
SirFact Reference Manual

(Model 204 Post Hoc Debug Facility)



Sirius Software, Inc.
875 Massachusetts Avenue, Suite 21
Cambridge, MA 02139

Telephone: (617) 876-6677

FAX: (617) 234-1200

E-mail: support@sirius-software.com

World Wide Web: <http://sirius-software.com>

August 5, 2010

© 2010 Sirius Software, Inc.



Proprietary Notices

The following products:

- *SirFact*
- *Sirius Functions*
- *Sirius Mods*
- *UL/SPF*

are proprietary products of Sirius Software, Inc.:

Sirius Software, Inc.
875 Massachusetts Avenue, Suite 21
Cambridge, Massachusetts 02139
USA

Model 204® is a proprietary product of Computer Corporation of America, a wholly-owned subsidiary of Rocket Software, Inc., which owns the trademark:

Rocket Software Corporate Office
M204 Division
275 Grove Street
Suite 3-410
Newton, Massachusetts 02466-2272
USA

SoftSpy™ is a proprietary product of Information Technology Systems:

Information Technology Systems
95 Wells Avenue
Newton, Massachusetts 02459-3216
USA

Contents

Proprietary Notices	iii
Contents	v
Summary of Changes	vii
Sirius Mods Version 7.1	vii
Sirius Mods Version 6.8	vii
Sirius Mods Version 6.7	vii
Sirius Mods Version 6.6	viii
Sirius Mods Version 6.5	viii
Sirius Mods Version 6.4	viii
Sirius Mods Version 6.2	ix
Sirius Mods Version 6.0	ix
Sirius Mods Version 5.6	ix
Sirius Mods Version 5.5	ix
Sirius Mods Version 5.4	x
Chapter 1: Overview	1
Versions and compatibility	1
Related manuals	2
Related products	2
System Requirements	3
Chapter 2: Post Hoc Debugging with SirFact	5
Ad Hoc vs. Post Hoc Debugging	6
Chapter 3: The SIRFACT Command	9
SIRFACT command summary	10
SIRFACT CANCEL	11
SIRFACT DISPLAY	14
SIRFACT DUMP	16
Procedure name substitutions	18
Managing SirFact dumps	19
SIRFACT IGNORE	21
SIRFACT MAXDUMP	22
SIRFACT QUIESCE	23
SIRFACT RESUME	24
SIRFACT RECNDUMP	25
SIRFACT SNAP	26

Chapter 4:	The ASSERT Statement	27
Chapter 5:	The SIRFACT Statement	31
Chapter 6:	The TRACE Statement	35
Chapter 7:	System Parameters	37
	The SIRFACT Parameter	37
	Subsystem procedure enqueues	38
	SIRFACT compilation data collection	38
	The SIRAPSYF Parameter	40
Chapter 8:	SirFact \$Functions	43
	CALLing Sirius \$functions	44
	\$FACT_CMD: Run a command on the SirFact SDAEMON	45
	\$FACT_CONTEXT: Set subroutine context for \$FACT_DATA	46
	\$FACT_DATA: Retrieve data from a SirFact dump	47
	\$FACT_DONE: Terminate a SirFact dump	48
	\$FACT_INIT: Open a SirFact dump	49
	\$FACT_OPTION: Set or get SirFact display options	50
	\$FACT_VNAME_WIDTH: Sets variable name width for \$FACT_DATA	51
	\$TRACE2LIST: Copy wrap-around trace table to \$list	52
Chapter 9:	Other SirFact Facilities in Sirius Mods	55
	Comment-initialized global variables	55
Chapter 10:	The FACT Subsystem	57
	C[ontext] command: Setting program context for data extraction	59
	D[isplay] command: Displaying data from the SirFact dump	60
	Additional syntax for VALUE and LIST	64
	Navigation and display commands	68
	Program function keys	69
	The FACT Print Screen	70
Appendix A:	Date Processing	73
Appendix B:	Terminal MODEL 6 Support	75
Index		77

Summary of Changes

This section describes significant changes to the documentation. Usually, these changes correspond to enhancements made to the underlying product, although they might be simple documentation improvements.

Sirius Mods Version 7.1

The following changes correspond to changes in *SirFact* since version 6.8:

- The SIRFACT DISPLAY MAXDUMP command output now includes a comment that reports the number of *SirFact* dumps taken since the *Model 204* online initialization. See [“SIRFACT DISPLAY” on page 14](#).

Sirius Mods Version 6.8

The following changes correspond to changes in *SirFact* since version 6.7:

- Support for Model 6 terminals ([“Terminal MODEL 6 Support” on page 75](#)).

Sirius Mods Version 6.7

The following changes correspond to changes in *SirFact* since version 6.6:

- The *SirFact* display of objects is enhanced, and some minor changes to existing syntax is included. See the “Objects” discussion in [“Additional syntax for VALUE and LIST” on page 64](#).
- A new *SirFact* \$function (“[\\$FACT_OPTION: Set or get SirFact display options” on page 50](#)) lets you change certain \$FACT_DATA data retrieval options, including case-sensitivity.
- The Assert statement ([“The ASSERT Statement” on page 27](#)) is enhanced:
 - A new `Continue` option lets Assert actions be performed *without* the request being canceled).
 - The `Info` option will now display the value of a %variable in the Assert error message.

Sirius Mods Version 6.6

The following changes correspond to changes in *SirFact* since version 6.5:

The *SirFact* display of XmlDoc contents now uses the Compact form of the *Janus SOAP* XML API Print method.

Sirius Mods Version 6.5

The following changes correspond to changes in *SirFact* since version 6.4:

- You can now invoke many of the *SirFact* \$functions using a User Language CALL statement instead of assigning the function result to a %variable. See [“CALLing Sirius \\$functions” on page 44](#).
- Support is added for displaying the value of some types of object %variables in the *SirFact* dump data. See [“Additional syntax for VALUE and LIST” on page 64](#).
- VERSIONS is added to the INFO data you can dynamically display in the *SirFact* dump data. INFO.VERSIONS retrieves the *Model 204* version, the *Sirius Mods* version, and the *SirFact* internal dump format version in effect when the *SirFact* dump was produced. See [“D\[isplay\] command: Displaying data from the SirFact dump” on page 60](#).

Sirius Mods Version 6.4

This release adds a variety of enhancements that simplify and improve APSY procedure maintenance:

- New SIRFACT parameter options (X'40' and X'80') for APSY maintenance (see [“The SIRFACT Parameter” on page 37](#)).
- New *Sirius Mods* system parameter: SIRAPSYF (see [“The SIRAPSYF Parameter” on page 40](#)).
- New SIRFACT command options: QUIESCE and RESUME (see [“SIRFACT QUIESCE” on page 23](#) and [“SIRFACT RESUME” on page 24](#)).

Sirius Mods Version 6.2

- “\$TRACE2LIST: Copy wrap-around trace table to \$list” on page 52.
- “The TRACE Statement” on page 35.
- New +D and +M substitution strings in SIRFACT DUMP command.
- SIRFACT available as operator command.
- REQUEST TOO LONG errors captured under certain circumstances.

Sirius Mods Version 6.0

- Add documentation for the new SIRFACT RECNDUMP command.
- Add documentation for the new SIRFACT SNAP command.
- Add documentation for message number parameters on the SIRFACT DUMP command.
- Add documentation for the new DISPLAY RIN, RON and FIELD commands in the FACT subsystem.

Sirius Mods Version 5.6

- Add documentation for the new SIRFACT parameter bits to trap FOR RECORD NUMBER (FRN) errors.

Sirius Mods Version 5.5

Following are features which were also released as zaps to one or more versions prior to version 5.4:

- Comment-initialized globals

Sirius Mods Version 5.4

This is the initial release of *SirFact*.

CHAPTER 1 *Overview*

1.1 Versions and compatibility

SirFact is a post hoc debugging tool made up of two distinct components.

First, there is a collection of object code enhancements to the *Model 204* database-engine nucleus. These enhancements are distributed as components of the *Sirius Mods* and make up a collection of products including those in the Janus family. The *Sirius Mods* include many non-debugging related products such as *Fast/Backup*, *Fast/Reload*, *Janus TCP/IP Base*, and *Janus Web Server*. No other *Sirius Mods* products are required to run *SirFact*.

The second component of *SirFact* is a collection of *Model 204* procedure files that contain User Language, documentation and assorted other data. These *Model 204* procedures files are distributed as components of the User Language Structured Programming Facility also known as *UL/SPF*. *UL/SPF* includes a SIRFACT file which contains code useful for looking at *SirFact* dumps. *UL/SPF* also includes the definitions for the FACT subsystem which is useful for looking at *SirFact* dumps. *UL/SPF* also includes files that are components of non-*SirFact* related products such as *SirPro*, *SirScan* and *SirMon*. No other *UL/SPF* products are required to run *SirFact*, though *SirPro* makes it more convenient to look at a list of dumps and to select one to be viewed from the FACT subsystem.

To install *SirFact* both the *Sirius Mods* and *UL/SPF* must be installed. When the *Sirius Mods* are installed, all other products owned by the installing site that are part of the *Sirius Mods* will also be installed. Similarly, when *UL/SPF* is installed, all other products owned by the installing site that are part of *UL/SPF* will be installed.

Because the *Sirius Mods* and *UL/SPF* have somewhat different release cycles, the version numbers for these two components will often differ in a distribution. For example, version 5.3 of the *Sirius Mods* might be shipped with version 5.0 of *UL/SPF*. All the products in *UL/SPF* depend on certain features being present in the version of the *Sirius Mods* that is installed in the *Model 204* load module under which *UL/SPF* is running. This implies that the *Sirius Mods* must be installed for any *UL/SPF* component to operate correctly. Conversely, the *Sirius Mods* do not depend on any features being present in *UL/SPF* or even on the presence of *UL/SPF*.

Sirius Software has a strong commitment to backward compatibility with the *Sirius Mods*. What this means, is that any User Language application that uses the *Sirius Mods* (including *UL/SPF*) will run correctly on subsequent versions of the *Sirius Mods*. It is always possible to upgrade the *Sirius Mods* without having to worry about upgrading *UL/SPF*. This is not to say that this is always a good idea, only that it is possible and that the installed version of *UL/SPF* products will continue to run as they had before the *Sirius Mods* upgrade.

While *SirFact* has a *UL/SPF* component most of the critical code is actually in the *Sirius Mods* - object code enhancements to the *Model 204* nucleus. The *UL/SPF* component of *SirFact* consists mostly of the FACT subsystem, which is a user interface for scanning *SirFact* dumps. Because of this, the version number of *SirFact* is generally considered to be the version of the *Sirius Mods* in which it is contained. *SirFact* was first available in version 5.4 of the *Sirius Mods* so the first version of *SirFact* was actually called version 5.4. This document, the ***SirFact Reference Manual***, assumes that a site is running *Sirius Mods* version 5.4 or later and has installed *UL/SPF* version 5.4 or later. Any documentation that requires a later version of the *Sirius Mods* or *UL/SPF* will be clearly marked to indicate this. For example, a \$function that is only available in versions 5.5 and later of the *Sirius Mods* will have a sentence such as “This function requires version 5.5 or later of the *Sirius Mods*” in its documentation. If a feature, \$function, command, or parameter is not indicated as requiring any specific version of the *Sirius Mods*, it can be assumed that it is available as documented in all versions of the *SirFact*; that is, all versions since version 5.4 of the *Sirius Mods* and version 5.4 of *UL/SPF*.

1.2 Related manuals

As mentioned in “[Versions and compatibility](#)” on page 1, *SirFact* requires the installation of both the *Sirius Mods* and *UL/SPF*. As such, the person responsible for the installation of *SirFact* should refer to the ***Sirius Mods Installation Guide*** and the ***UL/SPF Installation and Maintenance Guide***. Also, the ***Sirius Messages Manual*** contains documentation on *SirFact* error messages and so might be useful to system programmers or DBAs as well as installers.

There are a number of *Sirius* \$functions which you are authorized to use along with *SirFact*, including procedure processing \$functions and *\$list processing \$functions*. These are documented in the ***Sirius Functions Reference Manual***. The ***Sirius Functions Reference Manual*** does **not** contain documentation for the functions specific to *SirFact*. These functions are documented only in this manual in “[SirFact \\$Functions](#)” on page 43.

The FACT subsystem, the subsystem used for looking at *SirFact* dumps, can be invoked from *SirPro*. While this capability is documented in this manual in “[The FACT Subsystem](#)” on page 57, the ***SirPro User's Guide*** documents the layout of and other useful facilities available in *SirPro*.

1.3 Related products

The FACT subsystem, the subsystem used for looking at *SirFact* dumps, can be invoked from *SirPro*. *SirPro* also provides a nice user interface for scanning lists of dumps and programs.

SirFact does not address issues with tracking and distributing fixes to bugs once they are written. *SirLib* might prove useful to handle change control and fix distribution issues related to post hoc (and even ad hoc) debugging.

SirFact provides a powerful post hoc debugging facility. It is always better, however, to catch bugs before they go into unit-testing or production. As such, ad hoc (interactive) debugging facilities are useful in reducing the number of dumps that get into production. The *Janus Debugger* and the *Sirius Debugger* are interactive debugging facilities that are useful for debugging code while it is being developed. *SirScan* can also be useful for debugging certain applications, especially those that are not associated with a terminal; for example, Janus or Connect* bugs.

1.4 System Requirements

SirFact requires the following components to run:

- Mainframe operating systems:
 - MVS/SP Version 1.3 or later (including MVS/XA and MVS/ESA) or
 - CMS (releases currently supported by IBM) running under
 - VM/ESA or
 - z/VM
- *Model 204* Version 5 Release 1.0 or later

Post Hoc Debugging with SirFact

Debugging is the process of eliminating errors in software. Debugging essentially consists of three steps :

1. Determining an error has occurred.
2. Determining the cause of the error.
3. Correcting the cause of the error.

The last item is essentially a programming task and so is essentially beyond the scope of debugging tools and probably any other kinds of tools. Understanding the reason something is an error, the logic flaws that caused the error and how the logic needs to be corrected require **understanding** so are essentially human activities. It is not likely this process can be significantly improved by automation any time in the near future.

One might also consider managing and distributing the fixes to errors to be part of the debugging process. Whether or not this is accurate, fix management and distribution are different enough in their nature that they are rarely handled by debugging facilities and are more typically handled by change control facilities such as *SirLib* which is available for *Model 204*.

Software errors are typically referred to as bugs so that removing errors from software is called debugging. A bug is typically a problem with software that results in a display of incorrect data to the end-user or one that causes premature termination of a program. There is also a class of bugs that can cause poor software performance. In any case, if uncorrected, most bugs will ultimately be noticed by end-users as incorrect data, abnormal termination of a system, or poor performance.

Software logic errors can cause immediately noticed problems or they can cause errors long after the original logic error. For example, a bug in a piece of code might cause a global to be incorrectly set. This incorrectly set global might cause another piece of code to store some invalid data in the database that months later might be loaded by a different piece of code which then might terminate abnormally because of the invalid data or display incorrect data on the end-user's screen. Clearly, in this sort of situation, especially if the incorrect data might have been stored by any one of dozens of procedures, determining the cause of the problem can be very difficult.

This example illustrates the point that the further from the original logic error one catches the error the more difficult it is to determine the cause of the error. Because of this, one of the goals of debugging is to catch errors as early as possible. *SirFact* provides the "SIRFACT CANCEL" command (["The SIRFACT Command" on page 9](#)) and the "ASSERT" (["The ASSERT Statement" on page 27](#)) and "SIRFACT" (["The SIRFACT Statement" on page 31](#)) statements to facilitate catching errors earlier rather than later.

A second goal of debugging is to collect as much information as possible when an error occurs. A classic example of the antithesis of this is the *Model 204* message **M204.0553 SUBSCRIPT RANGE ERROR**. This message causes request cancellation but provides virtually no information as to the cause of the problem. Until recently this message didn't even include the name of the array to which the invalid subscript applied; fortunately now it does. Nevertheless, there are still many key pieces of information missing, such as the line of code the error occurred on, the value of the array's subscript, the values of other variables or fields from which the subscript was derived, etc.. *SirFact* provides the "SIRFACT" system parameter ("[The SIRFACT Parameter](#)" on page 37) and the "SIRFACT MAXDUMP" and "SIRFACT DUMP" commands ("[The SIRFACT Command](#)" on page 9) to collect as much information about application errors as possible. A tremendous amount of information about the application at the time of an error is collected in *SirFact* dumps, under control of these commands. The data in these dumps can then be viewed via *SirFact* \$functions ("[SirFact \\$Functions](#)" on page 43) or a ready-to-use application subsystem called FACT ("[The FACT Subsystem](#)" on page 57).

