
Sirius Mods Release Notes Version 6.3

October, 2002



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:

- *Janus SOAP*
- *Janus Sockets*
- *Janus Web Server*
- *SirFact*
- *SirSafe*

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	ii
Contents	iii
Chapter 1: Introduction	1
Chapter 2: Maintenance and Support	3
Chapter 3: All or Multiple Products	5
Chapter 4: Janus SOAP	7
Chapter 5: Janus Sockets	9
Chapter 6: Janus Web Server	11
Compression	11
Enhanced wildcard processing in JANUS WEB ON rules	11
Error handling	12
HEADJS and ONLOAD parameters on SCREEN rules	12
SCREENURL/NOSCREENURL parameters	13
\$WEB_NOCACHE function	13
\$WEB_SAVE_xxx and \$WEB_REST_xxx enhancements	13
Chapter 7: Sirius Functions	19
\$FIELD_LISTI and \$FIELD_IMAGE enhancements	19
\$lists as parameters to ECF programs	20
\$LIST_ADD_ORDERED, \$LIST_ADD_UNIQUE and \$LIST_ADD_UNIQUE_ORDERED	21
\$LSTR_GET_IMAGE and \$LSTR_SET_IMAGE	22
Session \$functions	23
\$SESSION_CREATE	24
\$SESSION_DELETE	25
\$SESSION_OPEN	25
\$SESSION_CLOSE	26
\$SESSION	26
\$SESSION_ID	26
\$SESSION_OWNER	27
\$SESSION_TIMEOUT	27
\$SESSION_LIST	27
Session \$lists	28

Global and session LONGSTRINGS	28
\$SIR_WILD	30
Chapter 8: SirSafe	31
Shared DASD support for Sysplex and LPARs	31
Shared DASD Background and Overview	31
Global Resource Serialization (GRS)	33
SIRENQ	34
Modified Shared DASD List Maintenance	35
Enhancements to the AUTHCTL Command	35
\$SIR_CHECK_ACCESS function	36
Chapter 9: Compatibility/Bug fixes	39
Janus Sockets	39
\$SOCK_SENDx argument values longer than 255	39
\$SOCK_RECV result	39
Sirius Functions	39
\$COMMAND and \$COMMNDL output width parameter	39
Fixes in Sirius Mods 6.3 but not in 6.2	40
Version co-requisites	40

CHAPTER 1 ***Introduction***

This document lists the enhancements and other changes contained in the newest release of *Sirius Mods*: version 6.3. The previous generally released version of *Sirius Mods*, 6.2, was released in May, 2002.

CHAPTER 2 *Maintenance and Support*

Drop support for *Model 204* back versions

As of version 6.3, *Sirius Mods* no longer supports *Model 204* version V4R1.0.

CHAPTER 3 *All or Multiple Products*

Note that in addition to changes to the features and functionality of the *Sirius Mods*, the documentation is constantly being improved. Documentation changes are shown at a section in the preface of each manual.

The following features are new or changed in *Janus SOAP*:

<!DOCTYPE ...>

This phrase is now allowed for the deserialization \$functions, using the DTD_IGNORE option.

Control of <?xml version...?>

The serialization \$functions, and \$XML_PRINT, now have options to either allow or suppress the “xml declaration”.

Other \$XML_PRINT options

\$XML_PRINT provides an INDENT option to control indentation width.

CDATA The <![CDATA[...]]> clause is now supported.

XMLDoc generating/updating

The \$XML_INSATT, \$XML_INSCOM, and \$XML_INSPI functions are now supported. In addition, there are no restrictions to the order in which a document is created. The \$XML_INSTEXT function is supported, but it is restricted to not allow insertion of adjacent TEXT nodes.

Global XMLDocs

The \$XML_DOC_GLOBAL and \$XML_DOC_GLOBAL_DEL functions allow you to create, retrieve, and delete global XMLDocs, so that their content can be preserved across User Language requests.

Session XMLDocs

The \$XML_DOC_SESSION and \$XML_DOC_SESSION_DEL functions allow you to create, retrieve, and delete session XMLDocs, so that their content can be preserved over a logical session that can span multiple logins and logouts.

UTF-8/16 Initial Unicode support is provided in this version:

- UTF-8 and UTF-16 encoding may occur in “inbound” streams (\$XML_STR2DOC and \$WEB_XML_RECV).
- UTF-8 is **always** generated for “outbound” XML (\$XML_DOC2STR and \$WEB_XML_SEND).

PI targets A processing instruction **target** (conceptually, the “name” of a PI) can now contain any character as described by the XML standard.

Text node length

A text node's length is now limited only by the maximum size of a \$list and the other data in a document, all of which are stored in a single internal \$list (about 3.5 gigabytes is the limit of a single \$list).

