PARMS.
        10 FILLER            PIC X(7)        VALUE "SYSOPT=".
        10 INPUT-SYSOPT      PIC X(3).
        10 FILLER            PIC X           VALUE ";".
   05 USER0-PARMS.
```

```
        10 FILLER            PIC X(7)        VALUE "MAXBUF=".
        10 INPUT-MAXBUF      PIC X(3).
        10 FILLER            PIC X(8)        VALUE ",MINBUF=".
        10 INPUT-MINBUF      PIC X(3).
        10 FILLER            PIC X(8)        VALUE ",SPCORE=".
        10 INPUT-SPCORE      PIC X(5).
        10 FILLER            PIC X(8)        VALUE ",LAUDIT=".
        10 INPUT-LAUDIT      PIC X(1).
        10 FILLER            PIC X           VALUE ";".
     05 LOGIN                PIC X(20)       VALUE
       "SUPERKLUGE;PIGFLOUR;".
     05 VEHICLE-FILE         PIC X(8)        VALUE "VEHICLES;".
     05 FIND-CRITERIA        PIC X(41)       VALUE
        "VEHICLE USE CLASS IS GREATER THAN 79;END;".


     05 GET-EDIT-SPEC.
        10 FILLER            PIC X(45)      VALUE
        "EDIT(VEHICLE USE CLASS,VIN,OWNER POLICY,MAKE,".
        10 FILLER            PIC X(43)      VALUE
        "MODEL,BODY,YEAR,SURCHARGE%)(X(7),J(2),X(8),".
        10 FILLER            PIC X(43)      VALUE
        "A(12),X(6),A(6),X(6),A(15),X(3),A(15),X(3),".
        10 FILLER            PIC X(26)      VALUE
        "A(4),X(4),A(2),X(8),J(2));".
     05 M204-ERR-MESSAGE     PIC X(80).
     05 PUT-EDIT-SPEC        PIC X(23)      VALUE
        "EDIT(SURCHARGE%)(Z(2));".
     05 PUTNAME              PIC X(8)       VALUE "PUTNAME;".
     05 GETNAME              PIC X(8)       VALUE "GETNAME;".


01 REPORT-OUTPUT-DETAIL.
     05 FILLER              PIC X(7).
     05 USE-CLASS           PIC X(2).
     05 FILLER              PIC X(8).
     05 VIN                 PIC X(12).
     05 FILLER              PIC X(6).
     05 OWNER-POLICY        PIC X(6).
     05 FILLER              PIC X(6).
     05 MAKE                PIC X(15).
     05 FILLER              PIC X(3).
     05 MODEL               PIC X(15).
     05 FILLER              PIC X(3).
     05 BODY                PIC X(4).
     05 FILLER              PIC X(4).
     05 YEAR                PIC X(2).
     05 FILLER              PIC X(8).
     05 SURCHARGE           PIC X(2).
     05 FILLER              PIC X(28)      VALUE SPACES.
01 REPORT-HEADING-AREA.
     05 FILLER              PIC X(40)      VALUE SPACES.
     05 FILLER              PIC X(46)      VALUE
        "HIGH RISK VEHICLES - INCLUDING NEW SURCHARGE %".
```

```
   05 FILLER                PIC X(46)      VALUE SPACES.
01 REPORT-DETAIL-HEADING.
   05 FILLER                PIC X(2)       VALUE SPACES.
   05 FILLER                PIC X(12)      VALUE
      "USE CLASS    ".
   05 FILLER                PIC X(16)      VALUE
      "      V  I  N        ".
   05 FILLER                PIC X(15)      VALUE
      "OWNER POLICY  ".
   05 FILLER                PIC X(18)      VALUE
      "        MAKE                      ".
   05 FILLER                PIC X(20)      VALUE
      "        MODEL                      ".
   05 FILLER                PIC X(07)      VALUE "BODY        ".
   05 FILLER                PIC X(07)      VALUE "YEAR        ".
   05 FILLER                PIC X(10)      VALUE
      "SURCHARGE%".
   05 FILLER                PIC X(25)      VALUE SPACES.
01 WS-PARAMETER-INPUT.
   05 PARAM-SYSOPT          PIC X(3).
   05  FILLER               PIC X.
   05 PARAM-MAXBUF          PIC X(3).
   05 FILLER                PIC X.
   05 PARAM-MINBUF          PIC X(3).
   05 FILLER                PIC X.
   05 PARAM-SPCORE          PIC X(5).
   05 FILLER                PIC X.
   05 PARAM-LAUDIT          PIC X(1).


PROCEDURE DIVISION.
INITIALIZATION.
   OPEN OUTPUT REPORT-FILE.
   ACCEPT WS-PARAMETER-INPUT FROM SYSIPT.
*************************************************************
* NOTE: PARAMETER INPUT MUST BE ENTERED IN THE FOLLOWING      *
*  FORMAT IN THE INPUT-FILE: 999,999,999,99999,9              *
*  WHICH CORRESPONDS TO THE VALUES OF:                        *
*  SYSOPT,MAXBUF, MINBUF,SPCORE,LAUDIT                       *
* ************************************************************


IF PARAM-SYSOPT NOT NUMERIC OR
   PARAM-SYSOPT > 186
   DISPLAY "INVALID SYSOPT PARAMETER, DEFAULT OF 128 USED"
   MOVE "128" TO INPUT-SYSOPT
ELSE
   MOVE PARAM-SYSOPT TO INPUT-SYSOPT.
IF PARAM-MINBUF NOT NUMERIC OR
   PARAM-MINBUF <  003
   DISPLAY "INVALID MINBUF PARAMETER, DEFAULT OF 3 USED"
   MOVE "003" TO INPUT-MINBUF
ELSE
   MOVE PARAM-MINBUF TO INPUT-MINBUF.
```

```
IF PARAM-MAXBUF NOT NUMERIC OR
   PARAM-MAXBUF  <  003
   DISPLAY "INVALID MAXBUF PARAMETER, DEFAULT OF 100 USED"
   MOVE "100" TO INPUT-MAXBUF
ELSE
   MOVE PARAM-MAXBUF TO INPUT-MAXBUF.
IF INPUT-MINBUF > INPUT-MAXBUF
   DISPLAY "MINBUF REQUESTED> MAXBUF, DEFAULT TO MAX=MIN".
   MOVE INPUT-MINBUF TO INPUT-MAXBUF.
IF PARAM-SPCORE NOT NUMERIC
   DISPLAY "INVALID SPCORE PARAMETER, DEFAULT TO 8192"
   MOVE "08192" TO INPUT-SPCORE
ELSE
   MOVE PARAM-SPCORE TO INPUT-SPCORE.
IF PARAM-LAUDIT > 7
   DISPLAY "INVALID LAUDIT PARAMETER, DEFAULT TO 7"
   MOVE "7" TO INPUT-LAUDIT
ELSE
   MOVE PARAM-LAUDIT TO INPUT-LAUDIT.
PERFORM NEW-PAGE.
CALL "IFSTRT" USING STATUS-IND, LANGUAGE-IND,
       EXEC-PARMS, USER0-PARMS.
IF STATUS-IND IS NOT EQUAL ZERO
   MOVE "IFSTRT  " TO ERROR-FUNCTION
   PERFORM ERROR-ROUTINE
ELSE
   CALL "IFLOG" USING STATUS-IND, LOGIN
   IF STATUS-IND IS NOT EQUAL ZERO
      MOVE "IFLOG     " TO ERROR-FUNCTION
      PERFORM ERROR-ROUTINE


ELSE
   CALL "IFOPEN" USING STATUS-IND, VEHICLE-FILE
   IF STATUS-IND IS NOT EQUAL ZERO AND
      STATUS-IND IS NOT EQUAL 16 AND
      STATUS-IND IS NOT EQUAL 32
      MOVE "IFOPEN   " TO ERROR-FUNCTION
      PERFORM ERROR-ROUTINE.
FIND-RECORDS.
   IF CRITICAL-ERROR GO TO FIND-RECORDS-EXIT.
* ************************************************************
*  DO NOT ATTEMPT TO FIND THE RECORDS IN THE MODEL 204 FILE   *
*  IF YOU PREVIOUSLY ENCOUNTERED AN ERROR WITH                *
*  IFSTRT, IFLOG,  OR IFOPEN.                                 *
* ************************************************************


   CALL "IFFIND" USING STATUS-IND, FIND-CRITERIA.
   IF STATUS-IND NOT EQUAL ZERO
      MOVE "IFFIND    " TO ERROR-FUNCTION
      PERFORM ERROR-ROUTINE
   ELSE
      CALL "IFCOUNT" USING STATUS-IND, FIND-COUNT
```

```
      IF STATUS-IND NOT EQUAL ZERO
      MOVE "IFCOUNT    " TO ERROR-FUNCTION
      PERFORM ERROR-ROUTINE
      ELSE
         IF FIND-COUNT = ZERO
            MOVE "NO RECORDS FOUND" TO WS-132-CHAR-LINE
            WRITE REPORT-RECORD FROM WS-OUTPUT-REPORT-LINE
            MOVE "YES" TO DONE-PROCESS-SW.
*************************************************************
*                                                           *
* DO NOT ATTEMPT TO GET MODEL 204 RECORDS IF YOU            *
*  ENCOUNTERED AN ERROR WITH IFFIND/IFCOUNT OR IF THERE     *
*  WERE NO RECORDS FOUND (THAT IS, FIND-COUNT IS ZERO)      *
*                                                           *
*  THE DONE-PROCESS SWITCH WILL INDICATE WHEN ALL THE       *
*  RECORDS IN THE "FOUND SET" HAVE BEEN PROCESSED.          *
*  IF NO RECORDS WERE FOUND THEN THE DONE PROCESS           *
*  SWITCH IS TO "YES" IMMEDIATELY AFTER IFCOUNT.            *
*                                                           *
*************************************************************
  PERFORM GET-AND-PROCESS-RECORDS
    UNTIL DONE-PROCESS OR CRITICAL-ERROR.
FIND-RECORDS-EXIT. EXIT.
TERMINATION.
  CLOSE REPORT-FILE.
  CALL "IFFNSH" USING STATUS-IND.
  IF STATUS-IND NOT EQUAL 1000
    MOVE "IFFNSH " TO ERROR-FUNCTION
    PERFORM ERROR-ROUTINE.
  STOP RUN.


GET-AND-PROCESS-RECORDS.
  CALL "IFGET" USING STATUS-IND, REPORT-OUTPUT-DETAIL,
    GET-EDIT-SPEC, GETNAME.
  IF STATUS-IND = 2
    MOVE "YES" TO DONE-PROCESS-SW
  ELSE
    IF STATUS-IND NOT EQUAL ZERO
      MOVE "IFGET " TO ERROR-FUNCTION
      PERFORM ERROR-ROUTINE
    ELSE
      PERFORM REPORT-AND-UPDATE.
REPORT-AND-UPDATE.
  IF USE-CLASS > 85 OR USE-CLASS = 85
    MOVE NEW-SURCHARGE TO SURCHARGE
    CALL "IFPUT" USING STATUS-IND, SURCHARGE,
              PUT-EDIT-SPEC, PUTNAME
    IF STATUS-IND NOT EQUAL ZERO
      MOVE "IFPUT " TO ERROR-FUNCTION
      PERFORM ERROR-ROUTINE.
  IF WS-LINE-COUNT > 50
    PERFORM NEW-PAGE.
  MOVE REPORT-OUTPUT-DETAIL TO WS-132-CHAR-LINE.
```

```
  WRITE REPORT-RECORD FROM WS-OUTPUT-REPORT-LINE.
  ADD 1 TO WS-LINE-COUNT.
ERROR-ROUTINE.
  MOVE "YES" TO CRITICAL-ERROR-SW.
************************************************************
*     THIS CRITICAL ERROR SWITCH IS SET IF THERE IS A BAD     *
*     IFAM CALL TO MODEL 204.                                 *
 ************************************************************


  MOVE STATUS-IND TO DISPLAY-STATUS-IND.
  DISPLAY "CRITICAL ERROR ENCOUNTERED WITH FUNCTION: "
    ERROR-FUNCTION ", WITH A RETURN CODE OF: "
    DISPLAY-STATUS-IND.
  CALL "IFGERR" USING STATUS-IND M204-ERR-MESSAGE.
  DISPLAY "M204 ERROR MESSAGE = " M204-ERR-MESSAGE.
NEW-PAGE.
* MOVE "1" TO WS-CCTL-CHAR.
  MOVE REPORT-HEADING-AREA TO WS-132-CHAR-LINE.
  WRITE REPORT-RECORD FROM WS-OUTPUT-REPORT-LINE.
* MOVE "0" TO WS-CCTL-CHAR.
  MOVE REPORT-DETAIL-HEADING TO WS-132-CHAR-LINE.
  WRITE REPORT-RECORD FROM WS-OUTPUT-REPORT-LINE.
* MOVE "  " TO WS-CCTL-CHAR.
  MOVE SPACES TO WS-132-CHAR-LINE.
  WRITE REPORT-RECORD FROM WS-OUTPUT-REPORT-LINE.
  MOVE ZEROS TO WS-LINE-COUNT.