2.1 Ad Hoc vs. Post Hoc Debugging

When most people think of debugging tools they think of tools used by a programmer during the development process or perhaps when a programmer is trying to reproduce in a test environment a problem that occurred in a production environment. These types of tools are typically interactive and allow things like setting of breakpoints, examination and "manual" modification of values of variables and tracing of code paths. Because the user of such interactive debugging tools must be familiar with the code being debugged and because an application being debugged by such a tool must run "inside" the debugging environment, these tools are generally only useful when a programmer is running the application. Because of the interactive nature of these types of debugging tools, they are sometimes referred to as "ad hoc" debugging tools. The *Janus Debugger* and *Sirius Debugger* are full-fledged ad hoc debugging tools available for *Model 204*.

As desirable as it is to catch all errors during the development process, this is simply not possible. All but the simplest code has just too many possible combinations of user and external inputs, database values and environmental variables for all possible combinations to be tested. Often a bug can only be induced by specific combinations of all these variables. Because of this bugs can and will be detected without a programmer present and so outside the environment of an ad hoc debugger. Even worse, some of these bugs will not be reproducible in a test environment so that a programmer could use an ad hoc debugger to attack the problem.

Very often, a user will not remember the exact sequence of inputs she entered to cause an error or perhaps the error was caused by a combination of user inputs and environmental problems beyond the view of the user. Often, the combination of factors that caused a bug will not be understood until the actual cause of the bug is understood. Occasionally, with timing related bugs, even understanding all the factors required to cause a bug to happen might not be sufficient to consistently reproduce the problem.

Ad hoc debuggers are basically useless for problems that occur in production or unit-test away from the watchful gaze of a programmer. Instead, a different debugging tool is needed for these kinds of problems, namely a post hoc debugging tool. A post hoc debugging tool is useful in solving problems that occur outside of development because

1. It contains facilities to trap errors earlier rather than later.
2. It collects and stores as much information as possible at the time of an error.

The only real full-fledged post hoc debugging tool available in the *Model 204* environment is *SirFact*.

Even in the later stages of development a post hoc debugging facility might be preferable to an ad hoc debugger. This is because a programmer might wish to quickly go through many code paths without an intrusive ad hoc debugger in the way. Because they run in production systems post hoc debugging tools must be extremely unobtrusive. In any case, a programmer might be willing to pay a price in getting slightly less interactive debugging capabilities for the benefit of having the debugger “out of the way” most of the time.

The SIRFACT Command

The SIRFACT command has a variety of subcommands that serve a grab-bag of functions. All SIRFACT commands are global, that is system-wide, in nature and so require system manager or system administrator privileges to execute. Under *Sirius Mods* version 6.2 and later, SIRFACT commands can also be issued as operator commands, that is, on the Online virtual console under VM, or as the response to the HALT message under OS/390.

Some SIRFACT subcommands make use of the wildcard characters asterisk (*), question mark (?), and double quotation mark ("):

- ? Represents any single character.
- * Represents any string of characters.
- " The escape flag that allows special characters to bypass wildcard translation.

The following examples illustrate the use of the wildcard characters:

- This command requests a display of all the SIRFACT CANCEL subcommands in effect for \$functions that begin with the letters "\$LIST":

```
SIRFACT DISPLAY CAN $LIST*
```

- This command indicates that a *SirFact* dump should be created for any request cancellation errors that occur in a subsystem with a name that is seven characters long and ends in the letters "CUST":

```
SIRFACT DUMP DUMPROC DUMP.+T SUBSYS ???CUST
```

- This command indicates that a *SirFact* dump should be created for any request cancellation errors that occur for user ?WEIRD. The double-quote (") is required to prevent the question mark (?) from being interpreted as a wildcard character:

```
SIRFACT DUMP DUMPROC DUMP.+T USER "?WEIRD
```

3.1 SIRFACT command summary

The following table shows the SIRFACT subcommands and their functions.

All subcommands are prefixed with “SIRFACT” (for example, SIRFACT CANCEL, SIRFACT MAXDUMP).

CANCEL	Indicates which return codes from \$functions should result in request cancellation. Can be abbreviated “CAN”.
DISPLAY	Shows the currently active SIRFACT subcommands. Can be abbreviated “DISP.”.
DUMP	Indicates which request cancellations should cause a <i>SirFact</i> dump to be taken and where the dump is to go.
IGNORE	Indicates which request cancellation error messages are not to produce <i>SirFact</i> dumps. Can be abbreviated “IGN.”.
MAXDUMP	Places limits on the number of <i>SirFact</i> dumps that will be taken. The default system limit is 0 so a SIRFACT MAXDUMP must be issued to get any SIRFACT dumps. Can be abbreviated “MAXD”.
NOCANCEL	Indicates that certain return codes from \$functions should not result in request cancellation. Cancels out the effect of a SIRFACT CANCEL command. Can be abbreviated “NOCAN”.
NODUMP	Indicates that certain request cancellations should not result in <i>SirFact</i> dumps being produced. Cancels out the effect of a SIRFACT DUMP command.
NOIGNORE	Indicates that certain request cancellation error messages are to produce <i>SirFact</i> dumps. Cancels out the effect of a SIRFACT IGNORE command. Can be abbreviated “NOIGN”.
QUIESCE	Facilitates updates of APSY subsystem procedures while the subsystem is up and in use.
RECNDUMP	Establishes the number of record numbers from each found set or LIST to be dumped. Can be abbreviated “RECND.”.
RESUME	Stops the effect of a SIRFACT QUIESCE command and returns the subsystem to normal operation.
SNAP	Requests that a <i>SirFact</i> dump be taken for another thread.

3.2 SIRFACT CANCEL

SIRFACT CANCEL function returncodes

SIRFACT CANCEL command syntax

The SIRFACT CANCEL subcommand requires a parameter that indicates the \$function to which it applies, followed by return codes that should be intercepted and cause request cancellation when returned by the function. The subcommand “CANCEL” can be abbreviated to “CAN”.

- function** The name of the function to which the command applies. This name cannot contain wildcard characters.
- returncodes** A blank delimited set of return codes that will be intercepted and cause request cancellation.

For example,

```
SIRFACT CANCEL $SETG 1
```

would cause cancellation of any request that gets a return code of 1 from \$SETG.

```
SIRFACT CAN $LISTADD -3 -5 -6 -7
```

would cause cancellation of any request that gets a return code of -3, -5 -6 or -7 from \$LISTADD.

The rationale for SIRFACT CANCEL is that certain return codes from certain \$functions indicate programming errors or severe environmental problems (return code 1 from \$SETG, for example), so these codes should result in immediate request cancellation, even in adhoc or development procedures. For example, a return code of -6 from \$LISTLOC, indicating an invalid \$list identifier, suggests a severe problem in a program. Even in an adhoc procedure, there is little reason to go on after such an error.

There is a distinction, however, between cancelling the request and creating a *SirFact* dump for the error. A dump would generally not be desirable for errors in adhoc or development procedure errors. Because of this, the creation of dumps is controlled by a different SIRFACT subcommand, the SIRFACT DUMP subcommand (“[SIRFACT DUMP](#)” on page 16). This subcommand limits the creation of dumps to specific subsystems, procedures, or users.

It is *strongly* recommended that the collection of SIRFACT CANCEL commands in the development and test regions match the collection in production. Otherwise, things that work in development might not work in test or production and vice versa. It can't be overemphasized that SIRFACT CANCEL should be used for return codes that indicate *programming errors* or *severe environmental* problems that are as invalid in development as in production. In the rare cases where such a return code might be

considered “normal” or at least acceptable in a chunk of code, the SIRFACT *statement* (“[The SIRFACT Statement](#)” on page 31) can be used to temporarily disable *SirFact* \$function error trapping for a specific user.

A SIRFACT CAN can be disabled with a SIRFACT NOCAN command for the same function. If multiple SIRFACT CANCELs (or NOCANCELs) are issued for the same \$function, only the last one is used. So the following would only trap return codes of -6 from \$LISTLOC:

```
SIRFACT CAN $LISTLOC -5
SIRFACT CAN $LISTLOC -6
```

To trap return codes of -5 and -6 from \$LISTLOC, you must issue

```
SIRFACT CAN $LISTLOC -5 -6
```

The implementation of SIRFACT CANCEL is very efficient and should cause **no** measurable performance degradation in the execution of functions for which return codes are being trapped. Performance of the *SirFact* error trapping is certainly significantly more efficient than testing for the same errors using User Language.

Whether or not a *SirFact* dump is produced for an error trapped by the SIRFACT CANCEL command, these trapped errors are *always* accompanied by a message indicating the procedure and line number in which the error occurred, as well as the name of the function causing the error and the function arguments. *SirFact* will use any one of several mechanisms to collect the procedure and line number information, each of them having some cost in either QTBL, STBL, VTBL, or CCATEMP space. For a discussion of the costs and tradeoffs of the various approaches, see “[The SIRFACT Parameter](#)” on page 37.

The following is a list of “reasonable” SIRFACT CANCEL commands for most Onlines:

```
SIRFACT CAN $BLDPROC          1
SIRFACT CAN $DATECHG         *
SIRFACT CAN $DATECNV         *
SIRFACT CAN $DATEDIF        99999999
SIRFACT CAN $EDIT            ' ' '#'
SIRFACT CAN $INCRG           1 2
SIRFACT CAN $LSTPROC         4
SIRFACT CAN $RDPROC          2
SIRFACT CAN $SETG            1
SIRFACT CAN $UNBIN           ' '
SIRFACT CAN $UNFLOAT         ' '
SIRFACT CAN $UNPACK          ' '
SIRFACT CAN $LISTADD         -3 -5 -6 -7
SIRFACT CAN $LISTADDI        -3 -5 -6 -8
SIRFACT CAN $LISTCNT         -5 -6
SIRFACT CAN $LISTCPY         -3 -5 -6
SIRFACT CAN $LISTDEL         -5
SIRFACT CAN $LISTFIND        -5 -6 -7 -8
SIRFACT CAN $LISTILN         -5 -6 -7
```

SIRFACT CAN \$LISTIMG	-5 -6 -8
SIRFACT CAN \$LISTINF	-5 -6 -7 -9
SIRFACT CAN \$LISTINFI	-5 -6 -7 -8
SIRFACT CAN \$LISTINS	-3 -5 -6 -7 -9
SIRFACT CAN \$LISTINSI	-3 -5 -6 -8 -9
SIRFACT CAN \$LISTLOC	-3 -5 -6 -7 -9
SIRFACT CAN \$LISTLUP	-3 -5 -6 -7 -9
SIRFACT CAN \$LISTMOVE	-5 -6
SIRFACT CAN \$LISTOVL	-5 -6 -7 -9
SIRFACT CAN \$LISTREM	-5 -6 -7
SIRFACT CAN \$LISTRST	-13
SIRFACT CAN \$LISTSAV	-3 -5 -13
SIRFACT CAN \$LISTSAVE	-3 -5 -13
SIRFACT CAN \$LISTSAVL	-3 -5 -6
SIRFACT CAN \$LISTSRT	-3 -5 -6 -10 -12
SIRFACT CAN \$LISTSORT	-3 -5 -6 -10 -12
SIRFACT CAN \$LISTSUB	-3 -5 -6 -7 -9
SIRFACT CAN \$LIST_GLOBAL	-3 -14 -15
SIRFACT CAN \$LIST_CONV_ITEM	-5 -6 -7
SIRFACT CAN \$LIST_COPY_ITEMS	-3 -5 -6 -7 -8
SIRFACT CAN \$LIST_DIFF_ITEM	-3 -5 -6 -7

As might be apparent from these examples, there are a few \$functions for which the return values are special-cased. These are:

- \$DATECHG** On errors, this function will return a number of asterisks (*) equal to the length of the second parameter (the output format). *SirFact* interprets a SIRFACT CANCEL \$DATECHG '*' to require request cancellation for any \$DATECHG call that returns one or more asterisks.
- \$DATECNV** On errors, this function will return a number of asterisks (*) equal to the length of the first parameter (the input format). *SirFact* interprets a SIRFACT CANCEL \$DATECNV '*' to require request cancellation for any \$DATECNV call that returns one or more asterisks.
- \$EDIT** On errors, this function will return a number of hashes (#) equal to the length of the second parameter (the edit mask). *SirFact* interprets a SIRFACT CANCEL \$EDIT '#' to require request cancellation for any \$EDIT call that returns one or more hashes.
- \$LISTINF** \$LISTINF always returns a string value. To indicate an error, \$LISTINF returns a string containing '-5', '-6', '-7' or '-9'. Yet, in theory at least, these values could also be reasonable values of \$list items. SIRFACT CANCEL provides a work-around for this design flaw in \$LISTINF, because SIRFACT CANCEL will **not** cause request cancellation when a cancellation value is actually a value in a \$list item. SIRFACT CANCEL will only cause cancellation when the return value is truly the result of an error.

It is strongly recommended that you set SIRFACT CANCEL as aggressively as possible. If a SIRFACT CANCEL should cause a problem with “dirty” use of \$functions, the preferred solution is to clean up the use of the \$functions. The next best solution is to wrap the “dirty” function calls in a SIRFACT OFF/SIRFACT ON bracket. Eliminating the SIRFACT CANCEL for a value is the least preferable solution, because:

1. It reduces the benefit of SIRFACT CANCEL to encourage good programming practices.
2. It eliminates the ability of SIRFACT CANCEL to catch unintentional misuses of \$functions.

3.3 SIRFACT DISPLAY

`SIRFACT DISPLAY [subcom [subset]]`

SIRFACT DISPLAY command syntax

The SIRFACT DISPLAY subcommand displays many of the SIRFACT subcommands currently in effect. SIRFACT DISPLAY with no *subcom* displays all SIRFACT commands from the subcommand classes listed below that are currently in effect. The subcommand “DISPLAY” can be abbreviated to “DISP”.

subcom The name of the class of subcommands to which the display is to apply. Valid values for *subcom* are:

CANCEL Displays active SIRFACT CANCEL commands. Can be abbreviated CAN.

CANCEL can be followed by a *subset* value that can contain wildcards indicating the \$functions for which SIRFACT CANCEL settings are to be displayed. SIRFACT NOCANCEL commands are not displayed because these simply eliminate the SIRFACT CANCELs for the specified \$functions.

DUMP Displays active SIRFACT DUMP and NODUMP commands.

IGNORE Displays active SIRFACT IGNORE commands. Can be abbreviated IGN.

IGNORE can be followed by a *subset* that can contain wildcards indicating the messages for which SIRFACT IGNORE settings are to be displayed. SIRFACT NOIGNORE commands are not displayed because these

simply eliminate the SIRFACT IGNOREs for the specified messages.

MAXDUMP Displays active SIRFACT MAXDUMP commands. Can be abbreviated MAXD.

MAXDUMP can be followed by a *subset* indicating the type of limit to be displayed. This can be “TOTAL” or “USER”.

As of *Sirius Mods* version 7.1, a SIRFACT DISPLAY MAXDUMP output line is followed by a comment line that reports the number of *SirFact* dumps taken since the *Model 204* Online initialization. For example:

```
SIRFACT DISPLAY MAXDUMP
SIRFACT MAXDUMP TOTAL 20
* 11 Number of dumps taken
```

RECNDUMP Displays active SIRFACT RECNDUMP command. Can be abbreviated RECND.

subset A string indicating which values of the indicated subcommand should be displayed. For SIRFACT DISPLAY CANCEL and SIRFACT DISPLAY IGNORE, *subset* can contain wildcard characters. For SIRFACT DISPLAY MAXDUMP, it must be a specific limit type.

subset is ignored for SIRFACT DISPLAY DUMP and SIRFACT DISPLAY RECNDUMP.