Whitespace handling

For TEXT nodes in most business-to-business applications, leading and trailing whitespace should be removed, and multiple whitespace characters should be combined as one. This is the default behavior of the deserialization \$functions (\$XML_STR2DOC and \$WEB_RECV_XML); it can be over-ridden (to preserve all whitespace in TEXT nodes) with the WSP_PRESERVE option.

XML parse errors

The information presented by an error deserializing an XML document (\$XML_STR2DOC and \$WEB_RECV_XML) shows where the error occurred in the serialized document character stream, including the fragment of the stream near the error with a “pointer” showing the current parsing location.

XPath parse errors

The information presented by an error parsing an XPath expression now is specific to what the parser expected, replacing the generic “Invalid XPath expression” message.

Further improvements to performance and scalability

Various internal changes have been made to facilitate more intensive XML and XPath processing.

The following features are new or changed in *Janus Sockets*:

LONGSTRING support

All *Janus Sockets* \$functions are now LONGSTRING capable. Most important in this regard is that \$SOCK_SEND can send a LONGSTRING containing more than 255 bytes in a single call, \$SOCK_RECV and \$SOCK_RECVPRS can receive more than 255 bytes in a single call and \$SOCK_TRAN_IN and \$SOCK_TRAN_OUT can translate more than 255 bytes in a single call.

\$SOCK_SSL_ON

This function makes it possible to make an initial connection from a *Janus Sockets* client application to a proxy server, use the CONNECT method to connect to a server through the proxy and then communicate via SSL with the back end server, simply having the proxy server act as a “tunnel” for the SSL data. Basically, the \$SOCK_SSL_ON should be used after a positive response has been received for a CONNECT method request on a *Janus Sockets* client socket. For more information on HTTP tunneling and the CONNECT method see <http://www.ietf.org/rfc/rfc2817.txt>. Many other references are also available on the web.

The following features are new or changed in *Janus Web Server*.

6.1 Compression

Janus Web Server can now use *deflate* compression upon request. Compression can be indicated on the JANUS DEFINE command, JANUS WEB ON rules and \$WEB_SET. The compression value can be 0 meaning to use no compression, 1 meaning to use deflate duplicate compression and 2 meaning to use duplicate compression and Huffman character encoding. Data compression requires a not insignificant amount of CPU (COMPRESS 2 requiring more than COMPRESS 1) and an extra 64K of virtual storage for each thread on any web port with compression enabled. For example, a port with compression enabled and 20 threads would require an extra 20*64K or about 1.3M of extra virtual storage for the compression buffers. The advantage of compression, of course, is that compressed data will use less network bandwidth, anywhere from 30% less to 80% less, depending on the datatstream. Images tend to be in a compressed format already so generally do not compress any more and are not compressed by *Janus Web Server*.

To enable deflate compression on a port, the COMPRESS parameter must be specified on the JANUS DEFINE command for the port. If this is not done, compression could be requested in JANUS WEB ON rules or with \$WEB_SET but is ignored. If some compression is required but the port default should be no compression, simply specify COMPRESS 0 on the port definition and set compression on a URL basis in JANUS WEB ON rules or on an application basis with \$WEB_SET.

6.2 Enhanced wildcard processing in JANUS WEB ON rules

Janus web ON rules now allow wildcard substitution in FILE and GROUP names in the OPEN clause. This substitution always occurs before substitution in the CMD or SEND clause. For example, with the rule

```
JANUS WEB WEBPORT ON /TEST?/* OPEN FILE QA* -  
      CMD 'I *'
```

the URL */test5/queequeg* would result in the file QA5 being opened and procedure QUEEQUEG inside QA5 being INCLUDED.

In addition, variable data from a URL can now be substituted out of order by indicating the wildcard match position after a double-quote character. For example, with the rule

```
JANUS WEB WEBPORT ON /*/TEST?/* OPEN FILE TEST
      CMD 'I MOBY"2,*'
```

the URL `/xyz/test5/tashtego` would result in the file TEST being opened and the command “I MOBY5,TASHTEGO” being issued. As this example illustrates, an asterisk after a positional replacement string uses the replacement string after the positional string. It is possible with positional replacement strings to re-use replacement strings. For example, with the rule

```
JANUS WEB WEBPORT ON /TEST* OPEN FILE TEST
      CMD 'I A"1' AND 'I B"1' and 'I C"1'
```

the URL `/test3` would result in the file TEST being opened and the commands “I A3”, “I B3” and “I C3” being issued.

6.3 Error handling

In cases, such as a User Language programming error that cancels the request, *Janus Web Server* automatically generates a web page describing the error. This has been improved so that:

- In addition to the last error message, up to two other messages may be displayed, these messages correspond to the messages that can be obtained by \$FSTERR and SETGRC.
- The response header parameters on this page are set so that it should always be displayed by the browser.