************************************************************
* JOB STEP #2:  LINK-EDIT THE PROGRAM                      *
************************************************************
/*
  INCLUDE IFIF1DOS
ENTRY IFAMTEST
// EXEC LNKEDT
/&
* ************************************************************
*  JOB STEP #3:                                             *
*  EXECUTE IFAM1 IF1PGM PROGRAM UNDER VSE                   *
*  RESULTS PRINTED IN REPORT ON SYSLST                      *
* ************************************************************


// JOB IF1RUN
// DLBL M204LIB, 'M204.PROD.LIBRARY'
// EXTENT SYSnnn,......balance of the EXTENT statement
// DLBL USERLIB,'user.applic.system.library'
// EXTENT SYSnnn,....balance of the EXTENT statement
*
// DLBL CCAJRNL,'SQADOS.M204SYS.CCAJRNL',0
// EXTENT SYS021,SYSWK1,,,72470,2000
// DLBL CCASTAT,'SQADOS.M204SYS.CCASTAT',0
// EXTENT SYS024,SYSWK4,,,1059,100
```

```
// DLBL CCATEMP,'SQADOS.M204SYS.CCATEMP',,DA
// EXTENT SYS021,SYSWK1,,,51330,2000
// DLBL VEHICLE,'PSGRAY.M204DB.VEHICLE',,DA
// EXTENT SYS023,SYSWK3,,,68101,2600
// ASSGN SYS007,SYSLST
// ASSGN SYS008,05E
// EXEC IF1PGM,SIZE=(AUTO,60K)
144,010,010,10000,0
/*
/&
* $$ EOJ
```

IFAM1 COBOL example (CMS)

Figures 1-2 and 1-3 show sample EXECs that may be used to execute an HLI application program in the IFAM1 environment under CMS.

Note: These EXECs could be used with the COBOL program on page 342 assuming that the program is compiled and linked as IFAM1PG.

Figure 1-2 shows an EXEC that generates an IFAM1 application module, IFAM1PG, running under CMS.

Figure 1-2. Sample EXEC to run an IFAM1 program (CMS)

```
&CONTROL ALL
*
* SAMPLE EXEC FOR GENERATING AN IFAM1 APPLICATION
*
* PROGRAM NAME = IFAM1PG
*
COBOL IFAM1PG
*
GLOBAL TXTLIB M204IFM1 COBOLVS COBLIBVS
LOAD IFAM1PG IFCM1 ( RESET IFAM1PG )
GENMOD IFAM1PG
*
&TYPE
&TYPE IFAM1PG MODULE A HAS BEEN GENERATED FOR YOUR USE
&TYPE
&EXIT &RETCODE
```

Figure 1-3 below shows an EXEC that defines the files for the IFAM1PG program running under CMS which is shown in Figure 1-2 on the preceding page.

Figure 1-3. Example of FILES EXEC for IFAM1 program (CMS)

```
&TRACE ALL
*
* EXEC TO DEFINE FILES FOR IFAM1 JOB
*
  &ERROR &EXIT &RETCODE
```

```
  FILEDEF * CLEAR

* ACCESS XXX 121 DISK
  GETFMADR 250
  &READ VARS & &XXX121M &XXX121D
  EXECIO 0 CP ( STRING LINK XXX 121 &XXX121D MW
  ACCESS &XXX121D &XXX121M

* GET TEMP DISK SPACE FOR CCATEMP FILE
  GETFMADR 250
  &READ VARS & &TEMPM &TEMPD
  EXECIO 0 CP (STRING DEFINE T3380 &TEMPD CYL 3
  &IF &RETCODE NE 92 &IF &RETCODE NE 0 &EXIT &RETCODE
  &STACK LIFO YES
  M204UTIL INIT &TEMPD TEM204
  ACCESS &TEMPD &TEMPM
  M204UTIL CREATE M204 CMS CCATEMP &TEMPM ( PRIMARY 40 TRK

* DEFINE FILES USED BY IFAM1 APPLICATION PROGRAM & MODEL 204
  FILEDEF  CCAAUDIT   DISK IFM1   CCAAUDIT A
  FILEDEF  CCAPRINT   DISK IFM1   CCAPRINT A
  FILEDEF  CCASTAT &XXX121M       DSN PSSOSMNT PROD CCASTAT M204
  FILEDEF  CCASNAP    DISK IFM1   CCASNAP A
  FILEDEF  CCATEMP &TEMPM         DSN M204 CMS CCATEMP
  FILEDEF  VEHICLES &XXX121M      DSN PSSOSMNT TEST VEHICLES M204
  FILEDEF  REPORT     DISK IFM1 REPORT A
  FILEDEF  SYSIN      DISK IFM1 SYSIN *
  FILEDEF  SYSOUT     DISK IFM1 SYSOUT A

* STACK IFAM1 APPLICATION PROGRAM NAME

  &STACK IFAM1PG

&EXIT &RETCODE
```

PL/I example

This section provides a sample program written in PL/I to run in the IFAM1 environment under z/OS using the Model 204 Host Language Interface. The sample PL/I program that is shown in Figure 1-4 can be run as shown with the application code embedded in the z/OS job stream.

Using a claims file

The IFAM1 PL/I application program accesses a file that contains insurance data, that is, a Model 204 data file named CLAIMS90, and uses the information to determine the average settlement amount for all liability claims settled in the first half of 1990. Liability claims are indicated by the field name = value pair of CLAIM TYPE = L. The format of SETTLEMENT DATE is YYMMDD.

IFAM1 PL/I example (z/OS)

Figure 1-4 shows an HLI application program that is written for IFAM1 in PL/I which, with the JCL that is shown in the job stream, runs under an z/OS operating system.

This application uses a single cursor IFSTRT thread.

Figure 1-4. Sample PL/I program (z/OS)

```
//JOBNAME JOB ,'IFAM1 TEST',MSGCLASS=A,CLASS=A
//PLI  EXEC PGM=IEL0AA,PARM='OBJECT, NODECK',REGION=100K
//SYSPRINT   DD SYSOUT=*
//SYSLIN     DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//  SPACE=(80,(250,100))
//SYSUT1     DD DSN=&&SYSUT1,UNIT=SYSDA,
//  SPACE=(1024,(200,50),,CONTIG,ROUND),
//  DCB=BLKSIZE=1024
//SYSIN      DD *
//*
/*-------------------------------------------------------*/
/* COMPUTE AVERAGE SETTLEMENT AMOUNT FOR LIABILITY      */
/*  CLAIMS SETTLED IN FIRST HALF OF 1990                */
/*-------------------------------------------------------*/
AVGSET: PROCEDURE OPTIONS (MAIN);
/*-------------------------------------------------------*/
/* M204 HOST LANGUAGE INTERFACE (IFAM) CALLS            */
/*-------------------------------------------------------*/


DECLARE
  IFSTRT   EXT ENTRY (FIXED BIN(31), FIXED BIN(31),
CHAR(*),CHAR(*)),
  IFLOG    EXT ENTRY (FIXED BIN(31), CHAR(*)),
  IFOPEN   EXT ENTRY (FIXED BIN(31), CHAR(*)),
  IFFIND   EXT ENTRY (FIXED BIN(31), CHAR(*)),
  IFCOUNT  EXT ENTRY (FIXED BIN(31), FIXED BIN(31)),
  IFGET     EXT ENTRY (FIXED BIN(31), CHAR(*) VAR, CHAR(*)),
  IFGERR   EXT ENTRY (FIXED BIN(31), CHAR(*) VAR),
  IFFNSH    EXT ENTRY (FIXED BIN(31));

/*-------------------------------------------------*/
/* M204 CALL ARGUMENTS                              */
/*-------------------------------------------------*/
 DECLARE
  IF_RET_CODE  FIXED BIN(31),
  01 IFSTRT_ARGS,
     05 LANGUAGE       CHAR(1)         INIT ('3'),
     05 EXEC_PARMS     CHAR(255) VAR   INIT
        ('SYSOPT=144'),
     05 USER0_PARMS    CHAR(255) VAR   INIT
         ('SPCORE=10000,MINBUF=03,MAXBUF=10,LAUDIT=7'),
```

```
  01 IFLOG_ARGS,
     05 ACCOUNT_PSWD   CHAR(255) VAR   INIT
        ('JANE;JANEPSWD'),

  01 IFOPEN_ARGS,
     05 FILE_PSWD       CHAR(255) VAR  INIT
        ('CLAIMS90;'),


  01 IFFIND_ARGS,
     05 CRITERIA        CHAR(255) VAR  INIT
        ('SETTLEMENT DATE IS BEFORE 800631;CLAIM TYPE =
          L;END;'),

  01 IFCOUNT_ARGS,
     05 COUNT            FIXED  BIN(31),

  01 IFGET_ARGS,
     05 EDIT_BUFFER      CHAR (255) VAR,
     05 EDIT_SPEC        CHAR (255) VAR  INIT
        ('EDIT (SETTLEMENT AMOUNT) (J(6));'),

  01 EDIT_BUFFER_ITEMS,
     05 SETTLEMENT_AMOUNT    PICTURE '999999',

  01 IFGERR_ARGS,
     05 MESSAGE   CHAR(80) VAR;

/*-----------------------------------------------------*/
/* OTHER DECLARATIONS */
/*-----------------------------------------------------*/


 DECLARE
    M204_ERROR_AND_TERMINATION   INTERNAL CONDITION,
    TOTAL_SETTLEMENT_AMOUNT      FIXED BIN(31) INIT (0),
    AVERAGE_SETTLEMENT_AMOUNT    FIXED BIN(31) INIT (0),
    I                            FIXED BIN(31),
    PLIRETC                      BUILTIN,
    YSPRINT                         FILE PRINT;

/*-----------------------------------------------------*/
/* M204 ERROR HANDLING*/
/* CONTROL IS TRANSFERRED HERE AFTER AN M204 ERROR FROM*/
/*  A CALL. */
/* CONTROL IS TRANSFERRED BY THIS ROUTINE TO THE*/
/* PROGRAM'S END. */
/*-----------------------------------------------------*/


 ON CONDITION (M204_ERROR_AND_TERMINATION)
   BEGIN;
   PUT SKIP(2) EDIT ('***  M204 ERROR. RETURN CODE =  ',
                   IF_RET_CODE)(A, F(6) );
```

```
   CALL IFGERR (IF_RET_CODE,  MESSAGE);
   PUT SKIP EDIT ('IFGERR MESSAGE = ',
                  IFGERR_ARGS.MESSAGE) (A, A);
   CALL PLIRETC (999);

     GO TO TERMINATION;  /* AT END OF PROGRAM */
     END;

/*-------------------------------------------------------*/
/* START M204 INTERFACE AND OPEN CLAIMS90 FILE           */
/*-------------------------------------------------------*/

 CALL IFSTRT (IF_RET_CODE,LANGUAGE, EXEC_PARMS,
              USER0_PARMS);
 IF (IF_RET_CODE ¬= 0)
   THEN DO;
      PUT DATA (IFSTRT_ARGS);
      SIGNAL CONDITION (M204_ERROR_AND_TERMINATION);
   END;