The display format for the SIRFACT DISPLAY command repeats the format of the SIRFACT commands that are used to set SIRFACT controls. As such, they can be captured (for example with a USE statement) for subsequent reuse. If no SIRFACT controls are in effect at all or for the specific subcommand and subset requested, SIRFACT DISPLAY simply returns without issuing any messages.

Examples of valid SIRFACT DISPLAY commands follow:

```
SIRFACT DISPLAY
SIRFACT DISPLAY CANCEL
SIRFACT DISPLAY CAN $LIST*
SIRFACT DISPLAY CANCEL $EDIT
SIRFACT DISPLAY DUMP
SIRFACT DISPLAY IGNORE
SIRFACT DISPLAY IGN MSIR.*
SIRFACT DISPLAY MAXDUMP
SIRFACT DISPLAY MAXD USER
```

3.4 SIRFACT DUMP

`SIRFACT DUMP procfile procname [criteria]`

SIRFACT DUMP command syntax

The SIRFACT DUMP subcommand indicates where *SirFact* dumps are to be taken and for which environments dumps are to be taken. All *SirFact* dumps are taken to procedures.

Information contained in a *SirFact* dump is easily viewed using the FACT subsystem, described in [“The FACT Subsystem” on page 57](#). You can also use a set of \$functions, described in [“SirFact \\$Functions” on page 43](#).

- procfile** The name of the procedure file to which dumps are to be taken. If the procedure file is not open at the time a *SirFact* dump is to be taken, the procedure file is opened with the privileges required to create the dump (X'1239') and then closed after the dump. If the procedure file is already open it *must* be open with the privileges required to take the dump, or the dump will not be taken.
- procname** The name of the procedure to be created in *procfile* for the *SirFact* dump. *Procname* can be a simple literal string or it can contain special substitution strings consisting of a plus sign (+) followed by a single character indicating what should be substituted into the dump procedure name. Because *SirFact* will never overwrite a procedure with the required procedure name, the use of substitution strings in *procname* is strongly encouraged. Procedure name substitution strings are described in greater detail in [“Procedure name substitutions” on page 18](#).
- criteria** A list of blank-delimited selection criteria that indicate to which “environments” the SIRFACT DUMP command is to apply. All these environments can also be specified on a SIRFACT NODUMP command and have the same meaning for that command. If *criteria* is not specified, the SIRFACT DUMP (or SIRFACT NODUMP) command applies to any request that receives a request cancellation message that is not being “ignored” by *SirFact* ([“SIRFACT IGNORE” on page 21](#)). Valid selection criteria are
- FILE** The procedure file to which the DUMP rule applies. The file name can contain wildcard characters. *FILE* refers to the file in which the outer procedure being evaluated resides, whether or not the INCLUDE was done in group context.
- IODEV** The IODEV number to which the DUMP rule applies.

msg number The *Model 204* message number to which the DUMP rule applies. This message number must start with either a “M204.” for standard *Model 204* error messages, “MSIR.” for *Sirius Mods* error messages or “USER.” for site-specific error messages. The message number could also be “SNAP” which means that the rule applies to *SirFact* dumps taken as a result of SIRFACT SNAP commands. For example:

```
SIRFACT DUMP SUBSERR -
      OUCH.+P.+T M204.0553
```

would send all dumps for subscript range errors to file SUBSERR and

```
SIRFACT DUMP FACTSNAP -
      OUCH.+P.+T SNAP
```

would send all dumps resulting from SIRFACT SNAP commands to file FACTSNAP.

PROC The procedure name to which the DUMP rule applies. The procedure name can contain wildcard characters. *PROC* refers to the outer procedure being evaluated, not the INCLUDE'd procedure in which the error occurred.

SUBSYS The APSY subsystem to which the DUMP rule applies. The subsystem can contain wildcard characters.

USER The userid to which the DUMP rule applies. The userid can contain wildcard characters.

For a DUMP rule to apply to an error being handled by *SirFact*, the error time environment must match **all** criteria specified in the SIRFACT DUMP command.

SirFact dumps are taken for any request cancellation message that is not being “ignored” (“SIRFACT IGNORE” on page 21) if the following criteria are met:

1. SIRFACT MAXDUMP has been set.
2. None of the SIRFACT MAXDUMP limits has been exceeded.
3. There is a SIRFACT DUMP command for which the current thread matches all criteria.
4. The dump procedure file as determined by the appropriate SIRFACT DUMP command is either already open with the correct privileges (any superset of X'1239') or can be opened with the correct privileges.

5. The output procedure as determined by the appropriate SIRFACT DUMP command does not already exist.

Under *Sirius Mods* version 6.2 and later, dumps will be taken for request cancellations resulting from “M204.1332: LONG REQUEST” messages even though technically this message is not a request cancellation message. Note that the dump will only be taken if the user is not given a chance to continue with a “DO YOU REALLY WANT TO CONTINUE?” message, that is if the PROMPT X'10' bit is set or if the request is running in an APSY subsystem or the request is running on a batch thread. Before *Sirius Mods* version 6.2 no *SirFact* dumps would ever be taken for “M204.1332: LONG REQUEST” messages.

Of course, *SirFact* dumps can be terminated if there is insufficient space in the dump procedure file to hold the dump. In this case, the partial dump will be deleted because it is essentially useless.

3.4.1 Procedure name substitutions

The dump procedure name specified in the SIRFACT DUMP command can contain the special character plus (+) followed by a single character indicating a substitution string. The following characters will cause the following substitutions to take place:

+ A literal plus.

1-9

A single digit sequence number as in “+2” or “+6”. *SirFact* tries sequence numbers from 0 through the indicated number until it finds one that creates a procedure name that doesn't already exist in the target proc file. Note that this behavior has changed under *Sirius Mods* version 6.2. Under older versions 1, 2 and 3 meant 0-9, 00-99 and 000-999 respectively and 4-9 were not available as substitution characters.

D The date that the error happened in “YYYYMMDD” format. This substitution character is only available in *Sirius Mods* version 6.1 and later.

F The file (not group) that contains the evaluating outer proc.

J The jobname of the job in which the error occurred.

M The number of the message that caused the request cancellation or the word SNAP if the dump was caused by a SIRFACT SNAP command. The substituted message number does not include the period so, for example, if the request cancellation was caused by a MSIR.0491, a “+M” would be replaced by a “MSIR0491”. This substitution character is only available in *Sirius Mods* version 6.2 and later.

P The name of the evaluating outer proc.

S The name of the current subsystem.

T The date and time that the error happened in “YYYYMMDDHHMISS” format.

U The userid that encountered the error.

Any other characters following a plus are simply dropped.

For example, if the SIRFACT DUMP command is

```
SIRFACT DUMP FACTDUMP DUMP.+P.+T
```

and a *SirFact* trapped error happens in procedure PRE.BADLY.CODED at 14:22:31 on May 12, 2001, the dump procedure name would be DUMP.PRE.BADLY.CODED.20010512142231. If the SIRFACT DUMP command is

```
SIRFACT DUMP FACTDUMP DUMP.+J.+2
```

and a *SirFact* trapped error happens in job DEVONLN, *SirFact* would try to take the dump to procedure “DUMP.DEVONLN.00”. If that procedure already exists, it will try taking the dump to “DUMP.DEVONLN.01”, and so on. If *SirFact* gets all the way to “DUMP.DEVONLN.99” without finding a proc that doesn't already exist, the dump is not taken.

Because most of the substitution characters in a dump are likely to occur in multiple dumps it is recommended that most SIRFACT DUMP rules have a procedure name that contains a “+1”, “+2”, “+3” or a “+T”.

3.4.2 Managing SirFact dumps

Multiple “SIRFACT DUMP” commands might apply to a particular situation. For example, if the following two commands are in effect:

```
SIRFACT DUMP DUMPCUST DUMP-+P-+2 SUBSYS CUSTOMER
SIRFACT DUMP JUNKDUMP DUMP-+S-+P PROC *JUNK
```

and a request cancellation occurs in procedure PCUST-MISCJUNK in subsystem CUSTOMER, both SIRFACT DUMP commands would seem applicable. In such cases, *SirFact* will always use the last applicable SIRFACT DUMP rule so that in this example, the *SirFact* dump would be taken to procedure DUMP-CUSTOMER-PCUST-MISCJUNK in file JUNKDUMP. Because *SirFact* uses the last applicable SIRFACT DUMP rule, these rules should be specified in most general first, most specific last order. For example

```
SIRFACT DUMP SIRFACTD DMP.+P-+T
SIRFACT DUMP INVDUMP DMP.+P-+T SUBSYS INVENTORY
SIRFACT DUMP WIDGETD DMP.+P-+T SUBSYS INVENTORY -
PROC *WIDGET
```

requests that most dumps would go to file SIRFACTD but dumps for subsystem INVENTORY would go to file INV DUMP except for those in procedures the end in “WIDGET” which would go to file WIDGETD.

If a SIRFACT DUMP command has criteria that are a superset of previous SIRFACT DUMP commands, those previous criteria are discarded. For example, in

```
SIRFACT DUMP JUNKDUMP +P.+1.DUMP SUBSYS INVENTORY
SIRFACT DUMP DUMPDATA +P.+1.DUMP SUBSYS INVENTORY
SIRFACT DUMP IDUMP +P.+1.DUMP SUBSYS I*
```

the last SIRFACT DUMP command would always supersede the previous two so when *SirFact* encounters that command, it discards the previous two, so that a SIRFACT DISPLAY DUMP would only show the last SIRFACT DUMP command. An unqualified SIRFACT DUMP command, that is, one with just the procedure file and name, supersedes **all** previous SIRFACT DUMP commands.

SirFact dumps are synchronous on a thread level but asynchronous on a system level. This means that the user or thread for which a *SirFact* dump is being taken will not be able to do anything while the dump is being taken but other users or threads will continue to run. This is different from CCASNAPs (and SYSMDUMPs) which stop the entire Online while they are being taken. In any case, even very large *SirFact* dumps should only take a few seconds to complete so should not have a severe negative impact on performance for either the affected user or any other user, for that matter.

SirFact dumps do **not** log roll-forward information for the dump procedures even if the procedure file to which the dump is going is configured to log roll-forward information (the FRCVOPT X'04' bit is not set). This will not interfere with the roll-forward logging for any other data going to the dump procedure file and will not cause any recovery problems, it simply means that if the system crashes after a *SirFact* dump and the procedure file is rolled back to a checkpoint before the *SirFact* dump in recovery, the dump will be lost.

Checkpoint logging is performed on the dump procedure file as indicated by FRCVOPT. If no checkpoint logging is being performed on the dump procedure file (FRCVOPT X'20' bit set) then there is no danger of losing a *SirFact* dump because of a roll-back during recovery but there *is* a very real chance of the file being left physically broken after an inopportune system crash. This might still be an acceptable risk since without checkpoint logging *SirFact* dumps will be extremely fast and have very little impact on system performance and because the procedure dictionary and table D allocations are fairly tolerant of physical inconsistencies. This latter point means that should the system crash and a *SirFact* dump procedure file be left physically inconsistent, most of the information in that file could be recovered after resetting FISTAT though after doing so, the file should be reorganized or re-INITIALIZED after moving the dump procs elsewhere. It is considerably more dodgy to depend on being able to do this if procedure deletions are being mixed with *SirFact* dump creation. Because of this, it is recommended that *SirFact* dump procedure file cleanup happen when *SirFact* dumps are unlikely (perhaps as part of a batch cycle) if the dump procedure file is being run with recovery turned off.

Of course, while *SirFact* dumps are very useful, it is probably not a tragedy if a few are lost because of a system crash (in fact the system crash itself would probably cause greater damage) so it might make sense to run with recovery completely turned off for the *SirFact* dump procedure file with **no** formal strategy for recovering from a system crash. In any case, because of the unusual nature of *SirFact* dumps, it is strongly recommended that no other data be stored in files that are to receive *SirFact* dumps.

3.5 SIRFACT IGNORE

SIRFACT IGNORE msg

SIRFACT IGNORE command syntax

The SIRFACT IGNORE subcommand indicates which request cancellation error messages are not to produce *SirFact* dumps.

msg The message number of the request cancellation message for which dumps are to be suppressed. The message number must include the message prefix (“M204” for standard *Model 204* messages, “MSIR” for Sirius messages and “USER” for user messages).

Usually a request cancellation error message is indicative of a programming error or a severe environmental problem. In the case of a programming error, it is very useful to get a *SirFact* dump to determine the cause of the error. In case of a severe environmental problem, it might be helpful to get a *SirFact* dump to determine the cause of the environmental problem (say a SIRFACT CANCEL intercepted \$SETG error) but more often than not a dump would not be particularly helpful, especially for problems that affect structures that are shared among many users. Finally, there are even a few request cancellation messages that are actually a normal part of operation! A successful VTAM TRANSFER, for example, results in the cancellation of the request that initiated the TRANSFER.

In any case, the SIRFACT IGNORE statement makes it possible to suppress *SirFact* dumps for messages for which the dumps would either be undesirable or useless. Examples of such error messages are

```
M204.0441 CCATEMP FULL
M204.0443 TABLE D FULL
M204.1229 TABLE B FULL -- INSERTS
M204.1230 TABLE B FULL -- APPENDS
M204.1231 TABLE B FULL -- SPILLS
M204.1270 TABLE C FULL - PROPERTY ENTRY
M204.1272 TABLE C FULL - PAGE ENTRY
M204.1273 TABLE C FULL - REDEFINE
M204.1808 ERROR IN EXECUTING TRANSFER STATEMENT
M204.1899 TRANSFER STATEMENT COMPLETE, ...
MSIR.0517 Cancelling request because of $SIRPARM
```

LISTFC setting (CCATEMP full)

Certainly, if CCATEMP or a file table fills up and *SirFact* dumps were not being suppressed for the associated message, a flood of *SirFact* dumps could occur as user after user trips over the same problem. To prevent getting *SirFact* dumps for these messages, simply issue the following commands during Online initialization:

```
SIRFACT IGNORE M204.0441
SIRFACT IGNORE M204.0443
SIRFACT IGNORE M204.1229
SIRFACT IGNORE M204.1230
SIRFACT IGNORE M204.1231
SIRFACT IGNORE M204.1270
SIRFACT IGNORE M204.1272
SIRFACT IGNORE M204.1273
SIRFACT IGNORE M204.1808
SIRFACT IGNORE M204.1899
SIRFACT IGNORE MSIR.0517
```

There might be other request cancellation errors for which a site might want to suppress *SirFact* dumps. Among these might be

```
M204.0452 TTBL FULL
M204.0562 REQUEST TOO LONG - table
M204.0574 REQUEST TOO LONG - VTBL
M204.0577 QTBL FULL -- DIRECT SEARCH
M204.2126 USER'S PUSHDOWN LIST OVERFLOWED
```

SoftSpy can issue some request cancellation errors which occur on the SoftSpy server thread; to ignore these errors, use the following command:

```
SIRFACT IGNORE M204.0000
```

3.6 SIRFACT MAXDUMP

```
SIRFACT MAXDUMP [type] limit
```

SIRFACT MAXDUMP command syntax

The SIRFACT MAXDUMP command establishes limits on the number of *SirFact* dumps to be taken. The default for the maximum dumps for the entire system is 0, so at least one SIRFACT MAXDUMP command must be issued to get **any** *SirFact* dumps.

type The type of limit being established. It can be one of the following values:

USER Maximum *SirFact* dumps to be taken in a single user logon. The default is no limit.

TOTAL Maximum *SirFact* dumps to be taken in the entire Online. The default is 0, so a “SIRFACT MAXDUMP TOTAL” must be issued to get any *SirFact* dumps.

If type is not specified, it is assumed to be “TOTAL”, so “SIRFACT MAXDUMP 20” is the same as “SIRFACT MAXDUP TOTAL 20”.

limit The maximum number of dumps to be taken for the indicated type.

A SIRFACT MAXDUMP should be issued at the start of any Online run for which you want *SirFact* dumps. If for some reason an application or environmental error is causing a “flood” of *SirFact* dumps, further dumps can be prevented with a “SIRFACT MAXDUMP 0”. Of course, this should not be a concern if a reasonable limit is set by a SIRFACT MAXDUMP at the start of the run.