Also, a new section has been added to the *Janus Web Server Reference Manual* which briefly mentions the various error handling considerations when using *Janus Web Server*.

6.4 HEADJS and ONLOAD parameters on SCREEN rules

HEADJS and ONLOAD can now be specified on JANUS WEB SCREEN rules. HEADJS indicates a URL that will be pointed to by a `<script>` tag inside the `<head>` tag in Janus Web Legacy Support screens. This allows embedding of JavaScript outside the HTML body which can be useful on occasion. It is often especially useful to have one or more JavaScript methods inside the header that are invoked when the page body is loaded as indicated the the onload attribute in the `<body>` tag and so is most useful in conjunction with the ONLOAD JANUS WEB SCREEN rule parameter.

ONLOAD indicates JavaScript statements to be placed inside the onload attribute in a <body> tag. This allows JavaScript to be invoked before the page is actually rendered but after it's been fully received from the network. The JavaScript statements indicated in the ONLOAD parameter are placed after any Janus Web Legacy support onload JavaScript statements (currently only used for cursor positioning) so that Janus Web Legacy support actions (such as cursor position) can be overridden by locally specified actions.

HEADJS and ONLOAD were actually implemented as ZAPs in *Sirius Mods* version 6.2 as ZAP6288 and in *Sirius Mods* version 6.3 as ZAP6135.

6.5 SCREENURL/NOSCREENURL parameters

SCREENURL and NOSCREENURL can now be specified on the JANUS DEFINE command for web ports and on JANUS WEB SCREEN rules. SCREENURL indicates that *Janus Web Server* should generate a new URL for Janus Web Legacy Support screens. Prior to *Sirius Mods* 6.3 a special legacy URL would be generated if NOSCREENREDIR was in effect for a request. As of *Sirius Mods* 6.3 NOSCREENREDIR does not generate a legacy URL unless SCREENURL is specified and, in fact, SCREENURL could be specified with SCREEREDIR.

6.6 \$WEB_NOCACHE function

Sets various response header parameters so that the browser will display a new version of the page, ignoring any cached version.

6.7 \$WEB_SAVE_xxx and \$WEB_REST_xxx enhancements

\$WEB_SAVE_RECSET, \$WEB_SAVE_LIST, \$WEB_REST_RECSET and \$WEB_REST_LIST have been enhanced in many ways. First they have all had an options argument added to them that consist of blank delimited options.

For \$WEB_SAVE_RECSET and \$WEB_SAVE_LIST the options are

- CANCEL** Cancel the request if the maximum possible (SRSMAX) saved record sets and \$lists are in-use.
- ERROR** Return an error code of 5 if the maximum possible (SRSMAX) saved record sets and \$lists are in-use.

- STEAL** Steals the oldest saved record set or \$list if the maximum possible (SRSMAX) saved record sets and \$lists are in-use.
- UCANCEL** Cancel the request if the maximum allowed (SRSMAXU) saved record sets and \$lists are in-use by the requesting userid.
- ERROR** Return an error code of 6 if the maximum allowed (SRSMAXU) saved record sets and \$lists are in-use by the requesting userid.
- STEAL** Steals the oldest saved record set or \$list if the maximum allowed (SRSMAXU) saved record sets and \$lists are in-use by the requesting userid.
- MOVE** Does a *move* style save, that is the pointers to the source record set or \$list are simply moved to the saved version so that the source record set or \$list looks empty after the save. This is more efficient than a COPY style save but does produce a potentially problematic side-effect.
- COPY** Does a *copy* style save, that is the contents of the source record set or \$list are copied to the saved version so that the source record set or \$list is not modified after the save. This is less efficient than a MOVE style save but does avoid a potentially problematic side-effect.

The default behavior of \$WEB_SAVE_RECSET and \$WEB_SAVE_LIST is as if STEAL, USTEAL and MOVE were specified. This is completely backward compatible with versions of the *Sirius Mods* before 6.3.

For \$WEB_REST_RECSET and \$WEB_REST_LIST the options are

- MOVE** Does a *move* style restore, that is the pointers to the source record set or \$list are simply moved to the restored version so that the source saved record set or \$list is no longer accessible after the call. This is more efficient than a COPY style save but does produce a potentially problematic side-effect.
- COPY** Does a *copy* style save, that is the contents of the source record set or \$list are copied to the restored version so that the source saved record set or \$list is still accessible after the call. This is less efficient than a MOVE style save but does avoid a potentially problematic side-effect.

The default behavior of \$WEB_REST_RECSET and \$WEB_REST_LIST is as if MOVE were specified. This is completely backward compatible with versions of the *Sirius Mods* before 6.3.