 CALL IFLOG (IF_RET_CODE, ACCOUNT_PSWD);
 IF (IF_RET_CODE ¬= 0)
   THEN DO;
      PUT DATA (IFLOG_ARGS);
      SIGNAL CONDITION (M204_ERROR_AND_TERMINATION);
   END;

 CALL IFOPEN (IF_RET_CODE, FILE_PSWD);
   IF ( ¬ (IF_RET_CODE  =  0  |  IF_RET_CODE = 16 )  )
   THEN DO;
      PUT DATA (IFOPEN_ARGS);
      SIGNAL CONDITION (M204_ERROR_AND_TERMINATION);
   END;


/*-------------------------------------------------------*/
/* RETRIEVE DATA AND COMPUTE AVERAGE SETTLEMENT CLAIM  */
/*-------------------------------------------------------*/
CALL IFFIND (IF_RET_CODE, CRITERIA);

IF (IF_RET_CODE ¬= 0)
  THEN DO;
     PUT DATA (IFFIND_ARGS);
     SIGNAL CONDITION (M204_ERROR_AND_TERMINATION);
   END;


CALL IFCOUNT (IF_RET_CODE,  COUNT);
  IF (IF_RET_CODE ¬= 0)
  THEN DO;
     PUT DATA (IFCOUNT_ARGS);
     SIGNAL CONDITION (M204_ERROR_AND_TERMINATION);
  END;
```

```
  IF COUNT = 0 THEN
    DO;
       PUT LIST ('NO RECORDS IN FOUND SET-CANNOT COMPUTE
                 AVG');
       GO TO TERMINATION;
    END;

 DO  I  =  1  TO COUNT;
   CALL IFGET (IF_RET_CODE,  EDIT_BUFFER,  EDIT_SPEC);
   IF (IF_RET_CODE ¬= 0)
      THEN DO;
         PUT DATA (IFGET_ARGS);
         SIGNAL CONDITION (M204_ERROR_AND_TERMINATION);
      END;

   EDIT_BUFFER_ITEMS = EDIT_BUFFER;

   TOTAL_SETTLEMENT_AMOUNT  =  TOTAL_SETTLEMENT_AMOUNT +
   EDIT_BUFFER_ITEMS.SETTLEMENT_AMOUNT;

END;

AVERAGE_SETTLEMENT_AMOUNT = TOTAL_SETTLEMENT_AMOUNT /
  IFCOUNT_ARGS.COUNT;

PUT EDIT ('AVG SETTLEMENT AMOUNT FOR CLAIMS SETTLED
          BETWEEN ' || '01/01/90 AND 06/31/90  =  ',
          AVERAGE_SETTLEMENT_AMOUNT) (A, F(6) );


/*-------------------------------------------------------*/
/* END OF PROGRAM - TERMINATION PROCESSING               */
/*-------------------------------------------------------*/
/* FALL THROUGH TO THIS CODE IF NO M204 ERROR. BRANCH   */
/* TO THIS CODE  FROM M204_ERROR_AND_TERMINATION        */
/*    CONDITION IF THERE WAS AN M204 ERROR.             */
/*-------------------------------------------------------*/


 TERMINATION:

CALL IFFNSH (IF_RET_CODE);

IF  (IF_RET_CODE ¬= 1000)
  THEN DO;
   PUT SKIP(2)  EDIT ('*** M204 IFFNSH ERROR.
     RETURN CODE = ',
                   IF_RET_CODE)
                     (A, F(6) );


                   CALL IFGERR (IF_RET_CODE, MESSAGE);
                   PUT SKIP EDIT ('IFGERR RETURN CODE=',
                   IF_RET_CODE,
```

```
                    'MESSAGE = ', MESSAGE)
                    (A, F(6), A, A);
                  END;
END;
//LKED  EXEC PGM=IEWL,PARM='RENT,LIST,LET,MAP,
//           COND=(9,LT,PLI)
//SYSLIB     DD  DSN=SYS1,PLIBASE,DISP=SHR
//SYSUT1     DD  DSN=&&SYSUT1,UNIT=SYSDA, -
//               SPACE=(1024,(200,20)), -
//               DCB=BLKSIZE=1024
//SYSLMOD    DD  DSN=LOCAL.M204.IFAM1.APPLIC,DISP=SHR
//SYSPRINT   DD  SYSOUT=C
//OB         DD  DSN=LOCAL.M204.OBJECT,DISP=SHR
//SYSOUT     DD  SYSOUT=C
//SYSLIN     DD  DSN=&&LOADSET,DISP=(OLD,DELETE), -
//               UNIT=SYSDA
// DD                DDNAME=SYSIN
//SYSIN      DD *
     INCLUDE OB(IFIF1OS)
     NAME IFAMTEST(R)
```

Run the IFAM1 PL/I application using the following JCL:

```
//RUNIFAM1   JOB ,'RUNIFAM1,MSGLEVEL=(1,1), -
//                MSGCLASS=C,CLASS=T
//*
//IFAM1     EXEC PGM=IFAMTEST
//*
//STEPLIB   DD DSN=LOCAL.M204.IFAM1.APPLIC,DISP=SHR
//          DD DSN=LOCAL.M204.LOAD,DISP=SHR
//CCAAUDIT  DD SYSOUT=C
//CCAPRINT  DD SYSOUT=C
//CCASNAP   DD SYSOUT=C
//SYSUDUMP  DD SYSOUT=C
//CCATEMP   DD DISP=NEW,UNIT=SYSDA,SPACE=(TRK,20)
//CCASTAT   DD DSN=M204.CCASTAT,DISP=SHR
//SYSRINT   DD SYSOUT=C
//CLAIMS90   DD DSN=M204.CLAIMS90,DISP=SHR
```

IFAM1 jobs: Compiling under Enterprise PL/I for z/OS

When compiling a PL/I application under the Enterprise PL/I for z/OS compiler, the following compiler parameter is required:

```
DEFAULT(LINKAGE(SYSTEM))
```

This causes the parameter list to be built in the same way that it was built by the old compilers (including turning on the high-order bit of the address of the last parameter).

For example:

```
//PLICMPL EXEC PGM=IBMZPLI,PARM='OBJECT,OPTIONS,
```

```
//              DEFAULT(LINKAGE(SYSTEM))',REGION=512K,...
```

If this compiler option is not specified, subsequent executions of the application will fail with 0C4 abends.

FORTRAN example

This section provides a sample program written in FORTRAN to run in the IFAM1 environment under z/OS using the Model 204 Host Language Interface. The sample FORTRAN program that is shown in Figure 1-5 can be run as shown with the application code embedded in the z/OS job stream.

Using a claims file

The IFAM1 FORTRAN application program accesses a file that contains insurance data, that is, a Model 204 data file named CLAIMS90, and uses the information to determine the average settlement amount for all liability claims settled in the first half of 1990. Liability claims are indicated by the field name = value pair of CLAIM TYPE = L. The format of SETTLEMENT DATE is YYMMDD.

IFAM1 FORTRAN example (z/OS)

Figure 1-5 shows an HLI application written for IFAM1 in FORTRAN, which, with the JCL that is shown in the job stream, runs under an z/OS operating system.

This application uses a single cursor IFSTRT thread.

Figure 1-5. Sample FORTRAN program (z/OS)

```
//JOBNAME JOB 'IFAM1',MSGLEVEL=1,MSGCLASS=A
//*************************************************************
//*
//* JCL TO COMPILE AND LINK AN IFAM1 FORTRAN PROGRAM
//*
*************************************************************

//JOBLIB     DD DSN=USER.LINKLIB,DISP=SHR
//FORT   EXEC    PGM=IEYFORT
//SYSPRINT   DD SYSOUT=*
//SYSLIN     DD DSN=&&LOADSET,
//           DISP=(MOD,PASS),UNIT=SYSDA,SPACE=(80,(500,100))
//SYSIN      DD *

C
C
C  GLOSSARY:
C    ALIBC:  AVERAGE LIABILITY CLAIM
C    CNT:    ASSIGNED VALUE OF 5, USED IN ERROS (ERROR ROUTINE)
C    EDITS:  M204 VARIABLE NAMES AND THEIR EDIT MASKS (IFGET)
```

```
C    FILEID:M204 FILE NAME (IFOPEN)
C    FIN:    ASSIGNED VALUE OF 7, USED IN ERROS (ERROR ROUTINE)
C    FIND:   ASSIGNED VALUE OF 4, USED IN ERROS (ERROR ROUTINE)
C    FINDS:  FIND CRITERIA PROVIDED BY USERS (IFFIND)
C    GET:    ASSIGNED VALUE OF 6, USED IN ERROS (ERROR ROUTINE)
C    IRC:    RETURN CODE FROM HOST LANGUAGE INTERFACE CALLS
C    OPEN:   ASSIGNED VALUE OF 3, USED IN ERROS (ERROR ROUTINE)
C    LIBC:   LIABILITY CLAIM AMOUNT RETURNED FROM M204 FILE
C    LOG:    ASSIGNED VALUE OF 2, USED IN ERROS (ERROR ROUTINE)
C    LOGIN:  M204 USER-ID AND PASSWORD (IFLOG)
C    LTYPE:  LANGUAGE TYPE (FORTRAN, COBOL, PL/I, OR BAL) IN
C                 THIS CASE, FORTRAN
C    NFINDS: NUMBER OF FINDS, USED AS TEST VALUE IN DO LOOP
C    NLIBC:  THE NUMBER OF LIABILITY CLAIMS, USED TO COMPUTE
C                 THE AVERAGE
C    PARMS:  HOST LANGUAGE INTERFACE PARAMETERS
C    STRT:   ASSIGNED VALUE OF 1, USED IN ERROS (ERROR ROUTINE)
C    TLIBC:  TOTAL LIABILITY CLAIM
C    USER0:  USER ZERO INPUT FOR HOST LANGUAGE INT EXECUTION
C
C  SUBROUTINES:
C
C    IFSTRT:  STARTUP MODEL 204 HOST LANGUAGE INTERFACE THREAD
C    IFLOG:   LOGIN TO M204
C    IFOPEN:  OPEN MODEL 204 FILE
C    IFFIND:  'FIND ALL RECORDS . . .'
C    IFCNT:   'COUNT RECORDS IN . . .'
C    IFGET:   'FOR EACH RECORD IN . . .'
C    IFFNSH:  'LOGOUT'
C    ERROS:   ERROR ROUTINE (M204 ERRORS)
C
C
     REAL     ALIBC,LIBC,TLIBC
     INTEGER  PARMS(20),USER0(20),FINDS(20),EDITS(20)
     INTEGER  LOGIN(10),FILEID(10)
     INTEGER  IRC,LTYPE,NFINDS,NLIBC
     INTEGER*2 STRT,LOG,OPEN,FIND,CNT,GET,FIN
     DATA STRT /1/,
          *   LOG /2/,
          *   OPEN /3/,
          *   FIND /4/,
          *   CNT /5/,
          *   GET /6/,
          *   FIN /7/
     DATA LTYPE/2/
C
C READ IN USER INPUT
C
     READ(05,500)  (PARMS(I),I=1,20)
     READ(05,500)  (USER0(I),I=1,20)
     READ(05,500)  (LOGIN(I),I=1,10)
     READ(05,500)  (FILEID(I),I=1,10)
     READ(05,500)  (FINDS(I),I=1,20)
     READ(05,500)  (EDITS(I),I=1,20)
```