As of *Sirius Mods* version 7.1, you can view the number of *SirFact* dumps taken since the *Model 204* Online initialization by issuing a SIRFACT DISPLAY MAXDUMP command (“SIRFACT DISPLAY” on page 14).

3.7 SIRFACT QUIESCE

```
SIRFACT QUIESCE subsys [WAIT sec] [BUMP]
```

SIRFACT QUIESCE command syntax

SIRFACT QUIESCE facilitates updates of APSY subsystem procedures while the subsystem is up and in use. This subcommand gets all subsystem users into a “quiescent” state where they are not likely to block procedure update operations or compile inconsistent copies of inner and outer subsystem procedures. Users are quiescent if they are evaluating a pre-compiled request (as determined by the request's prefix not its true pre-compilability) or between procedures in the subsystem driver loop.

When SIRFACT QUIESCE is issued:

- Any user that is going to switch to the next procedure in the subsystem driver is stopped until the subsystem is resumed.
- Any user that exits a pre-compiled procedure is stopped at end of request.
- Other users continue to run until they encounter one of the preceding two points.
- A user that is evaluating a pre-compiled request or that is between procedures is already considered quiesced.

The SIRFACT QUIESCE parameters are:

subsys The name of the subsystem to be quiesced.

sec The number of seconds to wait for the subsystem to be quiesced. If the subsystem quiesces before this time, SIRFACT QUIESCE returns at that point. If the subsystem is not quiesced after *sec* seconds, SIRFACT QUIESCE returns with a message stating that the subsystem is not yet quiesced, unless BUMP was specified.

The wait time cannot be set to 0. The default wait time is 3 seconds.

BUMP A *Model 204* BUMP, issued for any users still not quiesced after *sec* seconds. After bumping these users, SIRFACT QUIESCE waits another second for these users to notice the bump.

Bumped users do not actually disappear from the system: The BUMP sets the user's next procedure to the APSY error procedure, but the error procedure does not run because in a quiescing subsystem, users do not go to the next procedure.

If a SIRFACT QUIESCE does not completely quiesce a subsystem, it is quite likely that it will still allow for safe update and replacement of subsystem procedures — any user that hasn't quiesced within a reasonably long time will probably not cause or encounter any problems during an update operation. This is especially true if the X'40' bit of SIRFACT is set (see [“The SIRFACT Parameter” on page 37](#)). The BUMP option of SIRFACT QUIESCE offers further assurance that no subsystem users will impede or threaten procedure updates.

SIRFACT QUIESCE can be issued multiple times by one or multiple users for the same subsystem. In either case, only a single invocation of the SIRFACT QUIESCE inverse operation, SIRFACT RESUME, is necessary to stop the quiescing and resume normal operation of the subsystem. If a user is waiting for the return from SIRFACT QUIESCE, another user can cancel the quiesce and cause the first user to return immediately (with a warning message) by issuing a SIRFACT RESUME for the same subsystem.

As stated earlier, SIRFACT QUIESCE is valuable for doing APSY subsystem maintenance. Such maintenance probably requires appropriate settings of the SIRAPSYF parameter (as described in [“The SIRAPSYF Parameter” on page 40](#)) and of the SIRFACT parameter (as described in [“The SIRFACT Parameter” on page 37](#)).

3.8 SIRFACT RESUME

```
SIRFACT RESUME subsys
```

SIRFACT RESUME command syntax

SIRFACT RESUME counters the effect of SIRFACT QUIESCE (see [“SIRFACT QUIESCE” on page 23](#)): SIRFACT QUIESCE renders a currently operating subsystem

safe for updates by waiting for and stopping users until no users are inside a non-pre-compiled procedure or compiling pre-compiled procedures. SIRFACT RESUME stops this quiescing and returns the subsystem to normal operation.

If a user is waiting for the return from SIRFACT QUIESCE, another user can cancel the quiesce and cause the first user to return immediately (with a warning message) by issuing a SIRFACT RESUME for the same subsystem.

SIRFACT QUIESCE can be issued multiple times by one or multiple users for the same subsystem. In either case, only a single invocation of the SIRFACT QUIESCE inverse operation, SIRFACT RESUME, is necessary to stop the quiescing and resume normal operation of the subsystem.

3.9 SIRFACT RECNDUMP

SIRFACT RECNDUMP num

SIRFACT RECNDUMP command syntax

The SIRFACT RECNDUMP establishes the number of record numbers from each found set or LIST to be dumped. By default, *SirFact* dumps two record numbers from each found set or LIST, the first and the last. Note that *SirFact* does not actually dump the records in any found set or LIST unless a record in a found set or LIST is the current record in an active FOR EACH RECORD loop, in which case that single record is dumped.

The maximum value for SIRFACT RECNDUMP is 256 and the minimum is 1. The count of records in a found set or LIST is always dumped. If a found set or LIST contains fewer or the same number of records as the SIRFACT RECNDUMP value then the entire found set or list is dumped. If a found set or LIST contains more records than the SIRFACT RECNDUMP value then the first $(num+1)/2$ and last $num/2$ record numbers in the found set or LIST are dumped. For example, if SIRFACT RECNDUMP is 1 then only the first record number in a found set or LIST will be dumped, if the value is 32 then the first 16 and last 16 will be dumped and if the value is 127 then the first 64 and last 63 will be dumped.

The cost, of course, of a large SIRFACT RECNDUMP value is a somewhat larger dump if a request has many large found sets or LIST's though the cost is only about 6 bytes per record dumped per found set or LIST. So, for example, if there are 20 large found sets and LIST's in a request that is dumped a RECNDUMP value of 100 would make the dump $98*20*6$ or 11,760 bytes bigger than a dump with the default RECNDUMP value of 2. The cost to keeping RECNDUMP small is that potentially useful information is lost about the records in a found set or a LIST.

3.10 SIRFACT SNAP

```
SIRFACT SNAP usernum | userid [ FORCE ]
```

SIRFACT SNAP command syntax

The SIRFACT SNAP command requests that a *SirFact* dump is to be taken for another thread. This can be useful if a thread is looping or hung.

The SIRFACT SNAP command must be followed by either a user number or a userid. If a userid is specified, the userid must only be logged in on a single thread or *SirFact* will not take the dump for the thread. If a userid is logged in on multiple threads the dump must be requested by user number.

The optional FORCE parameter indicates that the *SirFact* dump is to be taken for the thread even if the thread holds one or more critical file resource locks. By default, *SirFact* will not take a dump for another thread on a SIRFACT SNAP command while the thread holds any critical file resource locks though it will try to wait a little bit (less than 3 seconds) for a window in which the thread being snapped does not hold any critical file resource locks.

There are certain cases where a SIRFACT SNAP command might fail for a looping thread. The most likely is when *SirFact* is unable to find a time when the thread being snapped does not hold a critical file resource lock. This behavior can be overridden by the specification of the FORCE parameter on the SIRFACT SNAP command. *SirFact* will also not take a dump for a thread if that thread is waiting on journal or checkpoint I/O. As with critical file resources locks, *SirFact* will try several times over the course of a few seconds to break in on the thread to be snapped when it is not waiting on checkpoint or journal I/O but if it fails, it gives up.

In most such failure cases, the SIRFACT SNAP command can be retried until it succeeds. If a few consecutive SIRFACT SNAP commands fail, the chances of success might be improved by setting the target thread's priority to LOW.

The SIRFACT SNAP command can be useful in diagnosing looping thread problems or problems where a user is presented with incorrect output. In both cases, help desk personnel or a system manager can get a *SirFact* dump for the thread with the problem and then do whatever is necessary to get the thread back on track.

SIRFACT snaps can be taken for non-evaluating threads even the one issuing the SIRFACT SNAP command, though the dump in such cases will have precious little besides GTBL.

Programming errors often cause symptoms that point directly to the error. For example, an incorrectly coded array assignment might result in a subscript range error on the very statement with the error. Alternatively, an assignment from the wrong variable to a screen item results in incorrect data appearing in the corresponding screen field. These kinds of programming errors are generally easy to isolate and fix, and they are usually caught during adhoc debugging or fairly quickly after a program goes into production.

Yet many other programming errors can cause more subtle problems that cause completely unrelated statements to fail, or even cause corruption of data that might not be detected until long after the original error has occurred. This often happens because much code depends on assumptions about the current environment, including assumptions about values of variables. A coding error or a misunderstanding of the environmental requirements of a chunk of code can cause the code to be run with invalid data. The code may execute but produce invalid results, or perhaps it may set values incorrectly that can cause problems in yet another part of the code.

There are several ways to deal with this problem with assumptions:

- Don't make assumptions in code. While it is an admirable goal to make code as flexible as possible, taken to the extreme, this approach produces code that is bloated by instructions to handle cases that never happen, setting return codes and status values that should never be set that then have to be checked elsewhere. Put another way, code has to do not only what is necessary but also has to perform many unnecessary tasks.
- Ignore the problem and hope for the best.
- Check key assumptions in code, and terminate the program with appropriate diagnostics to isolate the cause of the termination.

The last solution looks the most appealing. However, without the ASSERT statement, collecting the appropriate diagnostic information can only be done in User Language, so it can be tedious (numerous AUDIT, PRINT, or \$SETG statements), and it still provides only limited information.

The *Sirius Mods* provides a User Language statement, ASSERT, to get around these problems. The ASSERT statement serves three functions:

- It tests the validity of an assumption.
- It causes the current request to be cancelled if the assumption is incorrect. In an APSY subsystem, this causes transfer to the subsystem error procedure.

- It indicates the procedure and line number containing the failing ASSERT statement. Furthermore, in the presence of appropriate SIRFACT MAXDUMP and SIRFACT DUMP settings, it causes the creation of a *SirFact* dump that contains a wide variety of information about the program environment at the time of the error.

Stated another way, the ASSERT statement allows testing of assumptions and extensive diagnostic data collection with a single, simple statement.

```
ASSERT cond [, [SNAP] [INFO info] [CONTINUE] ]
```

ASSERT statement syntax

- cond* The conditions that are being asserted as true. These conditions have exactly the same syntax as conditions on IF statements.
- SNAP Indicates a CCASNAP is to be taken on an assertion failure. A CCASNAP is taken in addition to, but before, any *SirFact* dump associated with the assertion failure.
- info* Extra information that is included in the audit trail and terminal output for the assertion failure as part of a MSIR.0494 message. *info* must be enclosed in quotes if it contains spaces or other *Model 204* separator characters.
- CONTINUE Indicates that an assertion failure will *not* cause the request to be cancelled. An MSIR.0494 message, and possibly a *SirFact* dump, will still be produced, but the request will continue.

Some valid ASSERT statements are :

```
ASSERT (%X GT 0) AND (%NAME NE '')
ASSERT %X GT 0, SNAP
ASSERT NOT $SETG('NAME', 'VALUE')
ASSERT %X = 22, INFO %X
ASSERT %X, INFO 'Zero %X' SNAP
ASSERT %INCOME GT 10000, CONTINUE
```

Note: As of *Sirius Mods* version 6.7, a %variable in the *Info* clause is interpreted to mean the contents of the indicated variable. For example, the fourth ASSERT statement above (ASSERT %X = 22, INFO %X) produces the following message if %X is 21:

```
MSIR.0494: Assert info: 21
```

For versions prior to 6.7, an unquoted %variable name is interpreted as a literal, and the message produced is: MSIR.0494: Assert info: %X.

An ASSERT statement uses the same expression handler as the IF statement, so it is exactly as efficient as an IF statement with the same conditions.

To use an ASSERT, simply place it before any code that depends on some assumptions about variables or the environment. ASSERT statements should be coded to test values or relationships that are required for the code to run correctly, but whose values are not immediately apparent from the surrounding code.

In addition to catching coding errors, the ASSERT statement provides the following benefits :

- It makes clear the assumptions that the code depends on to anyone scanning the program. This makes it easier to understand the surrounding code. Similarly, it makes the environmental requirements clear to someone wanting to re-use a code fragment or call a common subroutine. While these benefits can be achieved with comments, the ASSERT statement has the added benefit that it enforces the restrictions.
- It eliminates doubt when scanning code while trying to debug a problem, and it prevents wasted time on “what if” scenarios that can be ruled out with a simple ASSERT.

The SIRFACT CANCEL command (“SIRFACT CANCEL” on page 11) makes it possible to trap return codes from \$functions that are indicative of programming errors or severe environmental problems. For simplicity and consistency, the scope of SIRFACT CANCEL for a \$function is global, that is, it applies to every program running on every thread in the Online. This should not be a problem, as once again, SIRFACT CANCEL should only be used to trap **severe** problems.

However, even the most unlikely return codes might be handled by User Language code in certain odd instances. For these cases, the **SIRFACT** User Language statement is provided to temporarily disable *SirFact* error trapping for **all** \$functions or FRN errors.

```
SIRFACT ON | OFF
```

SIRFACT statement syntax

ON | OFF **OFF** indicates that *SirFact* \$function error trapping is to be temporarily disabled for the current thread. **ON** indicates that it is to be reenabled.

Without the presence of a SIRFACT statement in a User Language program, *SirFact* \$function error trapping is always enabled.

The end of a User Language request automatically reenables *SirFact* \$function error trapping. That is, a SIRFACT OFF only applies to the program in which it is executed.

The SIRFACT OFF and SIRFACT ON statements are evaluated and are **not** compiler directives. This means that in the following chunk of code:

```
IF %RECOVER THEN
    SIRFACT OFF
END IF

%RC = $SETG('NEXTPROC', 'XP.MAIN-MENU')

IF %RECOVER THEN
    SIRFACT ON
END IF
```

the \$SETG will be executed with *SirFact* error trapping disabled if %RECOVER is non-zero; otherwise, it will run with *SirFact* error trapping enabled.

There is no harm in using `SIRFACT ON` if *SirFact* error checking is already enabled, nor in using `SIRFACT OFF` if *SirFact* error checking is already disabled. The previous example could just as easily have been coded:

```
IF %RECOVER THEN
  SIRFACT OFF
END IF

%RC = $SETG('NEXTPROC', 'XP.MAIN-MENU')

SIRFACT ON
```

Also valid (but pointless) is this:

```
SIRFACT OFF
SIRFACT OFF

%RC = $SETG('NEXTPROC', 'XP.MAIN-MENU')

SIRFACT ON
```

The `SIRFACT OFF` setting also disables `SIRFACT FRN` error trapping set by the X'08', X'10', or X'20' bits in the `SIRFACT` system parameter (“[The SIRFACT Parameter](#)” on [page 37](#)). This means that if `SIRFACT FRN` error trapping is turned on, but an `FRN` statement, by design, sometimes refers to a non-existent record, the `FRN` could be coded as follows:

```
SIRFACT OFF
%HAVEREC = 0
IN FILE ODD FRN %RECNO
  %HAVEREC = 1
END FOR
SIRFACT ON
```

It is good programming practice to minimize the number of User Language statements inside a `SIRFACT OFF/SIRFACT ON` bracket to prevent accidentally leaving *SirFact* \$function error checking disabled. It is even better programming practice to avoid the use of `SIRFACT OFF` altogether. For example, suppose that a `SIRFACT CANCEL` has been set up to cancel requests on a return code of `-6` from `$LISTDEL`. And suppose a procedure has the following chunk of code :

```
%LIST = $LISTRST('SAVEDLIST')
%RC   = $LISTDEL(%LIST)
```

Suppose sometimes this code is run where there is no `$LISTSAV`'ed \$list under the name “SAVEDLIST”. Before *SirFact*, the `$LISTRST` would simply return a `-14`, indicating that no \$list is saved under the indicated name. When the `-14` is passed to `$LISTDEL`, `$LISTDEL` would return a `-6`, indicating that `-14` is an invalid \$list identifier (all \$list identifiers are positive).

Probably this would not matter: since the ostensible purpose of this code is to make the name "SAVEDLIST" available to save another \$list, the \$LISTDEL "failure" is irrelevant. Unfortunately, with the `SIRFACT CAN $LISTDEL -6` active, this code would often cause a request cancellation.