The default behavior of all of these functions can, however, be changed on a thread basis by setting the SRSPARM parameter (which can be reset with \$RESETN). SRSPARM is a bitmask user parameter where the bits mean:

- X'01'** Make default for \$WEB_SAVE_RECSET and \$WEB_SAVE_LIST "ERROR"

- X'02'** Make default for \$WEB_SAVE_RECSET and \$WEB_SAVE_LIST "CANCEL"
- X'04'** Make default for \$WEB_SAVE_RECSET and \$WEB_SAVE_LIST "UERROR"
- X'08'** Make default for \$WEB_SAVE_RECSET and \$WEB_SAVE_LIST "UCANCEL"
- X'10'** Make default for \$WEB_SAVE_RECSET and \$WEB_SAVE_LIST "COPY"
- X'20'** CANCEL request on a COPY style \$WEB_SAVE_RECSET for an exclusively locked record set. If this bit is not set, this error would simply be reflected with an error code of 7. This operation is not allowed because there cannot be two threads that have the same records locked in exclusive mode which is what would be the case after a COPY type save of an exclusively locked record set.
- X'40'** Make default for \$WEB_REST_RECSET and \$WEB_REST_LIST "COPY"
- X'80'** CANCEL request on a COPY style \$WEB_REST_RECSET for an exclusively locked record set. If this bit is not set, this error would simply be reflected with an error code of 7. This operation is not allowed because there cannot be two threads that have the same records locked in exclusive mode which is what would be the case after a COPY type restore of an exclusively locked record set.

The default value for SRSPARM is 0 which preserves the current behavior of these functions.

In addition to the new parameters and \$function options there are a host of new stats

- SRSDEFT** Default timeout value for saved record sets and \$lists. Same as the value of the eponymous system parameter.
- SRSMAX** Maximum number of saved record sets and/or \$lists in the system. Same as the value of the eponymous system parameter.
- SRSMAXT** Maximum timeout value allowed for a saved record set or \$list. Same as the value of the eponymous system parameter.
- SRSMAXU** Maximum number of saved record sets and/or \$lists per userid. Same as the value of the eponymous system parameter.
- SRSNCUR** Current number of saved record set or \$list slots used out of the SRSMAX slots available.
- SRSNEXP** Number of saved record sets or \$lists that were expired, that is deleted because they had not been referenced within their timeout time (rate or total).

SRSNFUL	Number of saved record sets or \$lists that could not be saved because there were already SRSMAX non-expired saved record sets or \$lists saved in the system and CANCEL or ERROR was in effect for \$WEB_SAVE_RECSET or \$WEB_SAVE_LIST calls (rate or total).
SRSNHGH	High water mark of saved record set or \$list slots used out of th SRSMAX slots available.
SRSNRST	Number of record sets or \$lists restored via \$WEB_REST_RECSET or \$WEB_REST_LIST calls (rate or total).
SRSNSAV	Number of record sets or \$lists saved via \$WEB_SAVE_RECSET or \$WEB_SAVE_LIST calls (rate or total).
SRSNSTL	Number of saved record sets or \$lists that were stolen, that is deleted before their timeout time, because another user did a \$WEB_SAVE_RECSET or \$WEB_SAVE_LIST with STEAL in effect and the maximum saved record sets and \$lists (SRSMAX) were already saved.
SRSNSUS	Number of saved record sets or \$lists that were stolen, that is deleted before their timeout time, because a userid did a \$WEB_SAVE_RECSET or \$WEB_SAVE_LIST with USTEAL in effect and the maximum saved record sets and \$lists (SRSMAXUS) were already saved by the userid (rate or total).
SRSNULM	Number of saved record sets or \$lists that could not be saved because there were already SRSMAXUS non-expired saved record sets or \$lists saved by the userid and UCANCEL or UERROR was in effect for \$WEB_SAVE_RECSET or \$WEB_SAVE_LIST calls (rate or total).
SRSSAVG	Average age in seconds of saved record sets or \$lists stolen because a user did a \$WEB_SAVE_RECSET or \$WEB_SAVE_LIST and the system limit of saved record sets and \$lists (SRSMAX) had been hit and STEAL was in effect for the indicated \$function calls.
SRSSMIN	Minimum age in seconds of saved record sets or \$lists stolen because a user did a \$WEB_SAVE_RECSET or \$WEB_SAVE_LIST and the system limit of saved record sets and \$lists (SRSMAX) had been hit and STEAL was in effect for the indicated \$function calls.
SRSUAVG	Average age in seconds of saved record sets or \$lists stolen because a user did a \$WEB_SAVE_RECSET or \$WEB_SAVE_LIST and the user limit of saved record sets and \$lists (SRSMAXUS) had been hit and USTEAL was in effect for the indicated \$function calls.
SRSUMIN	Minimum age in seconds of saved record sets or \$lists stolen because a user did a \$WEB_SAVE_RECSET or \$WEB_SAVE_LIST and the user

limit of saved record sets and \$lists (SRSMAXUS) had been hit and USTEAL was in effect for the indicated \$function calls.