```
C
C ECHO INPUT PARAMETERS
C
     WRITE(06,610)
     WRITE(06,611)  (PARMS(I),I=1,20)
     WRITE(06,611)  (USER0(I),I=1,20)
     WRITE(06,611)  (LOGIN(I),I=1,10)
     WRITE(06,611)  (FILEID(I),I=1,10)
     WRITE(06,611)  (FINDS(I),I=1,20)
     WRITE(06,611)  (EDITS(I),I=1,20)
C
C START M204 THREAD
C
     CALL IFSTRT(IRC,LTYPE,PARMS,USER0)
     IF (IRC.EQ.0) GOTO 10
         CALL ERROS(STRT,IRC)
C
C LOGIN TO M204
C
  10 CALL IFLOG(IRC,LOGIN)
       IF (IRC.EQ.0) GOTO 20
       CALL ERROS(LOG,IRC)
C
C OPEN FILE
C
  20 CALL IFOPEN(IRC,FILEID)
             IF ((IRC.EQ. 0)    .OR.
          *       (IRC.EQ.16)    .OR.
          *       (IRC.EQ.32)) GOTO 30
             CALL ERROS(OPEN,IRC)
C
C FIND STATEMENT
C
  30 CALL  IFFIND(IRC,FINDS)
             IF  (IRC.EQ.0)  GOTO 40
             CALL ERROS(FIND,IRC)
C
C COUNT THE NUMBER OF RECORDS FOUND
C
  40 CALL  IFCNT(IRC,NFINDS)
             IF  (IRC.EQ.0)  GOTO 50
             CALL ERROS(CNT,IRC)
C
C CHECK TO SEE IF ANY RECORDS WERE FOUND
C
  50 IF  (NFINDS.GT.0)  GOTO 60
             WRITE(06,600)
             STOP 999
  60 WRITE(06,640)  NFINDS
C
C SIMULATION OF THE 'FOR EACH RECORD' LOOP
C
                DO  110    I=1,NFINDS
C
```

```
C 'GET' THE RECORDS FROM M204
C
                CALL IFGET(IRC,LIBC,EDITS)
                IF (IRC.EQ.0)  GOTO 70
                   CALL  ERROS(GET,IRC)
                   GOTO 110
C
C ADD THE LIBC TO TLIBC
C
     70              TLIBC=TLIBC+LIBC
                     NLIBC=NLIBC+1
     110 CONTINUE
C
C COMPUTE THE AVERAGE
C
                ALIBC=TLIBC/NLIBC
                WRITE(06,650) TLIBC,ALIBC
                CALL IFFNSH(IRC)
                IF  (IRC.EQ.1000)  GOTO 1000
C
C TERMINATE M204 THREAD
C
                CALL  ERROS(FIN,IRC)
          STOP 1999
     1000 STOP
     500 FORMAT(20A4)
     600 FORMAT('1',//,' REQUEST ENDED 999: NO RECORDS FOUND')
     610 FORMAT('1',' INPUT PARAMETERS')
     611 FORMAT(' ',20A4)
     640 FORMAT('1',/////,' NUMBER OF RECORDS ',I7)
     650 FORMAT(////,'TOTAL CLAIMS: ',F7.1,' AVERAGE CLAIM: ',F7.1)
     END
C
C
C GLOSSARY:
C    IERR: VECTOR OF SUBROUTINE NAMES; USED IN PRINTING ERROR
C                MESSAGES
C    MESSGE:     TEXT RETURNED FORM M204 (IFGERR)
C    IPRC:       PREVIOUS RETURN CODE
C    IRC:        RETURN CODE FROM HLI CALLS

C    ITYPE:      VALUE OF STRT, LOG, OPEN, FIND, CNT, GET, OR FIN
C
C SUBROUTINES:
C
C   IFGERR:      GET TEXT OF RETURN CODE FROM M204
C
C
                SUBROUTINE ERROS(ITYPE,IRC)
                REAL*8 IERR(7),MESSGE(10)
                INTEGER*2 ITYPE
                DATA IERR/'IFSTRT   ','IFLOG   ','IFOPEN   ',
    *            'IFFIND   ','IFCNT   ','IFGET ','IFFNSH '/
                IPRC=IRC
```

```
C
C GET RETURN CODE MESSAGE TEXT AND ABEND 999
C
                CALL IFGERR(IRC,MESSGE)
                WRITE(06,600) IERR(ITYPE),IPRC,MESSGE
   600 FORMAT('1',////,'BAD CALL:' ,A8,'RETURN CODE: ',I5,':  ',10A8)
       STOP 999
                END
/*
//LKED           EXEC PGM=IEWL,PARM='LIST,XREF,LET,RENT',
//               COND=(5,LT,FORT)
//SYSLIN         DD DSN=&&LOADSET,DISP=(OLD,DELETE)
// DD            DDNAME=SYSIN
//SYSLMOD        DD DSN=LOCAL.M204.IFAM1.APPLIC,DISP=SHR
//SYSLIB         DD DSN=SYS1.FORTLIB,DISP=SHR
//SYSUT1         DD UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//                  SPACE=(1024,(50,20))
//SYSOUT         DD SYSOUT=C
//SYSPRINT       DD SUSOUT=C
//OB               DD DSN=LOCAL.M204.OBJECT,DISP=SHR
//SYSIN DD *
  INCLUDE OB(IFIF1OS)
  NAME IFAMTEST(R)
```

Run the IFAM1 FORTRAN application using the following JCL:

```
//RUNIFAM1  JOB  ,'RUNIFAM1',MSGLEVEL=(1,1),MSGCLASS=C,CLASS=T
//*
//IFAM1     EXEC PGM=IFAMTEST
//*
//STEPLIB   DD DSN=LOCAL.M204.IFAM1.APPLIC,DISP=SHR
//          DD DSN=LOCAL.M204.LOAD,DISP=SHR
//FT05F001  DD *
SYSOPT=144;
MAXBUF=2,MINBUF=2,SPCORE=10000,LAUDIT=7;
userid;password;
CLAIMS90;
CLAIM TYPE=L;SETTLEMENT DATE IS BETWEEN 900100 AND 900631;END;
EDIT(SETTLEMENT AMOUNT) (F(4));
//CCAAUDIT   DD SYSOUT=C
//CCAPRINT   DD SYSOUT=C
//CCASNAP    DD SYSOUT=C
//SYSUDUMP   DD SYSOUT=C
//CCATEMP    DD DISP=NEW,UNIT=SYSDA,SPACE=(TRK,(20)
//CCASTAT    DD DSN=M204.CCASTAT,DISP=SHR
//SYSPRINT   DD SYSOYT=C
//CLAIMS90   DD DSN=M204.CLAIMS90,DISP=SHR
//
```

Assembler example

This section provides a sample program written in Assembler to run in the IFAM1 environment under z/OS using the Model 204 Host Language Interface.

The sample Assembler program that is shown in Figure 1-6 can be run as shown with the application code embedded in the z/OS job stream.

Using a claims file

The IFAM1 Assembler application program accesses a file that contains insurance data, that is, a Model 204 data file named CLAIMS90, and uses the information to determine the average settlement amount for all liability claims settled in the first half of 1990. Liability claims are indicated by the field name = value pair of CLAIM TYPE = L. The format of SETTLEMENT DATE is YYMMDD.

IFAM1 Assembler example (z/OS)

Figure 1-6 shows an HLI application written for IFAM1 in Assembler which, with the JCL that is shown in the job stream, runs under an z/OS operating system.

This application uses a single cursor IFSTRT thread.

Figure 1-6. Sample Assembler program (z/OS)

```
//JOBNAME JOB      CLASS=A,MSGCLASS=A
//ASM    EXEC PGM=IEV90,PARM='NODECK,OBJECT,XREF(SHORT)',
//            REGION=512K
//SYSLIB      DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1      DD SPACE=(CYL,(4,2)),UNIT=SYSDA
//SYSPRINT    DD SYSOUT=A
//SYSPUNCH    DD SYSOUT=A
//SYSLIN      DD DSN=LOCAL.M204.OBJLIB(IFAMT1),DISP=SHR
//SYSIN        DD *

IFAMT1  TITLE 'IFAM1 JOB'
R0      EQU    0
R1      EQU    1
R2      EQU    2
R3      EQU    3
R4      EQU    4
R5      EQU    5
R6      EQU    6
R7      EQU    7
R8      EQU    8
R9      EQU    9
RA      EQU    10
RB      EQU    11
RC      EQU    12
RD      EQU    13
RE      EQU    14
RF      EQU    15
IFAMT1  CSECT
        STM    RE,RC,12(RD)   SAVE THEIR REGISTERS
        LR     RB,RF          SET BASE REGISTER
        USING  AMT1,RB        ESTABLISH ADDRESSABILITY
```

```
         ST     RD,SAVEAREA+4  STORE THEIR SAVEAREA POINTER
         LA     R2,SAVEAREA    GET POINTER TO OUR SAVEAREA
         ST     R2,8(,RD)      STORE OUR SAVEAREA POINTER
         LR     RD,R2          POINT TO OUR SAVEAREA
************************************************************
* MAIN PROCESSING LOOP
************************************************************


MAIN     DS     0H             INITIALIZATION AND CONNECT
         MVC    THRDNO(4),RESET  INITIALIZE THREAD NUMBER
         CALL IFSTRTN,(RETCODE,LANGIND,LOGON,MODEUP,THRDNO,OSCHNL),VL
         L      R7,RETCODE     GET RETURN CODE
         LTR    R7,R7          DID WE GET A RETURN CODE ZERO
         BNZ    ENDIT          NO, GO END IT
         CALL IFOPEN,(RETCODE,FILEDATA),VL OPEN FILE
         L      R7,RETCODE     GET RETURN CODE
         LTR    R7,R7          NO, DID WE GET A RETURN CODE ZERO
         BNZ    TERM           GO FINISH UP
         CALL IFFIND,(RETCODE,SELECT),VL FIND M204 RECORDS
         L      R7,RETCODE     GET RETURN CODE
         LTR    R7,R7          DID WE GET A RETURN CODE ZERO
         BNZ    TERM           NO, GO CLOSE FILE AND FINISH
GETLOOP  DS     0H             GET ALL RECORDS
         CALL IFGET,(RETCODE,GETAREA,GETLIST),VL
         CLC    RETCODE,ENDSET   END OF FOUND SET = 2
         BE     TERM           YES, EXIT
         CALL IFPUT,(RETCODE,GETAREA,GETLIST),VL
TERM     CALL IFFNSH,(RETCODE),VL CLOSE M204 CONNECTION
ENDIT    L      RD,SAVEAREA+4  RESTORE R13 TO THEIR SAVEAREA
         ST     R7,16(,RD)     SET R15 TO RETURN CODE
         LM     RE,RC,12(RD)   RESTORE THEIR REGISTERS
         BR     RE             RETURN
        EJECT
************************************************************
* VARIABLES AND CONSTANTS
*
************************************************************
SAVEAREA DS     18F            IFAM REGISTER SAVE AREA
RETCODE  DC     F'0'           MODEL 204 RETURN CODE AREA
LANGIND  DC     F'2'           MODEL 204 LANGUAGE IND AREA
MODEUP   DC     F'1'           INDICATES UPDATE MODE
THRDNO   DC     F'0'           THREAD NUMBER
RESET    DC     F'0'           RESET THREAD NUMBER
IFERROR  DC     F'4'           ERROR RETURN CODE FOR IFSTRTN
ENDSET   DC     F'2'           END OF FOUND SET (IFGET)
FILEDATA DC     CL14'FILE CLAIMS90;'   DATA FOR IFOPEN
SELECT   DC     CL10'A=999;END;'       DATA FOR IFFIND
GETLIST  DC     CL5'DATA;'             DATA FOR IFGET
GETAREA  DS     CL40            DATA AREA FOR IFGET
CHAN     DC     CL8' '          CHANNEL NAME AREA
OSCHNL   DC     CL8'IFAMTEST'   DEFAULT TEST CHANNEL NAME
LOGON    DC     CL11'USER1;PSW1;'      M204 LOGON
         END    IFAMT1
```