One solution to this problem would be to simply stop trapping return code -6 from \$LISTDEL. Unfortunately, this would mean that the benefits of *SirFact* error trapping would be lost for mis-coded \$LISTDELS. Another option would be to change the above code to

```
%LIST = $LISTRST('SAVEDLIST')
SIRFACT OFF
%RC   = $LISTDEL(%LIST)
SIRFACT ON
```

This gets around the problem, but it is a bit sloppy and does not really make clear what is going on. A "cleaner" solution would be to change the code to

```
%LIST = $LISTRST('SAVEDLIST')
IF %LIST GT 0
    %RC   = $LISTDEL(%LIST)
END IF
```

This solution actually uses slightly less QTBL space, uses no more CPU, and makes the code clearer.

Ultimately, while the SIRFACT OFF statement can be very useful, it should be used with discretion.

CHAPTER 6 *The TRACE Statement*

The TRACE statement acts very much like a PRINT or AUDIT statement with the exception that the target for the TRACE statement can be dynamically changed by RESET'ing the ULTRACE parameter. The ULTRACE parameter is a standard bit-oriented user parameter where the bits mean:

- X'01'** Send TRACE output to terminal.
- X'02'** Send TRACE output to audit trail.
- X'04'** Send TRACE output to a CCATEMP wrap-around trace table.

The default setting for ULTRACE is X'01' which means that the TRACE statement will act pretty much like a PRINT statement. The bits can be combined so that output could be sent to both the terminal and the audit trail or to the CCATEMP wrap-around trace table and to the terminal and to the audit trail. If a TRACE statement is executed when ULTRACE is set to 0, the request is cancelled.

The CCATEMP wrap-around trace table is a user-specific trace table that consists of a set of CCATEMP pages up to a maximum specified by the ULTRACEP user parameter. If ULTRACEP pages are already in use when a TRACE statement is issued and output is being routed to the trace table (ULTRACE X'04' set), the trace data on the oldest page is discarded and the oldest page is re-used for the new trace data. The default value for ULTRACEP is 2.

The wrap-around trace table is dumped in *SirFact* dumps, and it can be viewed with the following command when using the FACT subsystem to examine a dump:

```
D[ISPLAY] T[RACE][.{* | N}]
```

where N specifies that the last *n* entries are to be displayed. Specifying D T will display all trace entries, and specifying D T.20 will display the last 20.

The wrap-around trace table can also be examined with the \$TRACE2LIST function (“\$TRACE2LIST: Copy wrap-around trace table to \$list” on page 52).

ULTRACE and ULTRACEP can both be RESET via the \$RESETN function.

CHAPTER 7 *System Parameters*

SirFact makes use of two system parameters: SIRFACT and SIRAPSYF. These parameters are *Sirius Mods* system parameters that are honored only if the Online is running with the *SirFact* product authorized. The parameters can only be set as a parameter on the EXEC card (or the *Model 204* call under CMS) or as a user 0 parameter in the CCAIN stream.

7.1 The SIRFACT Parameter

The SIRFACT system parameter consists of several bits that can control the collection of compilation data and the trapping of certain User Language coding errors. The bits defined for the *SirFact* parameter are:

- X'01'** Collect quad offset to procedure line number mapping information to CCATEMP.
- X'02'** Collect quad offset to procedure line number mapping information to server tables even if it is also being collected to CCATEMP.
- X'04'** Don't do comment initialized global dummy string substitution (“[Comment-initialized global variables](#)” on page 55).
- X'08'** Cancel FOR RECORD NUMBER or FRN statements where the record number is a null string.
- X'10'** Cancel FOR RECORD NUMBER or FRN statements where the record number is not a valid number.
- X'20'** Cancel FOR RECORD NUMBER or FRN statements where the record number is not found.
- X'40'** When a procedure is included (whether as part of an APSY subsystem or directly from *Model 204* command mode), copy it to CCATEMP. After the copy, release the share enqueue on the procedure. As a result, a user who includes a procedure does not prevent others from updating the procedure.

The copy of the procedure to CCATEMP incurs some extra (barely measurable) overhead.
- X'80'** Enable the *SirFact* APSY maintenance enhancements you specify with the SIRAPSYF parameter (see “[The SIRAPSYF Parameter](#)” on page 40).

Also, regardless of the SIRAPSYF settings, release the share enqueue on a procedure when the last line of the procedure is read, not when the line after the last line is attempted to be read.

7.1.1 Subsystem procedure enqueues

Unless the SIRFACT X'80' bit is set, a procedure that ends in an END statement is locked in share mode until procedure evaluation completes and *Model 204* attempts to get the next line from the procedure. If the evaluation takes a long time because the procedure does terminal I/O or contains a long-running program, or because of some other reason, the procedure is ordinarily not updateable until evaluation completes.

Note: Cosmetic elements like blank lines or comments after an END nullify the benefit of this enhancement, because the lock remains until the last line is read.

Should a procedure depend on the enqueue being held (because it does a \$RDPROC, \$PROCOPN, or something similar against itself and wants to avoid being deleted), either do not set the SIRFACT X'80' bit, or add a blank or comment line to the end of any such procedure. However, if the SIRFACT X'40' bit is on, all procedures are dequeued before they evaluate, so procedures cannot depend on \$RDPROCs or \$PROCOPNs against themselves working, unless they can depend on this because of local policies or rules.

7.1.2 SIRFACT compilation data collection

One of the most important pieces of information in determining the cause of an error is the code location of the error. Knowing the specific line of code in which an error occurred can be critical in determining the cause of the error. Yet typically, *Model 204* does not collect the information required to produce this information. This is because *Model 204* actually runs quads, not User Language statements, at evaluation time.

These evaluation time quads are built from the original User Language at compile time, but once built, there is generally no way to determine the line of User Language that created a particular quad. So when an error occurred, *Model 204* would generally be able to report only the QTBL offset of the executing quad, which would be a fairly useless piece of information for most purposes.

The solution to this problem is to collect information about the mapping of quads to User Language statements at compile time. There are several ways this is done, each with its own cost.

- The DEBUGUL parameter, when non-zero, causes the User Language statement-to-quad mapping to be stored in QTBL, VTBL, and STBL.

The DEBUGUL parameter is a standard *Model 204* user parameter. While using this parameter has little compile time overhead and no performance overhead, it does add 8 bytes to the length of each quad, which will tend to increase QTBL

requirements by 30-50%. There would be more minor increases in VTBL and STBL, but it is the QTBL utilization expense that makes it difficult to use DEBUGUL in a production environment.

- The SIRFACT parameter, when the X'01' bit is set, causes the User Language statement-to-quad mapping to be stored in CCATEMP.

Because accessing CCATEMP requires logical disk I/O, setting the SIRFACT X'01' bit will have a slight compilation time cost. It will also increase CCATEMP utilization, though probably by less than 10%. CCATEMP writes will also increase slightly, but since these are asynchronous, these will have minimal or no impact on system performance.

Finally, since the mapping pages are not accessed at evaluation time, setting the SIRFACT X'01' bit will have **no** effect on evaluation time performance and, because all the information is stored in CCATEMP, **no** effect on server sizes.

- If DEBUGUL is 0 and the SIRFACT X'01' bit is not set, the ASSERT statement will store extra information in QTBL and STBL to identify the location of an ASSERT statement error.

This causes the ASSERT statement to use 12 more bytes of QTBL and some additional STBL (to hold procedure names) in these cases. In any case, errors that happen as the result of ASSERT statements *always* produce a procedure and line number in the error message.

- If DEBUGUL is 0 and the SIRFACT X'01' bit is not set, \$functions that are being “monitored” because of the SIRFACT CANCEL command (“[SIRFACT CANCEL](#)” on [page 11](#)) will store extra information in QTBL and STBL to identify the location of a \$function error.

This causes each monitored \$function to use 12 more bytes of QTBL and some additional STBL (to hold procedure names). In any case, errors that happen as the result of SIRFACT CANCEL trapped \$function errors *always* produce a procedure and line number in the error message.

DEBUGUL and the SIRFACT X'01' bit cause the collection of the same information; the difference is where the information is stored, and the fact that DEBUGUL is a user parameter (so can be controlled on a user or application level, while the SIRFACT X'01' bit is a system parameter).

In any case, all the mechanisms used to collect mappings of User Language to quads result in error time messages that indicate the following:

- The procedure (or command level indication)
- The file that contains the procedure
- The line number within the procedure where the error happened

The actual contents of the line of User Language are not saved in any case, and so they are not displayed. It is possible in many cases for the procedure to change after it's been compiled (especially for INCLUDE'd procedures), in which case the indicated line numbers might not match exactly what is in the updated procedure.

No matter what the collection mechanism, the User Language to quad mapping is saved in *SirFact* dumps, so the location of the error can be retrieved from the dump. The User Language to quad mapping is also used by *SirFact* dump analysis to provide a subroutine callback trace, as in “subroutine at line whatever in proc whatever called from line whatever in proc whatever”. This is only available in *SirFact* dumps collected for programs compiled with the SIRFACT X'01' bit or with DEBUGUL set to a non-zero value.

Note that even if the User Language to quad mappings were not collected, all other information such as %variable values, global variable values, and so on are still available in *SirFact* dumps. Nevertheless, the location of the error and subroutine callback trace are often crucial pieces of information, so it is suggested that if *SirFact* dumps are to be collected, either set DEBUGUL to a non-zero value, or set the SIRFACT X'01' bit.

7.2 The SIRAPSYF Parameter

The SIRAPSYF system parameter controls a number of APSY subsystem maintenance features. The bit options defined for the features are described below. X'00' (no options enabled) is the parameter's default value.

Note: No SIRAPSYF features are enabled unless you also set the X'80' bit on the SIRFACT parameter (see [“The SIRFACT Parameter” on page 37](#)).

X'01' Allows procedure compilations to be saved (pre-compiled) for unlocked procedure group members. If an outer or an inner procedure in an unlocked file in a procedure group is changed, or if an outer procedure is added to an unlocked file in a procedure group, the procedure is recompiled and that compilation is saved.

Also allows the pre-compiling of a procedure with a pre-compile prefix that was not present in the procedure group when the subsystem was started.

If this bit is not set, using unlocked files to facilitate the updating of procedures in a running subsystem has an efficiency cost because procedure compilations are not saved.

This setting has no effect on procedures in subsystems that use a procedure file instead of a procedure group, and it has no effect on subsystems that use a procedure group but not unlocked files.

X'02' Detects changes to included procedures that reside in a pre-compiled procedure in a subsystem procedure group. If such an included procedure is changed, the pre-compiled procedure is recompiled.

This setting has no effect on procedures in subsystems that use a procedure file instead of a procedure group, and it has no effect on subsystems that use a procedure group but not unlocked files.

X'04' Tracks in a bitmap the CCATEMP pages allocated to pre-compiled procedures in a subsystem. When the subsystem is stopped, this bitmap is used to free the pages rather than chaining through them, which requires considerable CCATEMP I/O. Although the bitmap method has more (but probably not measurable) overhead while saving compilations, it can make the STOP SUBSYSTEM process significantly faster.

The bitmap is subsystem-wide and not procedure-specific. It does not reduce the time required for discarding the CCATEMP pages that are associated with a compilation that is being replaced.

Usage notes:

- These SIRAPSYF features along with the SIRFACT X'40' and X'80' bits are designed to simplify the updating of procedures and the pre-compiling of these updated procedures while their subsystem is in use.

Note: These settings do **not** eliminate the lock on outer procedures in locked procedure files. They are designed to suit a procedure group and the placement of updated procedures in unlocked file(s).

The SIRFACT QUIESCE and RESUME subcommands supplement these features by preventing subsystem users from interfering with procedure update operations. For more information, see [“SIRFACT QUIESCE” on page 23](#) and [“SIRFACT RESUME” on page 24](#).

- For both the X'01' and X'02' bits, an inner or outer procedure is considered changed if the actual procedure is modified or if a new version of the procedure is added to an earlier file in the procedure group.
- When using temporary procedure groups, a request compilation is not saved if any of the outer or inner procedures came from a file not in the subsystem's permanent group. Furthermore, if the outer procedure is found in a file not in the subsystem's permanent group, it will always be recompiled. If an inner procedure (but not the outer) is found in a file not in the subsystem's permanent group, whether the procedure is recompiled depends on the X'02' bit setting:
 - If the bit is off, the procedure might or might not be recompiled.
 - If the bit is on, the procedure is always recompiled.

Hence, it is recommended that where temporary procedure groups are to be used, the X'02' bit is to be set.

- The SIRAPSYF features you specify apply on a system-wide basis. To specify an override for an individual subsystem, you can specify in CCASYS a special deferred update DD name for a procedure group that defines its particular SIRAPSYF option:

```
SUBSYSTEMGMT          Subsystem File Use
                        Update Mode

Subsystem Name: SALES      From: -

File/Group Name      File Location  Group Y/N  Auto Y/N  Mandatory Y/N  Procs NUMLK  Deferred Name  Ordered-index Deferred Name
PR SALESPRC          N      Y      Y
1: PAYROLL           N      Y      Y
2: EMPLOYEE          N      Y      Y
3: CLIENTS DALLAS    N      Y      Y
4: PRODUCTS DALLAS    N      Y      Y
5: SALESGRP          Y      Y      Y      TAPE*07
===>

1=HElp      2=      3=QUIt      4=OPERation  5=PRORcedure  6=
7=BACKward  8=FORward  9=USERdef   10=          11=SYSclass  12=END
```

SIRAPSYF override

As shown for SALESGRP above, you must:

- In SUBSYSTEMGMT, specify the special deferred update DD name under “Deferred Name” on the Subsystem File Use screen. Or, if you use an ad hoc procedure, specify this special name for the APSFDN field in the SCLS records for the subsystem.
- Begin the deferred update DD name with the characters “TAPE*” and append the two hexadecimal digits that indicate the subsystem-specific bit settings you want for the *SirFact* APSY enhancements. “TAPE*07” would set the SIRAPSYF X'07' features for the group.

Setting a deferred index update file name is completely harmless in systems that do not have the *SirFact* APSY facility or do not have the facility enabled (by specifying the SIRFACT X'80' bit).

When a *SirFact* dump occurs, information from the time of the error is stored in a dump procedure as specified by the applicable SIRFACT DUMP command (“[SIRFACT DUMP](#)” on page 16). This information is binary data that includes the contents of the user's QTBL, STBL, VTBL, GTBL, NTBL, \$lists, and several other user-specific structures. To simplify the movement of data to other regions, the data is *base64-encoded*. Base-64-encoding is a means of storing binary data by using only 64 **displayable** characters. Base-64-encoding **does** use lowercase characters, however, so to load a *SirFact* dump into an Online using the PROCEDURE statement, *LOWER must be set.

SirFact dumps contain binary data for data structures that are interrelated in fairly complex ways. Modification of the data in these dumps, no matter how slight, could cause severe problems in analyzing the dump. Yet, the dumps are in *Model 204* procedures, which can be easily modified either intentionally or accidentally.

To detect any modification of the data in a *SirFact* dump, all dumps contain a 16-byte checksum (really an MD5 digest) of the contents of the dump. Any modification of the data in a dump will be detected because of a mismatch between the checksum in the dump and that calculated from the data. If such a mismatch is detected when the dump is read, *SirFact* will not return data from the dump, because all data structures in the dump will be suspect.

SirFact provides a set of \$functions that can be used to analyze a dump. These \$functions are used by the FACT subsystem (“[The FACT Subsystem](#)” on page 57) to provide a user interface for reading dumps. It is not necessary to explicitly use these \$functions to analyze *SirFact* dumps, yet documentation for these \$functions is provided in case a *SirFact* user wants to write a different user interface to *SirFact* dumps, perhaps one that takes advantage of site-specific facilities or standards.

Although *Sirius Mods* version 6.5 made mixed-case User Language available for use with many Sirius products, the *SirFact Reference Manual* retains the all-uppercase presentation for \$function names and User Language entities. For more information about mixed-case User Language, see the *Janus SOAP Reference Manual*.

To use the *SirFact* \$functions to look at a dump, you must first issue \$FACT_INIT. \$FACT_INIT starts an sdaemon (see the *Sirius Mods Installation Guide*), which then tries to read the dump procedure. If the dump procedure looks alright (it has the correct format and the checksum is correct), this sdaemon loads the data at the time of error into its own tables (and into CCATEMP where appropriate). Once the dump is loaded, this sdaemon remains logged on until one of the following events:

- A \$FACT_DONE is issued.