 CHAPTER 7 *Sirius Functions*

The \$functions and facilities presented in this chapter are new or have changed in version 6.3.

7.1 \$FIELD_LISTI and \$FIELD_IMAGE enhancements

The first argument to \$FIELD_IMAGE and second argument to \$FIELD_LISTI can now have a blank after the image name (and optional occurrence item name) followed by a prefix to be prepended to each image item name in generating field names. For example, before *Sirius Mods* 6.3 if a file had fields in a repeating group call ORDERDATA.PRODID, ORDERDATA.QUANTITY and ORDERDATA.PRICE an image definition used with \$FIELD_LISTI would need to look something like

```
IMAGE ORDERDATA
  ORDERDATA.PRODID   IS STRING LEN 8
  ORDERDATA.QUANTITY IS BINARY LEN 4
  ORDERDATA.PRICE    IS BINARY LEN 4
END IMAGE
```

so you could do

```
%RC = $FIELD_LISTI(%OLIST, 'ORDERDATA')
```

which means you'd have “ugly” looking image item names like %ORDERDATA:ORDERDATA.PRODID. By specifying a prefix in the \$FIELD_LISTI or \$FIELD_IMAGE call

```
%RC = $FIELD_LISTI(%OLIST, 'ORDERDATA ORDERDATA.')
```

the image definition could be simplified to

```
IMAGE ORDERDATA
  PRODID   IS STRING LEN 8
  QUANTITY IS BINARY LEN 4
  PRICE    IS BINARY LEN 4
END IMAGE
```

so that the image item references like %ORDERDATA:PRODID are much “nicer”.

\$FIELD_LISTI and \$FIELD_IMAGE also have a new extra argument (argument 4 for \$FIELD_IMAGE and argument 6 for \$FIELD_LISTI) which indicates a special value to

be treated as a null when populating the target \$list or image. This is useful because storing nulls in fields is problematic on many fronts in *Model 204* so most sites have a special value that acts as a placeholder for a null. Without the new argument to \$FIELD_LISTI and \$FIELD_IMAGE, an application would have to go through the \$list items or image item to find these placeholder values and convert them to real nulls. Obviously, this is tedious, error-prone and can be CPU intensive. By specifying the new *null-value* argument to \$FIELD_LISTI and \$FIELD_IMAGE, these functions will automatically convert the null placeholder to a real null. For example, if the string “_NULL_” is used to indicate a null value in a file, the following will convert all values of “_NULL_” to a null before populating the target \$list.

```
%RC = $FIELD_LISTI(%OLIST, 'ORDERDATA', , , , -  
                '_NULL_')
```

7.2 \$lists as parameters to ECF programs

A \$list identifier can now be passed to an ECF program as in

```
EXTERNAL CALL COBPROG WITH $LIST inlist outlist
```

where

inlist is the \$list identifier to be passed to the ECF program as input.

outlist is the \$list identifier to receive the data as modified by the ECF program. The contents of this \$list are replaced with the ECF program output. The length of the \$list items are made to match that of the input \$list and if the output \$list is the same identifier as the input \$list, the output data simply replaces the original input data. For PARMTYPE=INPUT ECF programs the output \$list is ignored.

The input \$list is copied into the ECF parameter area before the ECF program is called. The \$list items are simply concatenated to each other and there is no way to determine where one ends and the next begins so presumably the \$list items would all have the same length or some \$list items have the length or number of occurrences of a following set of \$list items which can be used in the ECF program to determine the structure of the data. However, the most common structure for passing data from a \$list to an ECF program is likely to be a \$list where all \$list items have the same length. To facilitate use of such a structure, the \$list ECF interface automatically places a one word (four byte) binary number of occurrences before the copied \$list items in the ECF parameter area. This makes it possible, if the ECF program is written in COBOL, to refer to the

\$list as a *DEPENDING* array where the number of occurrences can vary at run-time. To refer to the \$list, the COBOL program would look something like

```
LINKAGE SECTION.  
  01  STUBB.  
  03  NUM-HARPOONS      PIC 9(9) BINARY  
  03  HARPOON-INFO     OCCURS 0 TO 25000 TIMES  
                               DEPENDING ON NUM-HARPOONS.  
      ...  
PROCEDURE DIVISION USING STUBB.
```