```
//*
//LKED     EXEC PGM=IEWL,PARM='LET,LIST,MAP,SIZE=(256K,64K),
//                            XREF',REGION=512K
//SYSPRINT DD SYSOUT=A
//SYSLMOD  DD DSN=LOCAL.M204.LOADLIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(TRK,(15,10))
//CCA      DD DSN=LOCAL.M204.OBJLIB,DISP=SHR
//SYSLIN   DD *
   INCLUDE CCA(IFAMT1)
   INCLUDE CCA(IFIF1OS)
   NAME IFAMT1(R)
/*
```

SOUL (User Language) example

The three programs (PL/I, FORTRAN, and Assembler) in the preceding sections all use the same insurance file (that is, CLAIMS90) and the same HLI functions to determine the average settlement amount for all liability claims settled in the first half of 1990.

The SOUL solution is based on the same example as the sample PL/I, FORTRAN, and Assembler programs in the preceding sections.

For each action, or Model 204 command, or SOUL statement on the left, the column on the right shows the corresponding action of the HLI application, which includes any HLI call. Note that the HLI application below uses a single cursor IFSTRT thread.

SOUL solution	HLI application
Connect (dial the number and give the Model 204 application ID)	IFSTRT
Log in	IFLOG
OPEN CLAIMS90 (and enter password)	IFOPEN
BEGIN LOCATE: FD CLAIM TYPE = L SETTLEMENT DATE IS BETWEEN - 900100 AND 900631	IFFIND (with similar criteria)
TALLY: COUNT RECORDS IN LOCATE	IFCOUNT (assigns the count to a variable)
FR LOCATE	Loop
SUM: %TOTAL = %TOTAL + - SETTLEMENT AMOUNT	IFGET assigns the SETTLEMENT AMOUNT to a variable (and code computes the total)

SOUL solution	HLI application
AVRG: %AVG = %TOTAL/COUNT IN SUM	Code performs computation
PRT: PRINT %AVG END	Code performs a print function
CLOSE CLAIMS90	IFCLOSE
LOGOUT DISCONNECT	IFFNSH

Note: In addition to the functions listed, the HLI application must use IFGERR for error processing, because the error messages cannot go to the terminal user.

B

IFAM2/IFAM4 Job Program Samples

Overview

This appendix provides examples of IFAM2 and IFAM4 jobs, complete with program code, which includes a multiple cursor IFSTRT thread application. The sample programs illustrate the use of Host Language Interface functions in the IFAM2 and IFAM4 processing environments.

For more information

Refer to Appendix for examples of HLI applications written in different host languages and job setups that may be run in IFAM1. Refer to the *Rocket Model 204 Host Language Interface Programming Guide* for coding examples related to particular aspects of HLI processing.

Multiple cursor IFSTRT thread example

This section provides a sample program written in COBOL to run in the IFAM2 environment under CMS using the Model 204 Host Language Interface.

The COBOL program illustrates how to establish a multiple cursor IFSTRT thread. The application opens two files, VEHICLES and CLIENTS, and updates records in the CLIENTS file using data from the VEHICLES file. See page B-7 for the sample output generated by the execution of this program.

Note: The sample COBOL program in Figure can be compiled, linked, and loaded with the CMS EXEC on page B-15. This same program

(with no changes) could be run in an z/OS or VSE operating system environment by using the necessary JCL (which is not provided).

Refer to Chapter 2 and to the *Rocket Model 204 Host Language Interface Programming Guide* for more information about multiple cursor IFSTRT threads.

Figure B-1. Sample program: Multiple cursor IFSTRT thread

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  EX2
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-4381.
 OBJECT-COMPUTER.  IBM-4381.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*********************************************
* SAMPLE PROGRAM WHICH UPDATES THE CLIENTS FILE USING
* DATA CALCULATED FROM THE VEHICLES FILE.
* THIS PROGRAM USES A MULTIPLE CURSOR IFSTRT THREAD
*
* FOR PURPOSES OF TESTING, YOU CAN LIMIT THIS TO THE FIRST 10
* POLICYHOLDERS IN THE CLIENTS FILE (USE VARIABLE LOOPCTR TO
* CONTROL THIS AND UNCOMMENT LOOPCTR CHECK).
*
*********************************************

 01 INTEGER-ARGS     COMP SYNC.
    05  STAT-IND     PIC 9(5).
    05  LANG-IND     PIC 9(5)   VALUE 2.
    05  THRD-TYPE    PIC 9(5)   VALUE 2.
    05  THRD-NAME    PIC 9(5).
    05  FETCH-DIRECTION PIC 9(5) VALUE 1.

 01  CHAR-ARGS.
    05  LOGIN        PIC X(13)  VALUE "USER01;PASSW;".
    05  VEH-PARM     PIC X(10)  VALUE "VEHICLES;;".
    05  CLIENT-PARM  PIC X(9)   VALUE "CLIENTS;;".
    05  CHAN-NAME    PIC X(9)   VALUE "MSPIFM22;".
    05  IFGERR-MESSAGE PIC X(80) VALUE SPACES.
    05  POL-FIND     PIC X(6)   VALUE "POLFD;".
    05  POL-SET      PIC X(9)   VALUE "IN POLFD;".
    05  POL-CURSOR   PIC X(7)   VALUE "POLCUR;".
    05  POL-FETCH-NAME PIC X(8) VALUE "POLFTCH;".
    05  IFUPDT-POL   PIC X(7)   VALUE "UPDPOL;".

    05  FIND-POLICY  PIC X(39)  VALUE
        "IN CLIENTS FD RECTYPE=POLICYHOLDER;END;".

    05  FIND-VEH     PIC X(41)  VALUE
        "IN VEHICLES FD OWNER POLICY=%POLICY;END;".
    05  POL-BUF.
        10  OWN-POL      PIC X(6)   VALUE SPACES.
```

```
    05  FIND-VEH-EDIT PIC X(22) VALUE
        "EDIT (%POLICY) (A(6));".

    05  VEH-FIND     PIC X(6)   VALUE "VEHFD;".
    05  VEH-SET      PIC X(9)   VALUE "IN VEHFD;".
    05  VEH-CURSOR   PIC X(7)   VALUE "VEHCUR;".
    05  VEH-FETCH-NAME PIC X(8)  VALUE "VEHFTCH;".

    05  EDIT-1       PIC X(24)  VALUE
        "EDIT (POLICY NO) (A(6));".

    05  EDIT-2       PIC X(30)  VALUE
"EDIT (VEHICLE PREMIUM) (Z(8));".

    05  EDIT-3       PIC X(29)  VALUE
        "EDIT (TOTAL PREMIUM) (Z(10));".

 01  WORK-AREA.
    05  PREM-AMT     PIC 9(8)   VALUE ZEROES.
    05  TOTAL-PREM   PIC 9(10)  VALUE ZEROES.

 01  CALLNAME          PIC X(20).
 01  DISPLAY-STAT-IND  PIC 9(4).
 01  LOOPCTR           PIC 9(3) COMP-3 VALUE 0.
 01  MORE-POL-FLAG     PIC XXX VALUE "YES".
    88  NO-MORE-POL           VALUE "NO".
 01  MORE-VEH-FLAG     PIC XXX VALUE "YES".
    88  NO-MORE-VEH           VALUE "NO".

 PROCEDURE DIVISION.
 MAINLINE.
    CALL "IFSTRTN" USING STAT-IND,
         LANG-IND,
         LOGIN,
         THRD-TYPE,
         THRD-NAME,
         CHAN-NAME.
    MOVE "IFSTRTN" TO CALLNAME.
    PERFORM RC-CHECK.


         CALL "IFOPEN" USING STAT-IND, VEH-PARM.
         MOVE "IFOPEN" TO CALLNAME.
         PERFORM RC-CHECK.

         CALL "IFOPEN" USING STAT-IND, CLIENT-PARM.
         MOVE "IFOPEN" TO CALLNAME.
         PERFORM RC-CHECK.

         CALL "IFFIND" USING STAT-IND, FIND-POLICY, POL-FIND.
         MOVE "IFFIND-P" TO CALLNAME.
         PERFORM RC-CHECK.

         CALL "IFOCUR" USING STAT-IND, POL-SET, POL-CURSOR.
```

```
         MOVE "IFOCUR-P" TO CALLNAME.
         PERFORM RC-CHECK.


      GET-FIRST-POLICY.
         CALL "IFFTCH" USING STAT-IND, OWN-POL,
              FETCH-DIRECTION, POL-CURSOR, EDIT-1,
              POL-FETCH-NAME.
         DISPLAY "POLNO RETRIEVED IS: " OWN-POL
         IF STAT-IND EQUAL 2  THEN
             MOVE "NO" TO MORE-POL-FLAG
         ELSE
             MOVE "IFFTCH-P1" TO CALLNAME
             PERFORM RC-CHECK.

         PERFORM UPDATE-POLICIES UNTIL NO-MORE-POL.

         PERFORM M204-DISCONN.
         STOP RUN.

*******  E N D   O F   M A I N L I N E   C O D E  ******


*************** PERFORMED ROUTINES FOLLOW  ************
 RC-CHECK.
    MOVE STAT-IND TO DISPLAY-STAT-IND.
    IF STAT-IND NOT EQUAL 0 THEN
         DISPLAY "****** SERIOUS IFAM ERROR *******"
         DISPLAY "IFAM CALL: " CALLNAME ", RC: "
           DISPLAY-STAT-IND
         CALL "IFGERR" USING STAT-IND, IFGERR-MESSAGE
         DISPLAY "MESSAGE:" IFGERR-MESSAGE
         PERFORM M204-DISCONN
         STOP RUN.



 M204-DISCONN.
    DISPLAY "DISCONNECTING FROM M204 NOW".
    CALL "IFFNSH" USING STAT-IND.
    IF STAT-IND NOT EQUAL 1000 THEN
         MOVE STAT-IND TO DISPLAY-STAT-IND
         DISPLAY "IFAM CALL: IFFNSH, RC: " DISPLAY-STAT-IND.


 UPDATE-POLICIES.
   ADD 1 TO LOOPCTR.
   CALL "IFFIND" USING STAT-IND, FIND-VEH, VEH-FIND,
        OWN-POL, FIND-VEH-EDIT.
   MOVE "IFFIND-V" TO CALLNAME.
   PERFORM RC-CHECK.
   CALL "IFOCUR" USING STAT-IND, VEH-SET, VEH-CURSOR.
   MOVE "IFOCUR-V" TO CALLNAME.
   PERFORM RC-CHECK.
```

```
*
*  GET FIRST CORRESPONDING VEHICLE RECORD.
*
   MOVE "YES" TO MORE-VEH-FLAG
   CALL "IFFTCH" USING STAT-IND, PREM-AMT, FETCH-DIRECTION,
        VEH-CURSOR, EDIT-2, VEH-FETCH-NAME.
   IF STAT-IND EQUAL 2  THEN
        MOVE "NO" TO MORE-VEH-FLAG
   ELSE
        MOVE "IFFTCH-V1" TO CALLNAME
        PERFORM RC-CHECK.
   MOVE ZEROES TO TOTAL-PREM.
   PERFORM SUM-VEHICLES UNTIL NO-MORE-VEH.

   CALL "IFCCUR" USING STAT-IND, VEH-CURSOR.
   MOVE "IFCCUR-V" TO CALLNAME.
   PERFORM RC-CHECK.


*
*  UPDATE POLICYHOLDER RECORD WITH CALCULATED TOTAL
*
   CALL "IFUPDT" USING STAT-IND, TOTAL-PREM, POL-CURSOR,
        EDIT-3, IFUPDT-POL.
   MOVE "IFUPDT" TO CALLNAME.
   PERFORM RC-CHECK.
   DISPLAY "CALCULATED PREMIUM IS:" TOTAL-PREM
*
*  GET NEXT POLICYHOLDER RECORD TO BE PROCESSED.
*
   CALL "IFFTCH" USING STAT-IND, OWN-POL, FETCH-DIRECTION,
        POL-CURSOR, EDIT-1, POL-FETCH-NAME.
   DISPLAY "POLNO RETRIEVED IS: " OWN-POL
   IF STAT-IND EQUAL 2  THEN
        MOVE "NO" TO MORE-POL-FLAG
   ELSE
        MOVE "IFFTCH-P2" TO CALLNAME
        PERFORM RC-CHECK.
   IF LOOPCTR > 10 THEN MOVE "NO" TO MORE-POL-FLAG.

 SUM-VEHICLES.
   COMPUTE TOTAL-PREM = TOTAL-PREM + PREM-AMT.


*
*  GET NEXT VEHICLE RECORD FOR THIS POLICY
*
   CALL "IFFTCH" USING STAT-IND, PREM-AMT, FETCH-DIRECTION,
       VEH-CURSOR, EDIT-2, VEH-FETCH-NAME.
   IF STAT-IND EQUAL 2  THEN
       MOVE "NO" TO MORE-VEH-FLAG
   ELSE
```

```
       MOVE "IFFTCH-V2" TO CALLNAME
   PERFORM RC-CHECK.
*************** END OF PROGRAM ****************
```

Sample output from program

The sample output in Figure B-2 is generated by executing the sample COBOL program that starts in Figure B-1 on page 366 after it has been compiled, linked, and link-edited using “CMS EXEC examples” on page 375.