- Another \$FACT_INIT is issued by the same user. Only one *SirFact* dump can be opened by a single user at any time.
- The user that issued the \$FACT_INIT logs off.
- The sdaemon is bumped.

A *SirFact* sdaemon runs under the same userid and account ID as the invoking user.

After a successful \$FACT_INIT, the user that invoked the \$FACT_INIT communicates with the *SirFact* sdaemon using these \$functions:

\$FACT_CMD	Issues a command that runs on the <i>SirFact</i> sdaemon.
\$FACT_CONTEXT	Sets subroutine context for variable names.
\$FACT_DATA	Requests error time data from the sdaemon.
\$FACT_DONE	Tells sdaemon to go away.
\$FACT_OPTION	Sets or gets <i>SirFact</i> display options.
\$FACT_VNAME_WIDTH	Indicates how many characters to reserve for the variable names in the sdaemon output for \$FACT_DATA.
\$TRACE2LIST	Copies data from the <i>SirFact</i> wrap-around trace table in CCATEMP to a \$list, facilitating use of the wrap-around trace table for interactive debugging.

\$FACT_CMD and \$FACT_DATA return data into \$lists. The functions to create and manipulate \$lists are not documented here, but they are documented in the ***Sirius Functions Reference Manual***. *SirFact* customers are automatically authorized to use all the base \$list functions (and many more). In fact, *SirFact* customers can use these functions in applications that have nothing to do with *SirFact* or any other Sirius Software product.

8.1 CALLing Sirius \$functions

As of version 6.5 of the *Sirius Mods*, you can invoke many of the Sirius \$functions using a User Language CALL statement instead of assigning the function result to a %variable. For example:

```
%L = $LISTNEW
$LISTADD(%L, 'Once upon a midnight dreary')
$LISTADD(%L, 'As I pondered weak and weary')
CALL $LIST_PRINT(%L)
```

You can CALL such \$functions and still test for their return code, if necessary. For example:

```
CALL $LIST_PRINT(%L)
  IF $LIST_PRINT(%L) THEN
```

This "callability" is an optional approach; it does not replace %variable assignment.

The callable \$functions are indicated as such in their individual function descriptions in this document. Typically they are \$functions that do more than simply return a value, and the value they return is primarily an indicator of whether the function completed successfully. \$LISTCNT, for example, is a (non-callable) \$function that just returns a value.

8.2 **\$FACT_CMD: Run a command on the SirFact SDAEMON**

The \$FACT_CMD function is used to run a command on the *SirFact* SDAEMON and capture the results. This \$function is mainly used for the purpose of debugging *SirFact*. \$FACT_CMD returns the command output to a \$list.

\$FACT_CMD accepts two arguments and returns a numeric code. It is also callable (["CALLing Sirius \\$functions" on page 44](#)).

The first argument is the list identifier for the output \$list. Data is appended to the end of this \$list. The \$list can be created with the \$LISTNEW function as documented in the ***Sirius Functions Reference Manual***. This is a required argument.

The second argument is a *Model 204* command. The only commands currently allowed are "*LOOK", "*ZAP" and "VIEW". The results of the "VIEW" command are the current settings on the *SirFact* SDAEMON, not the settings at the time of the error for the dump being examined.

```
%RESULT = $FACT_CMD(listid, command)
```

\$FACT_CMD Function

%RESULT is set to indicate the success of the function.

0 - All is well, data returned
1 - No active <i>SirFact</i> request
2 - <i>SirFact</i> SDAEMON no longer around
3 - Out of CCATEMP
4 - \$list limit exceeded
5 - Required parameter not specified
6 - Invalid list-identifier
7 - Invalid command
255 - Other severe error in <i>SirFact</i> SDAEMON

\$FACT_CMD return codes

The statement

```
%RC = $FACT_CMD(%OLIST, 'V LQTBL')
```

requests the size of QTBL for the *SirFact* SDAEMON.

8.3 \$FACT_CONTEXT: Set subroutine context for \$FACT_DATA

The \$FACT_CONTEXT function is used to set the subroutine context for value class requests for subsequent \$FACT_DATA requests.

\$FACT_CONTEXT accepts one argument and returns a numeric code. It is also callable ([“CALLing Sirius \\$functions” on page 44](#)).

The only argument is the name of the complex subroutine to use as the context for subsequent value class \$FACT_DATA requests. A period (.) indicates the error time context and an asterisk (*) indicates non-subroutines or main program context.

%RESULT = \$FACT_CONTEXT(context)

\$FACT_CONTEXT Function

%RESULT is set to indicate the success of the function.

0 - All is well
1 - No active <i>SirFact</i> request
2 - <i>SirFact</i> SDAEMON no longer around
5 - Required parameter not specified
6 - Context is invalid
255 - Other severe error in <i>SirFact</i> SDAEMON

\$FACT_CONTEXT return codes

The following statement sets the context for subsequent value class requests for \$FACT_DATA to subroutine “NASTY.ALGORITHM”:

```
%RC = $FACT_CONTEXT('NASTY.ALGORITHM')
```

Following are the valid input values to \$FACT_CONTEXT:

- * (asterisk)** Switches FACT context to the mainline of the executing program.
- . (period)** Switches to whichever context the program was in at the time of the *SirFact* error.
- <subroutine name>** Switches to the context of the named complex subroutine. Simple subroutines are not valid contexts, as they share the same context as the main body of the program.

8.4 \$FACT_DATA: Retrieve data from a SirFact dump

The \$FACT_DATA function is used to retrieve data about the environment of the error that caused the currently open *SirFact* dump to be taken. There must be a current active dump set up via \$FACT_INIT. \$FACT_DATA returns the value(s) of the requested data to a \$list.

\$FACT_DATA accepts two arguments and returns a numeric code. It is also callable (“CALLing Sirius \$functions” on page 44).

The first argument is the list identifier for the output \$list. Data is appended to the end of this \$list. The \$list can be created with the \$LISTNEW function, as documented in the *Sirius Functions Reference Manual*. This is a required argument.

The second argument is a blank-delimited list of data items whose values are to be returned to the \$list. The format of each data item is:

```
[X.]c[.s]
```

where:

- X** An optional flag specifying that the requested data should be displayed in hexadecimal format.
- c** The class of data requested.
- s** An optional selection pattern for data of class *c*.

The classes and their formats are described in “[D\[isplay\] command: Displaying data from the SirFact dump](#)” on page 60.

```
%RESULT = $FACT_DATA(listid, datalist)
```

\$FACT_DATA Function

%RESULT is set to indicate the success of the function.

```
0 - All is well, data returned
1 - No active SirFact request
2 - SirFact SDAEMON no longer around
3 - Out of CCATEMP
4 - $list limit exceeded
5 - Required parameter not specified
6 - Invalid list-identifier
255 - Other severe error in SirFact SDAEMON
```

\$FACT_DATA return codes

The statement

```
%RC = $FACT_DATA(%OLIST, 'G.USER.* %SCR:CMD')
```

requests values for all globals that begin with the five characters “USER.” and requests the value of screen item %SCR:CMD.

8.5 \$FACT_DONE: Terminate a SirFact dump

The \$FACT_DONE function is used to terminate the *SirFact* SDAEMON being used to access a dump procedure.

\$FACT_DONE accepts no arguments and returns a numeric code. It is also callable (“[CALLing Sirius \\$functions](#)” on page 44).

```
%RESULT = $FACT_DONE
```

\$FACT_DONE Function

%RESULT indicates whether SDAEMON had been active.

```
0 - SirFact SDAEMON was not active for user
1 - SirFact SDAEMON now terminated
```

\$FACT_DONE return codes

The following statement terminates processing of the active *SirFact* dump:

```
%RC = $FACT_DONE
```

8.6 **\$FACT_INIT: Open a SirFact dump**

The `$FACT_INIT` function is used to initiate a *SirFact* sdaemon and have it read and load the data from a dump procedure into its own tables.

`$FACT_INIT` accepts three arguments and returns a numeric code.

The first argument is the name of the file or group that contains the *SirFact* dump to be opened. Any of the words "FILE", "GROUP", "TEMP GROUP" or "PERM GROUP" may precede the name to make the context explicit. The file or group specified must be open with procedure display privileges (CURPRIV X'0200' bit set). If the first argument is not specified, the compile time context is used.

The second argument is the name of the procedure that contains the dump. This is a required argument.

The third argument is the output line width for data being returned to a \$list by `$FACT_DATA` and `$FACT_CMD`. This must be a value between 32 and 4095 inclusive, and it defaults to 255 if not specified.

```
%RESULT = $FACT_INIT(fgname, procname, owidth)
```

\$FACT_INIT Function

%RESULT is set to indicate the success of the function.

```
0 - All is well, SirFact SDAEMON set up
1 - Dump file is not open
2 - Insufficient privilege to open proc
3 - Dump proc name missing or proc not found
4 - Proc is enqueued exclusive
5 - Invalid line width (arg 3)
6 - Insufficient virtual storage
7 - No SDAEMONS available
8 - Not a valid dump proc
9 - Backward compatibility problem
10 - Forward compatibility problem
11 - Model 204 release compatibility problem
12 - Can't get table sizes
255 - Severe error in SirFact SDAEMON
```

\$FACT_INIT return codes

This statement opens a procedure called `DUMP.JUNK.01` in file `DUMPPROC`:

```
%RC = $FACT_INIT('FILE DUMPROC', 'DUMP.JUNK.01')
```

8.7 \$FACT_OPTION: Set or get SirFact display options

The `$FACT_OPTION` function is used to set or get the values of options that affect the retrieval and display of `$FACT_DATA` output.

`$FACT_OPTION` accepts two arguments and returns a numeric code or a string. It is also callable ([“CALLing Sirius \\$functions” on page 44](#)).

- The first argument, which is required, is the name of the option that is being updated or retrieved. Current options are:

CASE Controls whether `$FACT_DATA` internally uppercases all non-quoted characters in its arguments before processing. Such auto-uppercasing provides case-insensitivity and mimics the way mixed-case User Language is supported. Prior to *Sirius Mods* version 6.7, `$FACT_DATA` respected the case of its arguments; that is, it performed no automatic uppercasing.

Valid `CASE` values are `LEAVE` (do not uppercase) and `TOUPPER`; `TOUPPER` is the default. `LEAVE` is useful for accommodating dumps created from code compiled with case-sensitive User Language, if case-sensitivity is necessary. Case-sensitive User Language is enabled by starting a User Language program with an all-uppercase `BEGIN` statement or by specifying the Sirius compiler directive `Case Leave`.

Note: Prior to *Sirius Mods* version 6.7, the `FACT` subsystem uppercased **all** data passed to `$FACT_DATA` (which processes `FACT` system `DISPLAY` commands). However, this meant that mixed-case method arguments were also uppercased before `$FACT_DATA` processing (for example, `d %myXmlDoc:print('/outer/inner')`, became `D %MYXMLDOC:PRINT('/OUTER/INNER')`). In version 6.7, this `FACT` subsystem uppercasing was replaced by the `$FACT_DATA` uppercasing (which correctly processes the example command as `D %MYXMLDOC:PRINT('/outer/inner')`).

The option to change the default case handling is **not** available to `FACT` subsystem users.

IMPLIM Sets the limit for the number of lines that `$FACT_DATA` can output for the implied-Print of an object variable's content. An implied Print is an automatic invocation of the `Print` method for the display of the content of an object variable, for those object classes for which such printing is

enabled. For more information about implied printing, see the "Objects" discussion in [“Additional syntax for VALUE and LIST” on page 64](#).

Valid IMPLIM values are numbers between 1 and 99999999. The default is 500.

- The second \$FACT_OPTION argument is an optional new value for the option you specify as the first argument. If you specify a value for this parameter that changes the current setting of the first parameter, the \$FACT_OPTION return depends on whether you are changing CASE or IMPLIM:
 - If CASE is changed successfully, the return code is 0.
 - If IMPLIM is changed, the return is the previous value of IMPLIM.

```
%RESULT = $FACT_OPTION(option, value)
```

\$FACT_OPTION Function

%RESULT is set to indicate the success of the function or the former value.

Examples of valid \$FACT_OPTION statements follow:

```
$FACT_OPTION('CASE', 'LEAVE')  
print $FACT_OPTION('CASE')  
%oldLimit = $FACT_OPTION('IMPLIM', %newLimit)  
$FACT_OPTION('IMPLIM', 2)
```

8.8 \$FACT_VNAME_WIDTH: Sets variable name width for \$FACT_DATA

The \$FACT_VNAME_WIDTH function is used to set the space allocated for variable names in \$FACT_DATA output.

\$FACT_VNAME_WIDTH accepts one argument and returns a numeric code. It is also callable ([“CALLING Sirius \\$functions” on page 44](#)).

The only argument is the width of the output space for variable names for subsequent \$FACT_DATA requests. The value specified must be between these values:

- 8
- The output line width implied or specified on \$FACT_INIT, minus 11.

This is a required parameter.

```
%RESULT = $FACT_VNAME_WIDTH(vwidth)
```

\$FACT_VNAME_WIDTH Function

%RESULT is set to indicate the success of the function.

```
0 - All is well  
1 - No active SirFact request  
2 - SirFact SDAEMON no longer around  
6 - Width is invalid
```

\$FACT_VNAME_WIDTH return codes

The following statement sets the space for variable names for subsequent \$FACT_DATA requests to 33 characters:

```
%RC = $FACT_VNAME_WIDTH(33)
```

8.9 \$TRACE2LIST: Copy wrap-around trace table to \$list

The \$TRACE2LIST function is used to copy data from the *SirFact* wrap-around trace table in CCATEMP (“[The TRACE Statement](#)” on page 35) to a \$list. This \$function can facilitate use of the wrap-around trace table for interactive debugging.

\$TRACE2LIST accepts two arguments and returns a numeric code. It is also callable (“[CALLing Sirius \\$functions](#)” on page 44).

The first argument is the list identifier for the output \$list. Data is appended to the end of this \$list. The \$list can be created with the \$LISTNEW function, as documented in the *Sirius Functions Reference Manual*. This is a required argument.

The second argument is a blank-delimited set of options. Valid options are:

- CLEAR** Indicates that TRACE2LIST is to delete all the data in the wrap-around trace table after extracting it. By default \$TRACE2LIST will leave the wrap-around trace table intact.
- CONT** Indicates that when TRACE2LIST has to split a line onto multiple \$list items (because the width indicated by the WIDTH parameter is exceeded), it will end each continued line with a hyphen (-) character. By default, \$TRACE2LIST does not place a hyphen at the end of continued lines.
- DATE** Include date of trace entries in the output \$list. By default, only the time of the entries is included.

- MAXREC** This must be followed by an integer value. It indicates the maximum number of trace entries to return. By default, all trace entries in the trace table are returned.
- NOTIME** Do not include the trace entry time stamps in the output \$list. By default, the time stamps are included.
- WIDTH** This must be followed by an integer value. It indicates the width of the output data in the output \$list. By default, the output width is the maximum length of a \$list item, that is, the value returned by \$LIST_MAXIL. This parameter is useful to force \$TRACE2LIST to format the trace entries to fit on the debugging user's screen.
- WORD** Indicates that when \$TRACE2LIST has to split a line onto multiple \$list items (because the width indicated by the WIDTH parameter is exceeded), it will attempt to split lines so as not to split a word onto multiple lines. That is, **WORD** means to split lines at blanks.

Extremely long words can still result in mid-word line continuation even if **WORD** is specified. By default, \$TRACE2LIST will split a line at the maximum output line width, whether the split is in mid-word or not.

```
%RESULT = $TRACE2LIST(listid, parms)
```

\$TRACE2LIST Function

%RESULT is set to indicate the number of items added to the \$list. All errors result in request cancellation.

It is permissible to not specify an output \$list ID, if and only if the **CLEAR** parameter is specified. Specifying the **CLEAR** parameter with no output \$list ID is the most efficient way to clear the wrap-around trace table if the current contents are not required. Setting the output \$list ID to zero is the same as not specifying that argument.

The following statement retrieves the last 100 entries in the wrap-around trace table to the \$list identified by %OLIST, and then it clears the wrap-around trace table.

```
%RC = $TRACE2LIST(%OLIST, 'MAXREC 100 CLEAR')
```

It is not permissible to specify both the **DATE** and **NOTIME** parameters.