If for some reason, this count is deemed undesirable in the linkage structure it can be suppressed via the NOCOUNT keyword in the EXTERNAL CALL statement as in

```
EXTERNAL CALL COBPROG WITH $LIST %LIST NOCOUNT
```

Multiple \$lists and images can be passed in a single ECF call as in

```
EXTERNAL CALL COBPROG WITH $LIST %LIST, -  
                          INIMAGE, -  
                          $LIST %LIST2
```

7.3 \$LIST_ADD_ORDERED, \$LIST_ADD_UNIQUE and \$LIST_ADD_UNIQUE_ORDERED

These functions all have the format

```
%RC = $LIST_ADD_ORDERED(list, string)  
%RC = $LIST_ADD_UNIQUE(list, string)  
%RC = $LIST_ADD_UNIQUE_ORDERED(list, string)
```

where

list is a \$list identifier to which the string might be added.

string is the string to add to the \$list.

\$LIST_ADD_ORDERED always adds the indicated string even if that string already exists on the \$list but the new item is inserted so that the \$list is in EBCDIC order. \$LIST_ADD_ORDERED assumes that the \$list is in EBCDIC order so it does a pseudo binary search to locate the correct insertion point.

\$LIST_ADD_UNIQUE always adds the indicated string to the end of the \$list but does not add it if there's already an identical \$list item on the \$list. \$LIST_ADD_UNIQUE does not assume any order for the \$list so sequentially scans the entire \$list for matches to the string being added.

`$LIST_ADD_UNIQUE_ORDERED` only adds the indicated string to the \$list if there isn't already an identical \$list item on the \$list. If there are no matching items, the new item is inserted so that the \$list is in EBCDIC order. `$LIST_ADD_UNIQUE_ORDERED` assumes that the \$list is in EBCDIC order so it does a pseudo binary search to locate a match or the correct insertion point.

These functions all return either the item number added or inserted if no match was found or the negative item number of the matching item if one was found. Obviously, `$LIST_ADD_ORDERED` always returns a positive item number since it does not look for matches to ensure uniqueness. The return code makes it easy to maintain a parallel \$list that contains say a count of the number of times a given value occurred, that is was passed as a string to `$LIST_ADD_UNIQUE` or `$LIST_ADD_UNIQUE_ORDERED`. The following illustrates such an approach:

```
%IN = $LIST_ADD_UNIQUE_ORDERED(%OLIST, %DATA)
IF %IN GT 0 THEN
  %RC = $LISTINS(%CLIST, %IN, 1)
ELSE
  %IN = -%IN
  %RC = $LISTREP(%CLIST, %IN, -
                $LISTINF(%CLIST, %IN) +1 )
END IF
```

7.4 `$LSTR_GET_IMAGE` and `$LSTR_SET_IMAGE`

`$LSTR_GET_IMAGE` returns the contents of an image as a LONGSTRING as in

```
%LSTR = $LSTR_GET_IMAGE(image)
```

where

image is the name of the image to be retrieved.

`$LSTR_SET_IMAGE` sets the contents of an image from a LONGSTRING as in

```
%RC = $LSTR_SET_IMAGE(image, value)
```

where

image is the name of the image to be set.

value is a longstring value to which to set the image.

`$LSTR_SET_IMAGE` returns the number of bytes set in the image.

`$LSTR_GET_IMAGE` and `$LSTR_SET_IMAGE` can be useful for maintaining multiple copies of or versions of an image in a single request, for maintaining one or more global

copies of an image (using global LONGSTRINGs) without using the GTBL space required by the standard GLOBAL IMAGE feature and for maintaining copies of an image associated with a session that survives a logout (using session LONGSTRINGs).

7.5 Session \$functions

Several new \$functions are available to support sessions. A *session* is a workstream that might extend beyond a *Model 204* login session and might even be transferred among multiple *Model 204* threads or users. Sessions are especially useful for web applications that require some context to be maintained over multiple web requests, each of which requires a login and logout. Sessions are either owned by a specific user or are public in which case they are considered to be owned by “*”, that is all users. Non-privileged users can only create, access and delete public sessions and those owned by themselves, that is by their own userids. Privileged, that is system manager or system administrator users can create, access and delete public sessions and those owned by any user.