The first line shows the system prompt at the CMS terminal (that is, Ready;). The second line shows the typed user entry, which is the command to execute the program (stored as EX2MCIN).

The output lines generated by the execution of the program display a policy number and calculated premium value for each record that is updated in the CLIENTS file.

Figure B-2. Sample program output: Multiple cursor IFSTRT thread

```
Ready;
ex2mcin
 POLNO RETRIEVED IS: 111111
 CALCULATED PREMIUM IS:0000000000
 POLNO RETRIEVED IS: 100340
 CALCULATED PREMIUM IS:0000000291
 POLNO RETRIEVED IS: 100642
 CALCULATED PREMIUM IS:0000000189
 POLNO RETRIEVED IS: 100037
 CALCULATED PREMIUM IS:0000000689
 POLNO RETRIEVED IS: 100944
 CALCULATED PREMIUM IS:0000001022
 POLNO RETRIEVED IS: 100060
 CALCULATED PREMIUM IS:0000000077
 POLNO RETRIEVED IS: 100774
 CALCULATED PREMIUM IS:0000000464
 POLNO RETRIEVED IS: 100035
 CALCULATED PREMIUM IS:0000000801
 POLNO RETRIEVED IS: 100942
 CALCULATED PREMIUM IS:0000000214
 POLNO RETRIEVED IS: 100640
 CALCULATED PREMIUM IS:0000000343
 POLNO RETRIEVED IS: 100338
 CALCULATED PREMIUM IS:0000000562
 POLNO RETRIEVED IS: 100080
 DISCONNECTING FROM M204 NOW
```

Multithreaded (single cursor) IFSTRT example

This section provides a sample program written in COBOL to run in the IFAM2 environment under CMS using the Model 204 Host Language Interface.

The COBOL program establishes two single cursor IFSTRT threads and opens a file on each thread (VEHICLES and CLIENTS). The application switches between the threads to update records in the CLIENTS file using data from the VEHICLES file.

This program modifies the database in a similar way as the example that uses a multiple cursor IFSTRT thread in Figure B-1 on page 366; however, there are differences in coding. See Figure B-4 on page 375 for the sample output generated by the execution of this program.

Note: The sample COBOL program in Figure B-3 can be compiled, linked, and loaded with the “CMS EXEC examples” on page 375. This same program (with no changes) could be run in an z/OS or VSE operating system environment by using the necessary JCL (which is not provided).

Figure B-3. Sample program: Multithreaded IFSTRT

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  EX2
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-4381.
 OBJECT-COMPUTER.  IBM-4381.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
***********************************************
* SAMPLE PROGRAM WHICH UPDATES THE CLIENTS FILE USING
* DATA CALCULATED FROM THE VEHICLES FILE.
* THIS PROGRAM USES TWO SINGLE CURSOR IFSTRT THREADS.
*
* FOR PURPOSES OF TESTING, YOU CAN LIMIT THIS TO THE FIRST 10
* POLICYHOLDERS IN THE CLIENTS FILE (USE VARIABLE LOOPCTR TO
* CONTROL THIS AND UNCOMMENT LOOPCTR CHECK).
*
***********************************************
01  INTEGER-ARGS     COMP SYNC.
    05  STAT-IND     PIC 9(5).
    05  LANG-IND     PIC 9(5)   VALUE 2.
    05  UPD-YES      PIC 9(5)   VALUE 1.
    05  UPD-NO       PIC 9(5)   VALUE 0.
    05  CLIENT-THRD  PIC 9(5).
    05  VEH-THRD     PIC 9(5).
    05  OLD-THRD     PIC 9(5).

01  CHAR-ARGS.
    05  LOGIN        PIC X(13)  VALUE "USER01;PASSW;".
    05  VEH-PARM     PIC X(10)  VALUE "VEHICLES;;".
    05  CLIENT-PARM  PIC X(9)   VALUE "CLIENTS;;".
    05  CHAN-NAME    PIC X(9)   VALUE "MSPIFM22;".
    05  IFGERR-MESSAGE PIC X(80) VALUE SPACES.

    05  FIND-POLICY  PIC X(25)  VALUE
        "RECTYPE=POLICYHOLDER;END;".
```

```
    05  FIND-VEH.
        10  FILLER   PIC X(13)  VALUE "OWNER POLICY=".
        10  OWN-POL  PIC X(6)   VALUE SPACES.
        10  FILLER   PIC X(5)   VALUE ";END;".

    05  EDIT-1       PIC X(24)  VALUE
        "EDIT (POLICY NO) (A(6));".

    05  EDIT-2       PIC X(30)  VALUE
        "EDIT (VEHICLE PREMIUM) (Z(8));".

    05  EDIT-3       PIC X(29)  VALUE
        "EDIT (TOTAL PREMIUM) (Z(10));".

01  WORK-AREA.
    05  PREM-AMT     PIC 9(8)   VALUE ZEROES.
    05  TOTAL-PREM   PIC 9(10)  VALUE ZEROES.

01  CALLNAME          PIC X(8).
01  DISPLAY-STAT-IND  PIC 9(4).
01  LOOPCTR           PIC 9(3) COMP-3 VALUE 0.
01  MORE-POL-FLAG     PIC XXX VALUE "YES".
    88  NO-MORE-POL           VALUE "NO".
01  MORE-VEH-FLAG     PIC XXX VALUE "YES".
    88  NO-MORE-VEH           VALUE "NO".

PROCEDURE DIVISION.
MAINLINE.
  CALL "IFSTRTN" USING STAT-IND,
       LANG-IND,
       LOGIN,
       UPD-NO,
       VEH-THRD,
       CHAN-NAME.
  MOVE "IFSTRTN" TO CALLNAME.
  PERFORM RC-CHECK.

  CALL "IFOPEN" USING STAT-IND, VEH-PARM.
  MOVE "IFOPEN" TO CALLNAME.
  PERFORM RC-CHECK.

  CALL "IFSTRTN" USING STAT-IND,
       LANG-IND,
       LOGIN,
       UPD-YES,
       CLIENT-THRD,
       CHAN-NAME.
  MOVE "IFSTRTN" TO CALLNAME.
  PERFORM RC-CHECK.

  CALL "IFOPEN" USING STAT-IND, CLIENT-PARM.
  MOVE "IFOPEN" TO CALLNAME.
  PERFORM RC-CHECK.
```

```
  CALL "IFFIND" USING STAT-IND, FIND-POLICY.
  MOVE "IFFIND" TO CALLNAME.
  PERFORM RC-CHECK.

GET-FIRST-POLICY.
  CALL "IFGET" USING STAT-IND, OWN-POL, EDIT-1.
  DISPLAY "POLNO RETRIEVED IS: " OWN-POL
  IF STAT-IND EQUAL 2  THEN
      MOVE "NO" TO MORE-POL-FLAG
  ELSE
      MOVE "IFGET-P1" TO CALLNAME
      PERFORM RC-CHECK.

  PERFORM UPDATE-POLICIES UNTIL NO-MORE-POL.

  PERFORM M204-DISCONN.
  STOP RUN.
*******  E N D   O F   M A I N L I N E   C O D E  ******

*************** PERFORMED ROUTINES FOLLOW  ************



RC-CHECK.
  MOVE STAT-IND TO DISPLAY-STAT-IND.
  IF STAT-IND NOT EQUAL 0 THEN
       DISPLAY "****** SERIOUS IFAM ERROR *******"
       DISPLAY "IFAM CALL: " CALLNAME ", RC: " DISPLAY-STAT-IND
       CALL "IFGERR" USING STAT-IND, IFGERR-MESSAGE
       DISPLAY "IFGERR-MESSAGE"
       PERFORM M204-DISCONN
       STOP RUN.

M204-DISCONN.
  DISPLAY "DISCONNECTING FROM M204 NOW".
  CALL "IFFNSH" USING STAT-IND.
  IF STAT-IND NOT EQUAL 1000 THEN
       MOVE STAT-IND TO DISPLAY-STAT-IND
       DISPLAY "IFAM CALL: IFFNSH, RC: " DISPLAY-STAT-IND.

UPDATE-POLICIES.
  ADD 1 TO LOOPCTR.
  CALL "IFSTHRD" USING STAT-IND, VEH-THRD, OLD-THRD.
  MOVE "IFSTHRD" TO CALLNAME.
  PERFORM RC-CHECK.

  CALL "IFFIND" USING STAT-IND, FIND-VEH.
  MOVE "IFFIND" TO CALLNAME.
  PERFORM RC-CHECK.
*
*  GET FIRST CORRESPONDING VEHICLE RECORD.
*
  MOVE "YES" TO MORE-VEH-FLAG
```

```
  CALL "IFGET" USING STAT-IND, PREM-AMT, EDIT-2.
  IF STAT-IND EQUAL 2  THEN
       MOVE "NO" TO MORE-VEH-FLAG
  ELSE
       MOVE "IFGET-V1" TO CALLNAME
       PERFORM RC-CHECK.
  MOVE ZEROES TO TOTAL-PREM.
  PERFORM SUM-VEHICLES UNTIL NO-MORE-VEH.

*
*  SWITCH BACK TO CLIENTS THREAD AND POLICYHOLDER RECORD
*
  CALL "IFSTHRD" USING STAT-IND, CLIENT-THRD, OLD-THRD.
  MOVE "IFSTHRD" TO CALLNAME.
  PERFORM RC-CHECK.
*
*  UPDATE POLICYHOLDER RECORD WITH CALCULATED TOTAL
*
  CALL "IFPUT" USING STAT-IND, TOTAL-PREM, EDIT-3.
  MOVE "IFPUT" TO CALLNAME.
  PERFORM RC-CHECK.
  DISPLAY "CALCULATED PREMIUM IS:" TOTAL-PREM
*
*  GET NEXT POLICYHOLDER RECORD TO BE PROCESSED.
*
  CALL "IFGET" USING STAT-IND, OWN-POL, EDIT-1.
  DISPLAY "POLNO RETRIEVED IS: " OWN-POL
  IF STAT-IND EQUAL 2  THEN
       MOVE "NO" TO MORE-POL-FLAG
  ELSE
       MOVE "IFGET-P2" TO CALLNAME
       PERFORM RC-CHECK.
  IF LOOPCTR > 10 THEN MOVE "NO" TO MORE-POL-FLAG.

SUM-VEHICLES.
   COMPUTE TOTAL-PREM = TOTAL-PREM + PREM-AMT.
*
*  GET NEXT VEHICLE RECORD FOR THIS POLICY
*
   CALL "IFGET" USING STAT-IND, PREM-AMT, EDIT-2.
   IF STAT-IND EQUAL 2  THEN
       MOVE "NO" TO MORE-VEH-FLAG
   ELSE
       MOVE "IFGET-V2" TO CALLNAME
       PERFORM RC-CHECK.