This \$function is new as of *Sirius Mods* version 6.2.

Other SirFact Facilities in Sirius Mods

In addition to the SIRFACT command, the *SirFact* parameters, the User Language statements for *SirFact*, and the *SirFact* \$functions, the following facilities are available as part of the *Sirius Mods* (that is, they are available in the *Model 204* nucleus) when *SirFact* is authorized.

9.1 Comment-initialized global variables

If a global variable begins with the backslash character (\), its initial value, (that is, the value if the variable does not have a value) is a single asterisk (*) when used for dummy string substitution (?&). These are also called “backslash globals.” The initial value for backslash globals returned by the \$GETG function is unaffected by this feature (that is, the initial value is the null string).

Backslash globals are provided primarily to allow you to place *Model 204* commands and User Language statements in your applications under the control of a backslash global; these commands and statements are disabled **unless** the global is set to the null string (or some other non-asterisk value). This approach is more convenient than having to explicitly set the global variables to an asterisk in order to disable the commands and statements; it also reduces GTBL requirements.

If you want to insert various statements (for example, ASSERT statements) in your User Language applications, leaving them disabled until a global variable is reset, you can use a backslash global as the first character of the statement.

For example, the following ASSERT statement is disabled by default:

```
?&\INPCHK ASSERT %INPUT GT 0
```

But the statement can be enabled by

```
BEGIN
%X = $SETG('\INPCHK', '')
END
```

This prefixing technique can also be useful in many applications other than *SirFact*.

Note: To have the initial value of backslash globals become the null string, set the X'04' bit of the SIRFACT system parameter to 1. With this bit setting, global variables whose names start with a backslash are treated the same as any other global variable.

 CHAPTER 10 *The FACT Subsystem*

The diagnostic dumps produced by *SirFact* are stored in base-64 encoded format in *Model 204* procedures inside *Model 204* files. Not readable in raw format, the dumps must have diagnostic information extracted from them with the \$FACT_* functions described in the previous chapter.

While users can write their own system using the \$FACT_* functions, Sirius provides the FACT subsystem, a ready-to-run, full-screen interface for viewing *SirFact* dumps. The FACT subsystem can be accessed via the command line by specifying the file and procedure names in this format:

```
FACT <filename> <procname>
```

For example:

```
FACT SAMPLFIL DUMP.PROC86.20040603120114
```

The FACT subsystem can also be invoked from a subsystem that displays lists of procedures containing *SirFact* dumps. To transfer control from a local subsystem to FACT, use the standard APSY transfer facility (the XFER global). Make sure to:

1. Set the global variable `FACT.RETURN` to the name of the APSY to which FACT should return control.
2. Set the global variable `COM` to the source file and source procedure containing the *SirFact* dump.

An example follows:

```
%DUMMY = $SETG('COM',%FILENAME WITH ' ' WITH -
           %DUMP_PROC_NAME)
%DUMMY = $SETG('NEXTPROC','XFER')
%DUMMY = $SETG('XFER','FACT')
%DUMMY = $SETG('FACT.RETURN','MYAPSY')
STOP
```

The *SirPro* subsystem is already configured to perform this transfer when an "F" prefix command is entered on the *SirPro* Procedure List screen:

```
----- FILE: ALANPROC ----- ULSPF504/4.1.1H/CMS ----- 04-07-07 12:03:37 -----
==>                                     Total Procs = 422
Se1   Procedure Name                    Account   Bytes   Date   Time
  1   PUBLIC/HEADER.GIF                 ALAN     3072   04/06/30 16:44:43
  2   PUBLIC/INDEX.HTML                 ALAN      429   04/06/30 16:44:25
  3   PUBLIC/HEADER.HTML               ALAN      217   04/06/30 16:44:13
  4   VERISIGNSEALMAROON.GIF           ALAN    30498   04/06/30  8:56:11
  5   VERISIGN.TXT                     ALAN     4844   04/06/30  8:31:46
  6   TEXT.CONVERSION                  ALAN     1480   04/06/25 14:09:59
  7   GOLDEN.TXT                       ALAN     5948   04/06/24 11:27:44
  8   CREATE.GROUP4                    ALAN      110   04/06/23  9:55:27
  9   SIRFACT2.TXT                     ALAN     1272   04/06/21 13:40:24
 10   DUMP.PUPR-EDIT2.20040525172149   ALAN    168232  04/06/15 17:25:15
 11   DUMP.O.20040526090031            ALAN     1596   04/06/15 17:25:09
F 12   DDD.20040607161906.DEVPRO       ALAN    137627  04/06/15 17:25:02
 13   DUMP..20040610173021            ALAN      990   04/06/15 17:24:55
 14   TEST.HTML                       ALAN     1872   04/06/06 16:48:14
 15   RAILROAD.HTML                   ALAN      738   04/06/06 14:32:55
 16   LAMEDUCK.JPG                     ALAN    19952   04/06/06 14:25:30
 17   CONTROL.ALANPROC                ALAN      62    04/06/02 11:56:36
-----
1/Help  2/Sort-Name  3/Quit  4/Sort-User  5/Sort-Date  6/Sort-Size
7/Up    8/Down       9/Repeat 10/Refresh 11/*UPPER 12/FULLNAME
```

SirPro Procedure List Screen

The *SirPro* interface is further documented in the *SirPro User's Guide*. The F prefix command in *SirPro* only works for sites that own *SirFact*. If a non-*SirFact* procedure is selected with an F, an error message is posted to the screen.

Whether the FACT system is accessed from the command line, from *SirPro*, or from a local subsystem, the main screen displayed is the same: a scrollable display of the basic dump information, extracted from the dump. As further commands are entered on the screen's command line, their output is appended to the display, and the user can scroll through the list. If a printed version of the output is required, **PF12** can be used to send the list to a print device.

```

----- Fact ULSPF504-----04/07/07 12:03:48
==>                                     Line: 1      Cols: 1   To 79
-----Context: Error time context-----
I.TIME           = '2004/05/25 17:21:49'
I.ERRMSG         = 'MSIR.0510: Cancelling request because of SIRFACT CANCEL
I.SUBSYS        = 'SIRPRO'
I.FILE          = 'SIRPRO'
I.PROC          = 'PUPR-EDIT2'
I.JOBNM         = 'ULSPF602'
I.STEPNM        = '046864'
I.JESID         = ''
I.USERID        = 'ALEX'
I.ACCOUNT       = 'ALEX'
I.TERMID        = 'ALEX2'
I.USERNUM       = '18'
I.IODEV         = '41'
I.WHAT          = 'Evaluating'
I.WHERE         = 'Error at line 573 of proc PUPR-EDIT2 in file SIRPRO'
I.CALL          = 'At main level: no subroutine calls'
I.FORLEV        = '0'
I.VERSIONS      = 'M204:5.3.0C   Sirius Mods:6.5 Dump:6'
-----
1/Help      2/Scale on  3/Quit      4/PFkey off
7/Up        8/Down       9/Repeat    10/Left     11/Right    12/Print

```

FACT Diagnostic Dump Extraction Screen

The initial FACT screen shows default context information for the error condition, including a date/time stamp, source procedure and file, userid of the user for whom the error occurred, etc. Additional information is extracted from the dump via D[isplay] or C[ontext] commands. The output of each command is appended to the list of previously extracted data. The list of all this formatted dump information is scrollable and printable from the FACT scan screen.

10.1 C[ontext] command: Setting program context for data extraction

Information in a *SirFact* dump is categorized by context within the program that is executing when the error occurs. The “context” is either a complex subroutine or the main body of the executing program. You can switch contexts within the dump file via the C[ontext] command. The D[isplay] output (described below in “[D\[isplay\] command: Displaying data from the SirFact dump](#)” on page 60) is sensitive to the context set by the Context command for the values and attributes of %variables, screen items, and image items.

Valid information contexts are the same as those documented for the \$FACT_CONTEXT function (“[\\$FACT_CONTEXT](#)” on page 46):

- * (asterisk) Switches FACT context to the mainline of the executing program.
- . (period) Switches to whichever context the program was in at the time of the *SirFact* error.

<subroutine name> Switches to the context of the named complex subroutine. Simple subroutines are not valid contexts, as they share the same context as the main body of the program.

The Context command adds a dividing line to the output list, and any D[isplay] information that follows applies to the current context until the context is switched again.

10.2 **D[isplay] command: Displaying data from the SirFact dump**

The Display command requests for specific kinds of data to be extracted from the current context in the *SirFact* dump.

The Display command specifies a series of data items. It is formatted as follows:

```
D[isplay] [X.]c[.s] [X.]c[.s] ...
```

where:

- X** An optional flag specifying that the requested data should be displayed in hexadecimal format
- c** The class of data requested.
- s** An optional selection pattern for data of class *c*.

For example:

- To display global variables beginning with the string "STR": `DISPLAY G.STR*`
- To display context information (user, job, program, error, etc.): `D I 1li`. To display the value of %B in hexadecimal:
 - `D X.%B`

The valid *c* values are described in the following list. They are the same as for the \$FACT_DATA function ("[\\$FACT_DATA](#)" on page 47). All can be abbreviated to a single letter, except for `RIN` and `ROUT`, which require two letters. So `V.%JUNK` is the same as `VALUE.%JUNK`, and `G.X*` is the same as `GLOBAL.X*`.

You can enter the Display command and *c* values in any case: all non-quoted values are translated to uppercase before being processed.

ATTRIBUTE Attributes of a %variable, image item, or screen item. For %variables, wildcards are not allowed, and the percent character (%) must be present in the variable name (as in `A.%I`). `ATTRIBUTE` returns

information like variable type, variable length and DP value, and, for object %variables (as of version 6.5), the object class name.

FIELD

Value of one or more fields in active FOR EACH RECORD loops. FIELD can be followed by a FOR EACH RECORD loop nesting level, where 0 is the innermost nesting level, -1 is the next innermost, and so on. FIELD, or the nesting level, or both, can then be followed by a fieldname, which can contain wildcards.

The nesting level is only necessary to distinguish like-named fields in different FOR EACH RECORD loops, though they will be distinguished on display anyway.

Examples of FIELD specifications follow:

- | | |
|-----------------------------|--|
| <code>FIELD.FOO(3)</code> | returns the value(s) of the third occurrence of field FOO in every active FOR EACH RECORD loop |
| <code>FIELD.0.BAR(*)</code> | returns all occurrences of field BAR in the innermost (active) FOR EACH RECORD loop |
| <code>FIELD.-1.*</code> | returns all fields in the second innermost (active) FOR EACH RECORD loop |
| <code>FIELD.*</code> | returns all field values in all active FOR EACH RECORD loops |

GLOBAL

Value of one or more global variables. Wildcards are allowed.

The selection pattern can be left off, so `GLOBAL` or `G` is the same as `GLOBAL.*`

INFO

The following informational entities are available from the dump, displaying values that were in effect at the time the dump was taken.

Although they are displayed by default at the beginning of the FACT subsystem dump output (see the second Figure in “[The FACT Subsystem](#)” on page 57), you can dynamically request an entity's display by entering at the command prompt the entity name preceded by `INFO.` (optionally preceded by `X.`).

Wildcards are allowed for the entities (`D INFO.W*` returns informational data for the WHAT and WHERE entities); and the selection pattern can be left off entirely, returning information for all the entities for the current context (`INFO` or `I` is the same as `INFO.*`).

Each entity returns one or more informational strings.

ACCOUNT	Value of the <i>Model 204</i> ACCOUNT parameter for the active user.
CALL	If applicable, the program location of the active ON unit or active subroutine, one line showing the location of the return point for each outer context.
ERRMSG	Error message that provoked the dump.
FILE	Name of the file that contains the executing procedure.
FORLEV	Nesting level of the active FOR EACH RECORD loop (0 is innermost, -1 is next innermost, and so on)
JESID	For MVS only, the job number assigned the job by the spooling subsystem.
JOBNM	The name on the job card.
IODEV	Value of the <i>Model 204</i> IODEV parameter of the active user.
PROC	Procedure active when the error occurred.
STEPM	The identifier of the step within the job being executed.
SUBSYS	The name of the subsystem within which the error occurred.
TIME	The date and time the error occurred.
TERMID	Value of the <i>Model 204</i> TERMID parameter of the active user.
USERID	Login user ID.
USERNUM	Thread number assigned the user by <i>Model 204</i> .
VERSIONS	The <i>Model 204</i> version, the <i>Sirius Mods</i> version, and the <i>SirFact</i> internal dump format version in effect when the dump was produced.
WHAT	General activity category (<i>Model 204</i> WHAT flag) of user when error occurred.
WHERE	Location within the code at which the error occurred.

LIST Value of one or more \$list items. LIST must be followed by a %variable that contains the list identifier. For specific list items, the list identifier can be followed by a parenthesis and an indication of the range of item numbers and columns required.

Example LIST specifications follow:

LIST.%LIST(5) returns list item number 5 in the \$list identified by %LIST

LIST.%LIST(3-4,10-29) returns columns 10 through 29 in items 3 through 4 in a list identified by %LIST

LIST.%LIST(3.2,10.20) returns columns 10 through 29 in items 3 through 4 in a list identified by %LIST

If a list identifier is in an array, the array item number must be specified and cannot be a range. For example, LIST.%LISTA(9)(10) requests list item 10 in the list identified by %LISTA(9).

An asterisk wildcard is not allowed (D LIST.* is not valid). See [“Additional syntax for VALUE and LIST” on page 64](#) for the complete syntax used with \$lists.

RIN The number of records and some of the record numbers in a found set label. The count of records in the found set is always returned, and the number of record numbers returned is the minimum of the number of records in the found set and the SIRFACT RECNDUMP ([“SIRFACT RECNDUMP” on page 25](#)) value at the time the dump was taken.

Wildcards are not allowed.

RON The number of records and some of the record numbers on a LIST. The count of records in the LIST is always returned, and the number of record numbers returned is the minimum of the number of records in the LIST and the SIRFACT RECNDUMP ([“SIRFACT RECNDUMP” on page 25](#)) value at the time the dump was taken.

Wildcards are not allowed.

TRACE All or a specified number of the entries in the *SirFact* wrap-around tracetable in CCATEMP (see [“The TRACE Statement” on page 35](#)). TRACE can be followed by the number of entries to return: T.* and TRACE return all entries in the trace table; and T.20 returns the last 20 entries.

VALUE Value of a %variable, image item, or screen item. For %variables, wildcards are not allowed, and the percent character (%) must be present in the variable name (as in VALUE.%I).

The term `VALUE` can be left off, so `%I` is the same as `VALUE.%I`.

Note:

- You must know the variable names you are looking for: *SirFact* cannot process a `V.*` command to show all variables that exist in a program or within a complex subroutine, because the variable names are not available to the program at runtime in plain text format.
- Image and Screen variables are not collected until they have been Prepare'd, Identify'd, or Read. If *SirFact* replies that the specified Image or Screen variable does not exist, then it was not referenced at the time of the *SirFact* dump.

See “[Additional syntax for VALUE and LIST](#)” for additional syntax that can be used for arrays and objects. The support for object `%variables` began in version 6.5.

After the `D` command is executed, a separator line is output to the screen, and the output of the command is appended to any previous output. FACT automatically scrolls the screen to the top of the new information.

The returned values from the `D` command contain the requested data item followed by the value. The class requested is shown in its minimum abbreviated form, except for the `VALUE` class, which is omitted.

10.2.1 Additional syntax for VALUE and LIST

In addition to simply specifying a `%variable` in a `VALUE` or `LIST` display, there is syntax that can (or must) be used for arrays, structures, \$lists, and objects. This section explains that syntax.

Arrays

Array variables must be followed (in parentheses) by as many subscript indicators as the dimension of the array. You specify a single digit or a range for each subscript; you can use the following:

- a dash (-), to indicate a range of array elements
- a period (.), to separate a starting subscript and number of elements
- an asterisk (*), to indicate all of, or the rest of, a dimension

For example:

- `VALUE.%DATA(11)` returns array item 11 in array `%DATA`
- `VALUE.%DATA(*)` returns all array items in array `%DATA`

- `VALUE.%DATA(5-*)` and `VALUE.%DATA(5.*)` return array items 5 through the last in array `%DATA`
- `VALUE.%DATA(7-11)` and `VALUE.%DATA(7.5)` return array items 7 through 11 in array `%DATA`.