There are several system parameters that control the availability and behavior of the session feature:

- SESNPRV** The maximum number of private sessions that can be active. Setting this value allocates about 160 bytes of virtual storage for each possible session. The default value for SESNPRV is 0 which means that no private sessions can be created.
- SESNPUB** The maximum number of public sessions that can be active. Setting this value allocates about 160 bytes of virtual storage for each possible session. The default value for SESNPUB is 0 which means that no public sessions can be created.
- SESUMAX** The maximum number of private sessions that can be active for a single owner (userid). The default value for SESUMAX is 0 which means that a given userid can only have one session active at a time.
- SESOPT** A bitmask parameter that controls the behavior of the session feature. The meaning of the bits are:
- X'01'** Clean up timed out sessions. If this bit is on and the X'02' bit is off, every \$SESSION_CREATE, \$SESSION_OPEN, \$SESSION_CLOSE and \$SESSION_DELETE call cleans up timed out sessions. This cleanup could occur well after the timeout occurred if \$SESSIONxxx calls are relatively infrequent. Use this option if timed-out session cleanup is desired but is not worth incurring the overhead of having a PST to do the cleanups.

X'02' Create a PST to clean up timed out sessions. If this bit is on a PST is attached in the Online whenever there is an active but not open session. If a session times out, the PST immediately deletes the session, freeing up the CCATEMP pages used by the session. Use this session if cleaning up sessions very quickly is worth the (probably slight) cost of having a PST around to do the cleanup.

The default value for SESOPT is 0 which means that timed out sessions are not cleaned up and, in fact, can still be opened after timeout if they haven't been deleted yet. Under the default settings, sessions are only deleted when they are timed out, a new private or public sessions is being created and the maximum like-typed (private or public) sessions are active. In this case the oldest like-typed session is deleted.

SESDEFTO Sets the default timeout value for a session if one is not specified in the \$SESSION_CREATE call. The default value for SESDEFTO is 900 which indicates a default timeout of 900 seconds or 15 minutes. If a session is not re-opened within the timeout value of the \$SESSION_CLOSE, it is considered *timed-out* and liable to deletion under control of the SESOPT flags. The SESDEFTO setting can be overridden in many of the \$SESSIONxxx functions.

SESDEFOW Sets the default open wait time value for a session if one is not specified in the \$SESSION_OPEN call. The default value for SESDEFOW is 0 which means a \$SESSION_OPEN call will not wait if the request session is in-use, that is open by another thread. If a session is still in-use after the open wait time, a return code of 2 is set by \$SESSION_OPEN to indicate the problem. The SESDEFOW setting can be overridden in the \$SESSION_OPEN call.

7.5.1 \$SESSION_CREATE

Creates a new session and has the format

```
%RC = $SESSION_CREATE(sesid, owner, timeout, opts)
```

where

sesid is the id by which the session is to be referred. This is a required argument.

owner is the userid that will own this session. An *owner* of "*" means that the session is public, that is available to all users. This argument defaults to the creating user's userid.

timeout sets the time after the \$SESSION_CLOSE after which the session is considered *timed-out*, that is eligible for deletion. A value of -1 means use

the SRSDEFTO parameter value. This argument defaults to -1, that is the SRSDEFTO parameter value.

opts a blank delimited set of options to control the create process. The options are:

OPEN Automatically open the session upon creating it. This is the default behavior.

NOOPEN Don't automatically open the session upon creating it. This is not the default behavior but might be useful if creating a session for another user or creating many sessions at once.

\$SESSION_CREATE returns a 0 if successful and a non-zero value otherwise.

7.5.2 **\$SESSION_DELETE**

Deletes a session and has the format

```
%RC = $SESSION_DELETE(sesid, owner, timeout)
```

where

sesid is the id of the session to be deleted. This is a required argument.

owner is the userid that owns the session to be deleted. An *owner* of "*" means that the session is public, that is available to all users. This argument defaults to the creating user's userid.

timeout sets the time to wait for an in-use session to be closed to perform a synchronous delete. If timeout is 0 or the session being deleted is the session currently opened by the invoking user or the session is not closed within the indicated timeout time, the session is deleted asynchronously when the session is closed. This argument defaults to 0 which means that if the session is in-use it is deleted asynchronously.

\$SESSION_DELETE returns a 0 if the session has been deleted, a 1 if the session did not exist and a 2 if it was in-use and marked for asynchronous deletion.

7.5.3 **\$SESSION_OPEN**

Opens a previously created session and has the format

```
%RC = $SESSION_OPEN(sesid, owner, timeout)
```

where

- sesid** is the id of the session to be opened. This is a required argument.
- owner** is the userid that owns the session to be opened. An *owner* of “*” means that the session is public, that is available to all users. This argument defaults to the creating user's userid.
- timeout** sets the time to wait for an in-use session to be closed before giving up. A minus -1 indicates to use the value of the SRSDEFOW parameter. This argument defaults to -1, that is the value of the SRSDEFOW parameter.

\$SESSION_OPEN returns a 0 if the session has been opened, a 1 if the session did not exist and a 2 if it was in-use.