*************** END OF PROGRAM ***************
```

Sample output from program

The sample output in Figure B-4 is generated by executing the sample COBOL program that starts in Figure B-3 on page 371 after it has been compiled,

linked, and link-edited using the CMS EXEC that is shown in Figure B-5 on page 376.

The first line shows the system prompt at the CMS terminal (that is, Ready;). The second line shows the typed user entry, which is the command to execute the program (stored as EX2MCIN).

The output lines generated by the execution of the program display a policy number and calculated premium value for each record that is updated in the CLIENTS file.

Figure B-4. Sample program output: Multithreaded IFSTRT

```
Ready;
ex2ifm2
 POLNO RETRIEVED IS: 100340
 CALCULATED PREMIUM IS:0000000291
 POLNO RETRIEVED IS: 100642
 CALCULATED PREMIUM IS:0000000189
 POLNO RETRIEVED IS: 100037
 CALCULATED PREMIUM IS:0000000689
 POLNO RETRIEVED IS: 100944
 CALCULATED PREMIUM IS:0000001022
 POLNO RETRIEVED IS: 100060
 CALCULATED PREMIUM IS:0000000077
 POLNO RETRIEVED IS: 100774
 CALCULATED PREMIUM IS:0000000464
 POLNO RETRIEVED IS: 100035
 CALCULATED PREMIUM IS:0000000801
 POLNO RETRIEVED IS: 100942
 CALCULATED PREMIUM IS:0000000214
 POLNO RETRIEVED IS: 100640
 CALCULATED PREMIUM IS:0000000343
 POLNO RETRIEVED IS: 100338
 CALCULATED PREMIUM IS:0000000562
 POLNO RETRIEVED IS: 100080
 CALCULATED PREMIUM IS:0000000111
 POLNO RETRIEVED IS: 100584
 DISCONNECTING FROM M204 NOW
```

CMS EXEC examples

Figure B-5 and Figure B-6 show sample EXECs that can be used to compile, link, and run an HLI application program in the IFAM2 environment under CMS.

Note that these EXECs can be used with either of the sample IFAM2 COBOL applications in Figure B-1 on page 366 or Figure B-3 on page 371.

Example of an EXEC that compiles and links the program

Figure B-5 shows an EXEC that compiles, links, and link-edits an IFAM2 application running under CMS. The EXEC prompts the user for the name of

the COBOL IFAM2 program that is to be compiled and linked. It then loads the program.

Note that M204IFAM TXTLIB must be available on an accessed disk to run this EXEC. See Figure B-6 on page 376 for an example of the M204IFAM EXEC, which must be accessible on the machine. Note that the name of COBOL TXTLIBs might vary.

Figure B-5. EXEC to compile, link, and link-edit IFAM2 program (CMS)

```
/*  Exec to compile and link an IFAM2 program   */
trace off
address command
parse upper arg pgmname

say 'Compiling program now, pgm =' pgmname
'COBOL' pgmname
say 'rc =' rc

if rc = 0 then do
  say 'Loading program now'
  'GLOBAL TXTLIB COBOLVS COBOLIBVS M204IFAM'
  'LOAD' pgmname 'IFCM'
  end
else
  exit rc
say 'rc =' rc
if rc = 0 then do
  say "Gen'ing module now"
  'GENMOD' pgmname
  end
say 'rc =' rc

exit
```

Example of M204IFAM EXEC that must be accessible

Figure B-6 shows an EXEC that starts an IFAM2 COBOL program running in the user machine. The M204IFAM EXEC is required to assign the channel name to the user ID.

The EXEC assumes that an IFAM2 COBOL program has previously been compiled, linked, and loaded, as is done, for example, with the EXEC Figure B-5 on page 376.

Figure B-6. Sample EXEC: M204IFAM (CMS)

```
/*  REXX program to establish connection*/
/*  to M204 Online */
trace off
parse upper arg channel
if channel = 'MQFIFAMC' then
  push 'DEVCHAN'
```

```
else
  push 'USERID1'
exit 0
```

Compiled IFAM on a single cursor IFSTRT thread

This section provides a sample program written in COBOL to run in the IFAM2 environment using the Model 204 Host Language Interface.

The COBOL program illustrates how to use Compiled IFAM calls on a single cursor IFSTRT thread.

Note: The sample COBOL program in Figure could be run in an z/OS, VSE, or CMS operating system environment by using the necessary JCL or EXECs (which are not provided).

Refer to Chapter 5 and to the *Rocket Model 204 Host Language Interface Programming Guide* for more information about compiled IFAM.

Figure B-7. Sample program: Compiled IFAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    IFAMTEST.
AUTHOR.
    JOE ZEE.
DATE-WRITTEN.
    MAY 15, 1990.
*
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT REPORT-FILE ASSIGN TO UT-S-REPORT.

*
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  INTEGER-CALL-ARGS         COMP SYNC.
    05 STATUS-IND             PIC 9(5).
    05 IFCOUNT-COUNT          PIC 9(5).
    05 LANGUAGE-IND           PIC 9(5) VALUE 2.
    05 READ-IND               PIC 9(5) VALUE 0.
    05 UPDT-IND               PIC 9(5) VALUE 1.
    05 THRD1                  PIC 9(5).

*
01  STRING-CALL-ARGS.
```

```
    05  M204-ERR-MESSAGE      PIC X(80) VALUE SPACES.
    05  ERROR-FUNCTION        PIC X(8).
    05  IFSTRT-LOGIN          PIC X(20) VALUE
        "SUPERKLUGE;PIGFLOUR;".
    05  IFOPEN-FILE-PARM      PIC X(10) VALUE
        "CLAIMS80;;".
    05  IFFIND-NAME           PIC X(6) VALUE
        "FIND1;".
    05  IFFIND-SPEC           PIC X(13) VALUE
        "KEY1=ZEE;END;".
    05  IFFIND-SPEC-1         PIC X(14) VALUE
        "KEY1=%KEY;END;".
    05  IFFIND-SPEC-2         PIC X(18) VALUE
        "EDIT(%KEY) (A(3));".
*
    05  IFGET-NAME-1          PIC X(5) VALUE
        "GET1;".
    05  IFGET-NAME-2          PIC X(5) VALUE
        "GET2;".
    05  IFGET-EDIT-SPEC-1     PIC X(28) VALUE
        "EDIT(KEY1,FORD) (A(3),A(5));".
    05  IFGET-EDIT-SPEC-2     PIC X(19) VALUE
        "EDIT(FORD) (A(21));".
    05  IFGET-SEQUENCE        PIC X(17) VALUE
        "IN ORDER BY NORD;".
*
    05  IFFIND-DATA           PIC X(3) VALUE "UNO".
    05  IFGET-FIELDS-1.
        10 IFGET-KEY          PIC X(3) VALUE SPACES.
        10 IFGET-FORD         PIC X(5) VALUE SPACES.
    05  IFGET-FIELDS-2        PIC X(21) VALUE SPACES.
    05  IFGET-DUM-1           PIC X VALUE ";".
    05  IFGET-DUM-2           PIC X VALUE ";".

*
 PROCEDURE DIVISION.
*************************************************************
*
*  THIS MODEL IS A SAMPLE IFAM2 (OR IFAM4) COBOL PROGRAM  *
*
*************************************************************
 MAIN-ROUTINE.
*
 INITIALIZATION.
*   OPEN OUTPUT REPORT-FILE.
    CALL "IFSTRT" USING STATUS-IND, LANGUAGE-IND,
         IFSTRT-LOGIN, UPDT-IND, THRD1.
    DISPLAY "IFSTRT RETCODE: " STATUS-IND.
    IF STATUS-IND IS NOT EQUAL ZERO
         MOVE "IFSTRT " TO ERROR-FUNCTION
         GO TO ERROR-ROUTINE.
 *
 PROCESS-1.
    CALL "IFOPEN" USING STATUS-IND, IFOPEN-FILE-PARM.
```

```
    DISPLAY "IFOPEN RETCODE: " STATUS-IND.
    IF STATUS-IND IS NOT EQUAL ZERO AND
         STATUS-IND IS NOT EQUAL TO 16 AND
         STATUS-IND IS NOT EQUAL TO 32
         MOVE "IFOPEN " TO ERROR-FUNCTION
         GO TO ERROR-ROUTINE.
*
    CALL "IFFIND" USING STATUS-IND, IFFIND-SPEC.
    DISPLAY "IFFIND RETCODE: " STATUS-IND.
    IF STATUS-IND NOT EQUAL ZERO
         MOVE "IFFIND " TO ERROR-FUNCTION
         GO TO ERROR-ROUTINE.
*
    CALL "IFCOUNT" USING STATUS-IND, IFCOUNT-COUNT.
    DISPLAY "IFCOUNT RETCODE: " STATUS-IND IFCOUNT-COUNT.
    IF STATUS-IND NOT EQUAL ZERO
         MOVE "IFCOUNT " TO ERROR-FUNCTION
         GO TO ERROR-ROUTINE.
    IF IFCOUNT-COUNT EQUAL ZERO
         GO TO PROCESS-2.

*
    CALL "IFGETC" USING STATUS-IND, IFGET-EDIT-SPEC-2,
         IFGET-NAME-2.
    DISPLAY "IFGETC RETCODE: " STATUS-IND.
    IF STATUS-IND NOT EQUAL ZERO
         MOVE "IFGETC " TO ERROR-FUNCTION
         GO TO ERROR-ROUTINE.
*
    PERFORM IFGET-1 THRU IFGET-END-1 IFCOUNT-COUNT TIMES.
*
 IFGET-1.
*
   CALL "IFGETE" USING STATUS-IND, IFGET-FIELDS-2,
        IFGET-NAME-2.
   DISPLAY "IFGETE RETCODE: " STATUS-IND.
   IF STATUS-IND NOT EQUAL ZERO
        MOVE "IFGETE " TO ERROR-FUNCTION
        GO TO ERROR-ROUTINE.
*
*  PROCESS FIELDS RETRIEVED BY IFGET
*
 IFGET-END-1.
   EXIT.
*
 PROCESS-2.
   CALL "IFFIND" USING STATUS-IND, IFFIND-SPEC-1,
        IFFIND-NAME, IFFIND-DATA, IFFIND-SPEC-2.
   DISPLAY "IFFIND RETCODE: " STATUS-IND.
   IF STATUS-IND NOT EQUAL ZERO
        MOVE "IFFIND " TO ERROR-FUNCTION
        GO TO ERROR-ROUTINE.
*
   CALL "IFCOUNT" USING STATUS-IND, IFCOUNT-COUNT.
```

```
   DISPLAY "IFCOUNT RETCODE: " STATUS-IND IFCOUNT-COUNT.
   IF STATUS-IND NOT EQUAL ZERO
        MOVE "IFCOUNT " TO ERROR-FUNCTION
        GO TO ERROR-ROUTINE.
   IF IFCOUNT-COUNT EQUAL ZERO
        GO TO TERMINATION.

*
   PERFORM IFGET-2 THRU IFGET-END-2 IFCOUNT-COUNT TIMES.
   GO TO TERMINATION.
*
 IFGET-2.
   CALL "IFGET" USING STATUS-IND, IFGET-FIELDS-1,
        IFGET-EDIT-SPEC-1, IFGET-NAME-1, IFGET-DUM-1,
        IFGET-DUM-2, IFGET-SEQUENCE.
   DISPLAY "IFGET RETCODE: " STATUS-IND.
   IF STATUS-IND NOT EQUAL ZERO
        MOVE "IFGET " TO ERROR-FUNCTION
        GO TO ERROR-ROUTINE.

*
*  PROCESS FIELDS RETRIEVED BY IFGET
*
 IFGET-END-2.
   EXIT.
*
 ERROR-ROUTINE.
   DISPLAY "ERROR ENCOUNTERED WITH FUNCTION: " ERROR-FUNCTION
      ", WITH A RETURN CODE OF: " STATUS-IND.
   CALL "IFGERR" USING STATUS-IND M204-ERR-MESSAGE.
   DISPLAY "M204 ERROR MESSAGE = " M204-ERR-MESSAGE.
*
 TERMINATION.
   CALL "IFFNSH" USING STATUS-IND.
   DISPLAY "IFFNSH RETCODE: " STATUS-IND.
   STOP RUN.
/*
```

IFDIAL thread example (z/OS)

This section provides a sample program written in COBOL to run in the IFAM2 environment under z/OS or CMS using the Model 204 Host Language Interface.

The COBOL program illustrates how to establish an IFDIAL connection and how to send SOUL commands and statements to the Model 204 region.

The sample COBOL program in Figure B-8 can be run as shown with the application code embedded in the z/OS job stream.

Note: A similarly structured COBOL program that uses an IFDIAL connection in IFAM2 is compiled and linked as program IFAM2UL to run in CMS using Figure B-5 on page 376 and Figure B-6 on page 376.