These range qualifiers can be used in multiple dimensions as in `VALUE.%MULTI(*,6-12,2.3)`.

Sirius \$lists `$List` items use a similar range and item number specification as that used for `%variables`. Currently you can only display information about `$lists` whose identifiers are stored in `%variables`:

```
D L.%LIST(1-5)
D L.%LIST(20.2)
```

If `%LIST` is an array of `$list` identifiers you must specify the array item number and then the range of values to extract from the list, like this:

```
D L.%LIST(5)(1-5)
D L.%LIST(2)(20.2)
```

SirFact currently doesn't support range for the array items in this case.

To see only specific columns of `$list` items, add a second range as in

```
D L.%LIST(1-5, 1.20)
D L.%LIST(20.2 100-110)
```

To see the number of `$list` items, use a single “subscript” of a number sign:

```
D L.%LIST(#)
```

For a single `$list` item, without a column restriction, you can also use a colon and the item number as a suffix; to see the number of items, you can use the “:#” suffix. For example, to see item number 3, and to see the number of items:

```
D L.%LIST:3
D L.%LIST:#
```

Structures You display the value of structure variables by specifying the name of the variable after the structure. For example, to display the value of structure variable `VAR` in structure `%STRUCT`, you can use the following:

```
D %STRUCT:VAR
```

If the structure variable is an array, use the array syntax described on the previous page. For example, the following displays the value of items 7 through 11 in array variable DATA of structure %STRUCT:

```
D %STRUCT:DATA(7-11)
```

Support for displaying structures is available as of version 6.7 of *Sirius Mods*.

Objects

The format for displaying the contents of a *Janus SOAP* system or user class member (variable or method) is as follows:

```
D %object:member[(parms)]
```

This format is very nearly the same as the format used in a User Language program to access the objects. Some examples follow:

```
D %MG:MILEAGE
```

Displays the value of a (user class) public or private variable: the content of class variable MILEAGE, when %MG is a user class object.

Note: If MILEAGE were a shared variable in class CAR, it could also be displayed by the following command:

```
D %(CAR):MILEAGE
```

This use of the class name in parentheses instead of an object variable is valid for system class as well as user class shared variables.

```
D %RS:COUNT
```

Displays the value of a system class member: the return from the Count method, when %RS is a Recordset object variable, for example.

```
D %SL:ITEM(5)
```

Displays the value of a system class member: the return from the Item method applied to the fifth item, when %SL is a Stringlist object variable, for example.

```
D %NAMES('Bush'):FIRSTNAME
```

Displays the value of a (user class) NamedArraylist collection method: the content of the return from the FIRSTNAME method, when %NAMES('Bush') is a NamedArraylist collection item.

D %NAMES('Bush'):LIST:COUNT

Displays the value of strung-together class members: the content of the return from the COUNT method, when %NAMES('Bush') is a NamedArraylist collection item of a user class, and LIST is a variable that is a Stringlist object.

D %DOC

Displays the value of an object variable: the content of %DOC, when %DOC is an instance of a system XmlDocument object.

The information displayed always includes whether or not the %variable is Null. If the object is not Null, the output from the Print method applied to the object is automatically included (for some classes, such as the Stringlist class or the XmlDocument class). For an XmlNodeList object variable, for example, which has no Print method in its class, the FACT display simply includes whether or not the variable is Null.

As an example, if %L is a Stringlist object, a `d %l` command might produce:

```
%L                = Not null
%L:PRINT          = 'This is a test 1'
                  = 'This is a test 2'
                  = 'This is a test 3'
```

This example also demonstrates that the *SirFact* display

- Uppercases non-quoted command input before processing (this is adjustable if you are using the *SirFact* \$functions to look at dumps — see “[\\$FACT_OPTION: Set or get SirFact display options](#)” on page 50).
- Quotes the output values but not informational or error messages
- Uses the “terminal output” form (adding equal signs and quotation marks) for the Print method result and not its normal format.

This Print output is truncated at 500 lines, a limit that is adjustable if you explicitly specify the Print method (see the following “Note”), or if you are using the *SirFact* \$functions to look at dumps (see “[\\$FACT_OPTION: Set or get SirFact display options](#)” on page 50).

Note: Some methods that produce terminal output (such as most Print methods or, for example, the *Janus SOAP XML Serial*

method) can be invoked explicitly in the Display command. For example, if %xmlPier is an XmlDoc object, the following commands will provide the same information:

```
D %XMLPIER
D %XMLPIER:PRINT
```

Advantages of invoking a Print method explicitly include:

- You can specify Print method parameters:

```
D %XMLPIER:PRINT('/outer/inner')
```

- The output is not subject to the 500-line (default) limit of an “implied” Print.

Although object contents can be accessed from *SirFact* very much the same as they are accessed in User Language, the following restrictions apply:

- The special range, count, and wildcard syntaxes supported for some other display types are not supported for objects. For example, to request the display of the first three items in Stringlist %SL, you **cannot** use `D %SL(1-3)`. You can get the result you want, however, by using Stringlist Print method parameters, as in:

```
D %SL:PRINT(, ,1,3)
```

- Not all system methods may be invoked in a Display command. In general, methods that update an object or that create a new object instance are not allowed. For example, if %SL is a Stringlist object, `D %SL:ADD('A new item')` is **not** allowed.

10.3 Navigation and display commands

The command descriptions follow:

PFKEYS ON/OFF	Toggles the display of the PFKey labels.
SCALE ON/OFF	Toggles the display of a scale line.
RIGHT X	Shifts the display to the right X columns.
LEFT X	Shifts the display to the left X columns.

F xxxxx or /xxxxx	Finds the string xxxxx in the formatted audit trail data, beginning on the current line.
-F or -/	Same as above but performs the search backwards from the current line
M (with PF7,8,10,11)	Moves the screen to the Top, Bottom, 1st column or last depending upon PFKey pressed.
L xxx	Moves the current line to the requested line number xxx.

10.4 Program function keys

The key descriptions follow:

PF1	Display help information for the current screen.
PF2	Toggle on/off the horizontal scale.
PF3	Exit the current screen and return to the previous screen or menu.
PF4	Toggle on/off the display of pfkeys - when pfkeys are not displayed the extra two lines are used to display data.
PF7	Scroll back to the previous page.
PF8	Scroll forward to the next page.
PF9	Repeat the last valid command entered in the command window.
PF10	Scroll left.
PF11	Scroll right.
PF12	Print the extracted data. This key passes the list of retrieved diagnostic data to an internal print routine and presents a print option screen to the user.

10.5 The FACT Print Screen

```

----- SirFact Print 04-07-07 12:04:11 -----
==>

SirFact Dump from ULSPF504 source: DUMP.PUPR-EDIT2.20040525172149
Enter Y (yes) or N (no) to use the above messages as the print header:

      You were viewing line 1      of      16
      Print from line 1      to line 16

Destination ==> (Select from options 1 through 3 below)

1. Dataset DDNAME      ==> OUTALAN
2. Printer ID          ==> ALAN
3. $PRINT Class        ==> X

      with      ==> COPIES=1
      ==>

      Lines Per Page    ==> 60 (UDDLPP)      Record Format    ==> 12 (UDDRFM)
      Characters per Line ==> 133 (UDDCCC)      Header Control  ==> (HDRCTL)

-----
1/Help          3/Quit
  
```

FACT Print Screen

The Print screen allows the user to specify how to route the formatted dump extract to a print device.

To print, enter the following information in the screen fields:

- Print from and to line(s)** The starting and ending lines within the SirFact extract to be printed.
- OUTPUT DESTINATION** The destination of the procedure. Select 1 to print to a the printer named in the PRINTER field, 2 to output the procedures to the file named in the FILE field, and 3 to invoke a USE \$PRINT, to the specified CLASS and PRINTER.
- PRINTER** A valid predefined printer name. If this field is blank or invalid when a 1 is specified in the OUTPUT DESTINATION field, the print comes to the user's terminal.
- OUTFILE** A valid preallocated sequential file.
- CLASS** The job class to use for USE \$PRINT requests. This job class should be a valid class at your site. USE \$PRINT is an obsolete method of printing, and its use is discouraged. If the PRINTER is blank or invalid when a USE \$PRINT is invoked, the print is routed to PRINTER=LOCAL by *Model 204*.

WITH criteria	Enter in these two fields any print routing criteria normally placed in the “WITH” clause of a USE statement (such as COPIES=xx, TAG or ID statements).
UDDLPP	The number of lines to be printed or displayed per page. The default is 60 lines per page.
UDDCCC	The number of characters to be printed or displayed per line. The default is 133 characters per line.
HDRCTL	Header control settings allow the user to suppress various levels of automatic print headers.
UDDRFM	The number of characters to be printed or displayed per line. The default is 133 characters per line.

Function keys active on the Print screen:

- PF1** Display help information for the current screen.
- PF2** Toggle on/off the horizontal scale.
- PF3** Exit the current screen and return to the previous screen or menu.

APPENDIX A *Date Processing*

This chapter presents date processing issues, including usage of *SirFact* past the year 1999, an explanation of its processing of dates, and any rules and restrictions you must follow to achieve correct results using date values with *SirFact*. *SirFact* uses dates in the following ways:

- to examine the CPU clock (as returned by the STCK hardware instruction) to determine the current date, in case *SirFact* is under a rental or trial agreement
- to set the procedure name for a *SirFact* dump to contain the date and time at which the dump was created. These timestamps always use a four-digit year.
- to display the current date, as returned by the TIME SVC (if the *Sir2000 User Language Tools* is active, the date is obtained by temporarily switching the clock using the APPDATE INTERNAL command) as page headers in various end-user displays

To correctly use *SirFact* past the year 1999, *UL/SPF* version 5.4 and *Sirius Mods* version 5.4 or later are required. For headers on pages or rows that occur on printed pages or displayed screens, Sirius Software products generally use a full four digit year format, although they may display dates with two digit years in circumstances where the proper century can be inferred from the context.

Above and beyond the post-1999 requirements specific to *SirFact*, you must examine all uses of date values in your applications to ensure that each of your applications produces correct results. Furthermore, both the operating system and *Model 204* must correctly process and transmit dates beyond 1999 in order for *SirFact* to operate properly.

APPENDIX B *Terminal MODEL 6 Support*

As of *Sirius Mods* version 6.8, *SirPro* users can take advantage of the additional available screen space offered by terminal models beyond the standard Mod 2 (24 X 80), Mod 3 (32 X 80), Mod 4 (43 X 80), and Mod 5 (27 X 132). The new terminal models are supported by setting the terminal model to 6:

RESET MODEL 6

There's really no such thing as a Model 6 terminal, but setting the terminal model to 6 tells *Model 204* to issue a Write Structured Field Query to the terminal to have the terminal indicate its geometry (number of rows and columns) to *Model 204*. In this way, *Model 204* can dynamically set a terminal's geometry, whether it's one of the standard geometries (Mod 2, 3, 4, or 5) or not. Many terminal emulators allow alternate 3270 sizes to be set. This makes it possible to set the terminal geometry to match the optimal combination of font size and physical screen size for a particular workstation, rather than trying to set the emulator font size to work well with one of a limited number of screen geometries.

Unfortunately, the standard User Language screen definitions don't allow the defining of fields that extend beyond column 79. However, \$scrHide, \$scrSize, and \$scrWide make it possible for User Language screens to take advantage of columns beyond column 79. In addition, these functions make it possible to dynamically modify screen definitions to allow a single screen definition to work with an arbitrary variety of screen sizes. While these functions are a bit awkward to use and somewhat limited, they're not unreasonable for building dynamic scrolling screens — scrolling screens being particularly suited for larger screen geometries.

To facilitate User Language applications for varying screen sizes, the VIEW command for the MODEL parameter is enhanced to show the screen geometry in addition to the model number for model 6 terminals:

```
> V MODEL
MODEL      6 34*142   TERMINAL MODEL
```

So \$view issued for the above terminal returns 6 34*142, from which a User Language application could readily determine that the screen has 34 rows and 142 columns.

To enable model 6 support, the SIRTERM system parameter must be set in the CCAIN stream. This is a bitmask parameter with the following meanings for the bits:

X'01' Enable MODEL 6 support.

X'02' Always issue a Write Structured Field Query for terminals connecting to *Model 204* through VTAM. This allows *Model 204* to dynamically determine the screen geometry of any terminal connecting to it through VTAM, without having to issue a RESET MODEL 6 command.

The downside of this setting is that it could add a small amount of time to the initial connection process and a slight amount of extra network traffic.

If a terminal is using a non-standard screen geometry via model 6 support, the *Model 204* editor and command line will correctly use the available screen space.

Index

\$

\$functions, *see* *SirFact* \$functions

A

ASSERT statement ... 27-28

 testing assumptions ... 28

B

Backslash global variables ... 55

C

Callable \$functions ... 44

CANCEL option, SIRFACT DISPLAY ... 14

CANCEL subcommand, SIRFACT
 command ... 11

Canceling requests ... 11, 14

 development vs. production ... 11

 setting cancellations ... 14

CASE option, \$FACT_OPTION ... 50

Comment-initialized global variables ... 55

Context command, FACT Subsystem ... 59

D

DEBUGUL parameter, *Model 204* ... 39

Display command, FACT Subsystem ... 60

DISPLAY subcommand, SIRFACT
 command ... 14

DUMP option, SIRFACT DISPLAY ... 14

DUMP subcommand, SIRFACT
 command ... 16

Dumps, limiting number of ... 22

Dumps, *SirFact* ... 16, 21-22, 25-26, 28, 47, 50,
 60

E

Errors, ignoring ... 21

F

FACT subsystem ... 57

Fact Subsystem PF keys ... 69

G

Global variables ... 55

 Backslash global variables ... 55

 Comment-initialized global variables ... 55

 \ global variables ... 55

I

IGNORE option, SIRFACT DISPLAY ... 14

IGNORE subcommand, SIRFACT
 command ... 21

IMPLIM option, \$FACT_OPTION ... 50

M

MAXDUMP option, SIRFACT DISPLAY ... 15

MAXDUMP subcommand, SIRFACT
 command ... 22

Mixed-case User Language ... 43

MODEL parameter, VIEW command ... 75

P

Performance issues ... 20

Printing ... 70-71

 \$PRINT CLASS ... 70

 Characters per line ... 71

 Header control ... 71

 Lines per page ... 71

 OUTFILE ... 70

 output designation ... 70

 PRINTER ... 70

 range specification ... 70

 Record format ... 71

 WITH criteria ... 70

Procedure updates, subsystem ... 23

Q

Quads, *Model 204* ... 38
QUIESCE subcommand, SIRFACT
 command ... 23
Quiescing, subsystem ... 23-24

R

RECNDUMP ... 25
RECNDUMP option, SIRFACT DISPLAY ... 15
RECNDUMP subcommand, SIRFACT
 command ... 25
Record numbers, dumping ... 25
Recovery ... 20-21
 checkpoint logging ... 20
 recovering dumps ... 21
 roll-forward ... 20
RESUME subcommand, SIRFACT
 command ... 24

\ global variables ... 55

S

Sdaemon, *SirFact* ... 43, 49
SIRAPSYF parameter ... 40
SirFact \$functions
 \$FACT_CMD ... 45
 \$FACT_CONTEXT ... 46
 \$FACT_DATA ... 47
 \$FACT_DONE ... 48
 \$FACT_INIT ... 49
 \$FACT_OPTION ... 50
 \$FACT_VNAME_WIDTH ... 51
 \$TRACE2LIST ... 52
SIRFACT command ... 9
 CANCEL subcommand ... 11
 DISPLAY subcommand ... 14
 DUMP subcommand ... 16
 IGNORE subcommand ... 21
 MAXDUMP subcommand ... 22
 NODUMP subcommand ... 16
 QUIESCE subcommand ... 23
 RECNDUMP subcommand ... 25
 RESUME subcommand ... 24
 SNAP subcommand ... 26
SIRFACT parameter ... 37
SIRFACT statement ... 31
SIRTERM system parameter ... 75

SNAP subcommand, SIRFACT command ... 26
Subsystem procedure, updates to ... 23

W

Wildcard characters, SIRFACT command ... 9