Refer to Chapter 2 and to the *Rocket Model 204 Host Language Interface Programming Guide* for more information about IFDIAL threads.

Example of a COBOL program using IFDIAL (z/OS)

Figure B-8 shows an HLI program written in COBOL, which, with the JCL that is shown in the job stream, runs in IFAM2 under an z/OS operating system. Note that this is an example of an IFDIAL application.

Figure B-8. Sample Program: IFDIAL Thread (z/OS)

```
//CPLLKGO EXEC COBUCLG,
//     PARM.COB='LOAD,NOSEQ,APOST',
//     REGION.LKED=200K,
//     PARM.LKED='LIST,LET,SIZE=(192K,100K),MAP',REGION,GO=64K
//COB.SYSIN DD *
  IDENTIFICATION DIVISION.
              PROGRAM-ID. CRAMDIAL.
              ENVIRONMENT DIVISION.
              INPUT-OUTPUT SECTION.
              FILE-CONTROL.
              SELECT INPUT-FILE ASSIGN TO UT-S-INFILE.
              SELECT OUTPUT-FILE ASSIGN TO UT-S-OUTFILE.

              DATA DIVISION.
              FILE SECTION.
  FD          INPUT-FILE LABEL RECORDS ARE OMITTED
                         DATA RECORD IS INPUT-REC.
  FD          OUTPUT-FILE LABEL RECORDS ARE OMITTED
                         DATA RECORD IS OUTPUT-BUFFER.
              01  INPUT-REC     PIC X(80).
WORKING-STORAGE SECTION.
   01  FULL-WORDS COMP SYNC.
              02  ERR           PIC 9(5).
              02  CBL-IND       PIC 9(5) VALUE 2.
              02  READ-ERR      PIC 9(5).
              02  WRITE-ERR     PIC 9(5).
   01  IFAM-ARGUMENT-STRINGS.
              02  FROM-M204     PIC X(256) VALUE SPACES.
              02  TO-M204       PIC X(256) VALUE SPACES.
              02  M204-ERR-MSG  PIC X(80)  VALUE SPACES.
   01 OUT-BUFFER PIC X(256) VALUE SPACES.
   PROCEDURE DIVISION.
   OPEN-FILES-ROUTINE.
   OPEN INPUT INPUT-FILE.
   DISPLAY '%%% TEST OF CRAM/USER LANGUAGE INTERFACE'.
     CALL 'IFDIAL' USING ERR CBL-IND.
     IF ERR NOT = 0
        DISPLAY '%%% IFDIAL FAILED ' ERR
        GO TO END-ROUTINE.
        MVE SPACES TO FROM-M204, TO-M204.
     READ-FILE.
        READ INPUT-FILE INTO TO-M204 AT END GO TO END-ROUTINE.
```

```
   SEND-LINE.
        CALL 'IFWRITE' USING WRITE-ERR, TO-M204.
        IF WRITE-ERR = 2 OR 12
           GO TO PRINT-FILE.
        IF WRITE-ERR = 1
           GO TO READ-FILE.
        DISPLAY '%%% IFWRITE FAILED ' WRITE-ERR.
           GO TO END-ROUTINE.

   PRINT-FILE.
        MOVE SPACES TO FROM-M204.
        CALL 'IFREAD' USING READ-ERR, FROM-M204.
          IF READ-ERR = 1
             WRITE OUT-BUFFER FROM FROM-M204
             GO TO TEST-WRITE-STATUS.
          IF READ-ERR = 12
             GO TO TEST-WRITE-STATUS.

          IF READ-STAT = 2
             WRITE OUT-BUFFER FROM FROM-M204
             GO TO PRINT-FILE.
        DISPLAY '%%% IFREAD FAILED ' READ-ERR.
             GO TO END-ROUTINE.

   TEST-WRITE-STATUS.
        IF WRITE-ERR = 12
           GO TO SEND-LINE
        ELSE
           GO TO READ-FILE.
   END-ROUTINE.
     CLOSE INPUT-FILE.
     CALL 'IFHNGUP' USING ERR.
          IF ERR NOT = 0
     DISPLAY '%%% IFHNGUP FAILED ' ERR
          PERFORM ERROR-ROUTINE.
     STOP RUN.

//LKED.OBDD DSN=M204.OBJECT,DISP=SHR
//LKED.SYSINDD *
INCLUDE OB(IFIF)
//GO.SYSUDUMPDD SYSOUT=A
//GO.SYSOUTDD SYSOUT=A
//GO.OUTFILEDD SYSOUT=A
//GO.INFILEDD *
LOGIN TESTER2
PASWRD2
OPEN FOOTBALL
ACCESS
BEGIN
%A=$USER
%B=$TIME
PRINT %A AT 30 AND %B AT 60
END

BEGIN
```

```
SEARCH: FIND ALL RECORDS FOR WHICH AGE IS GREATER THAN 30
REPORT: FOR EACH RECORD IN SEARCH
    PRINT 'PLAYER' AND NAME AND 'IS' AND AGE -
        AND 'YEARS OLD.'
    PRINT 'HIS TEAMMATES ARE;' AT 5
    REFER: NOTE TEAM
    MATCH: FIND ALL RECORDS FOR WHICH TEAM=VALUE IN REFER
    EVERY: FOR EACH RECORD IN MATCH
    PRINT FIRST.NAME AT 10 AND NAME
    SKIP 1 LINE
    END FOR
END FOR
FINISH: PRINT 'END OF REPORT'
END
/*
```

IFDIAL thread example (CMS)

Figure B-9 and Figure B-10 on page 384 show sample EXECs that can be used to compile and execute an HLI application program in the IFAM2 environment under CMS.

Note: These EXECs might be used with a COBOL program named IFAM2UL (not shown) that uses an IFDIAL connection in a manner similar to the program in Figure B-7 on page 377. See “Sample input to IFAM2UL program” on page 386 and “Sample output from IFAM2UL program” on page 386 that might be used for IFAM2UL.

Example of an EXEC that compiles, links, and loads the program

Figure B-9 shows an EXEC, IFAM2LNK, that compiles, links, and loads an IFAM2 application running under CMS. IFAM2LNK prompts the user for the name of the COBOL IFAM2 program that is to be compiled and linked. It then loads the program.

Note that M204IFAM TXTLIB must be available on an accessed disk to run this EXEC. See Figure B-6 on page 376 for an example of the M204IFAM EXEC that must be accessible on the machine.

Figure B-9. Example of EXEC to compile and link IFAM2 program (CMS)

```
&CONTROL OFF NOMSG
*
* USE THIS EXEC TO COMPILE, LINK, LOAD A COBOL IFAM2 PROGRAM
*
&ERROR &CONTINUE
&IF &INDEX EQ 0 &GOTO -ASKPGM
&BEGTYPE ALL
ERROR: ENTER PARMS ONLY AS PROMPTED.
&END
&EXIT 999
&EXIT 0
```

```
*
-ASKPGM &TYPE ENTER NAME OF THE IFAM2 PROGRAM TO BE COMPILED
LINKED
&READ VARS &PGM
&IF X&PGM EQ X &GOTO -ASKPGM
&IF X&PGM EQ XHX &GOTO -HALT
*

COBOL &PGM
GLOBAL TXTLIB M204IFAM COBOLVS COBLIBVS
LOAD &PGM IFCM
GENMOD &PGM
&TYPE EXEC COMPLETE ... GOODBYE
&EXIT 0
*
-HALT &TYPE EXEC HALTED BY USER REQUEST.
&EXIT 998
```

Example of an EXEC that runs the program

Figure B-10 shows an EXEC that starts an IFAM2 COBOL program running in the user machine.

The EXEC assumes that an IFAM2 COBOL program named IFAM2UL has previously been compiled, linked, and stored as IFAM2UL MODULE. IFAM2UL designates input file INPUT and output file REPORT.

Note that the Model 204 service machine must be up and running and have IODEV=39 available for the IFDIAL communication.

Figure B-10. Example of EXEC to run IFAM2 program (CMS)

```
&CONTROL OFF
*
*  SAMPLE IFAM2 EXEC
*
*  THIS EXEC IS USED TO INITIATE THE START OF
*  AN IFAM2 IFDIAL PROGRAM
*
*  CLEAR EXISTING NON-PERMANENT FILEDEFS
*
FILEDEF * CLEAR
*
*  DEFINE THE FILES NEEDED BY THE COBOL PROGRAM
*
FILEDEF REPORT DISK IFAM2UL REPORT A
FILEDEF INPUT DISK IFAM2UL INPUT A
*
IFAM2UL
*
&TYPE IFAM2 PROGRAM ENDED
&TYPE LOOK FOR OUTPUT IN FILE CALLED 'IFAM2UL' 'REPORT'
```

Example of the M204IFAM EXEC that must be accessible

Figure B-11 on page 385 shows an example of M204IFAM EXEC that must be accessible to the IFAM2 COBOL program running in the user machine. Model 204 calls this EXEC when the IFDIAL call is encountered in the program.

The EXEC provides Model 204 with the ID of the service machine and the desired communication method, IUCV. Input into this EXEC is the channel name that could be used by the EXEC author to determine the proper service machine ID in a multiservice machine environment.

Note that this sample does not use the channel information, but assumes that the service machine ID is M204PROD.

Figure B-11. Example of M204IFAM EXEC for IFAM2 program (CMS)

```
&CONTROL OFF
*
*  M204 SAMPLE IFAM INTERFACE
*
*  THIS EXEC PROCEDURE IS PROVIDED AS A SAMPLE TO AID IN THE
*  CREATION OF A TAILORED IFAM INTERFACE TO MODEL 204 FOR A
*  SPECIFIC INSTALLATION.
*  THE SINGLE PARAMETER TO THE M204IFAM EXEC IS THE CHANNEL
*  NAME TO WHICH A CONNECTION IS BEING REQUESTED BY THE
*  APPLICATION PROGRAM

            &CHANNEL = &1

*  ESTABLISH DEFAULT VALUES

            &TARGET = IDSERV

*  TO ALLOW THE CONNECTION TO BE ATTEMPTED, THE EXEC MUST STACK
*  A SINGLE LINE AND EXIT WITH A RETURN CODE OF ZERO.
*  THE FIRST WORD OF THE LINE (&TARGET) IS THE USERID OF THE
*  VIRTUAL MACHINE WITH WHICH COMMUNICATION IS NEEDED.
*  OPTIONALLY, THE SECOND WORD (TYPE) CAN BE IUCV (USE IUCV
*  ONLY), OR IUCVVMCF (WHICH USES IUCV)
*  IUCVVMCF IS THE DEFAULT.

-ALLOW
            &STACK LIFO &TARGET TYPE
            &EXIT 0

*  TO REFUSE THE CONNECTION ATTEMPT REQUEST, THE EXEC MUST EXIT
*  WITH A NON-ZERO RETURN CODE

-REFUSE
            &EXIT 1
```

Sample input to IFAM2UL program

The following sample input file, INPUT, is a SOUL request that is invoked by the IFAM2UL program (not shown, which is referenced in the CMS EXEC Figure B-11 on page 385.

```
LOGON FRED
FREDPSWD
OPEN CLIENTS
BEGIN
PRINT 'REPORT STARTED'
SKIP 1 LINE
PRINT 'FIRST LINE'
END
CLOSE CLIENTS
LOGOUT
```

Sample output from IFAM2UL program

The following sample output, REPORT, is generated by the IFAM2UL program (not shown), which is referenced in Figure B-11 on page 385. Note that the report output is based on the INPUT file, shown in the previous section “Sample input to IFAM2UL program”.

```
*** M204.0347: PASSWORD
*** M204.0353: FRED FRED LOGIN 90 MAY 17 14.14

*** WELCOME TO PRODUCTION M204 SYSTEM ***

*** M204.0620: FILE CLIENTS OPENED -- NO UPDATES ALLOWED
    REPORT STARTED

FIRST LINE
*** M204.0608: FILE CLOSED: CLIENTS
*** M204.0604: CLOSING DEFAULT, USER MUST ESTABLISH NEW
    DEFAULT
```

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z