7.5.4 **\$SESSION_CLOSE**

Closes the current session and has the format

```
%RC = $SESSION_CLOSE(sesid, owner, timeout)
```

where

- sesid** sets a new id for the session being closed. This is an optional argument and if not specified leaves the session's id unchanged.
- owner** is the userid that is to be set as the new owner of the session being closed. An *owner* of “*” means that the session is public, that is available to all users. This is an optional argument and if not specified leaves the session's owner unchanged.
- timeout** is the new value to be used as the session timeout value. A minus -1 indicates to use the value of the SRSDEFTO parameter. This is an optional argument and if not specified leaves the session's timeout value unchanged.

\$SESSION_CLOSE returns a 0 if the session is closed, a -1 if no session is open and a number greater than 0 for other errors.

7.5.5 **\$SESSION**

Has no parameters and returns a 1 if the user has a session currently open and a 0 otherwise.

7.5.6 **\$SESSION_ID**

Has no parameters and returns the id of the currently open session if there is one and a null otherwise.

7.5.7 \$SESSION_OWNER

Has no parameters and returns the owner of the currently open session if there is one and a null otherwise. A value of "*" indicates that the currently open session is a public session.

7.5.8 \$SESSION_TIMEOUT

Has no parameters and returns the timeout value of the currently open session if there is one and a -1 otherwise.

7.5.9 \$SESSION_LIST

Returns a list of active sessions to a \$list and has the format

```
%RC = $SESSION_LIST(listid, sesid, owner)
```

where

listid Is the \$list identifier to receive the output from \$SESSION_LIST. This is a required argument.

sesid Is the session id for which information is to be returned. This argument can contain wildcards — for example "PEQ*" indicates that information is to be returned for all sessions beginning with the letters "PEQ". This is an optional argument and defaults to "*" meaning all session ids are to be listed.

owner Is the session owner for which information is to be returned. This argument can contain wildcards though for a non-system manager user only sessions owned by the requesting user and public sessions will be listed. Note that a "*" will list all private and public sessions but a "*" will list public sessions only. This is an optional argument and defaults to the userid of the invoking user.

The format of the data in the output \$list is

Col 1-10 Owner of the session; * for public sessions.

Col 11-16 Session timeout value.

Col 17-30 Session creation time in YYYYMMDDHHMISS format.

Col 31-44 Last session access time in YYYYMMDDHHMISS format. If the session is currently open this value is the time of the \$SESSION_LIST invocation.

Col 45-50 User number with session open. If the session is not currently open by any users these columns contain all blanks.

Col 51- The session id.

\$SESSION_LIST returns the number of sessions for which information was added to the output \$list.

7.6 Session \$lists

Several new \$functions are available to support session \$lists. Session \$lists are just like global \$lists except they survive a logoff and can be reestablished with the \$SESSION_OPEN function. Any attempt to reference a session \$list results in request cancellation if there is no current session established via a \$SESSION_CREATE or \$SESSION_OPEN call.

The new functions available to support session \$lists are \$LIST_SESSION, \$LIST_SESSION_DEL and \$LIST_SESSION_LIST which work exactly like \$LIST_GLOBAL, \$LIST_GLOBAL_DEL and \$LIST_GLOBAL_LIST respectively except that they refer to \$lists for the current session rather than those global to the current login.

7.7 Global and session LONGSTRINGs

Several new \$functions are available to support global and session LONGSTRINGs. Global LONGSTRINGs are LONGSTRINGs whose values survive end of request. Session LONGSTRINGs not only survive end of request but survive a logoff and can be reestablished with the \$SESSION_OPEN function. Any attempt to reference a session LONGSTRING results in request cancellation if there is no current session established via a \$SESSION_CREATE or \$SESSION_OPEN call.

A LONGSTRING is bound to a session or global LONGSTRING with \$LSTR_SESSION or \$LSTR_GLOBAL respectively as in

```
%RC = $LSTR_SESSION(gname, lstr, options)
%RC = $LSTR_GLOBAL(gname, lstr, options)
```

where

gname is the global or session name to which the LONGSTRING is to be bound.

lstr is the LONGSTRING %variable. Subroutine parameters are not allowed.

options indicates whether the bound global should already exist and whether the global should be cleared by the function. Valid values for *options* are OLD, NEW, ANY, PREP, PREPOLD, PREPNEW, PREPANY, OLDPREP, NEWPREP and ANYPREP with a default of ANY.

Unlike global or session \$lists, a global or session LONGSTRING can only be bound to a single LONGSTRING in a request. If there is a need to refer to the same session or global LONGSTRING in multiple subroutine contexts within a request, that session or global LONGSTRING should be bound to a COMMON LONGSTRING that can be referenced in all the appropriate subroutine contexts.

A session or global LONGSTRING can be set from another LONGSTRING by \$LSTR_SESSION_SET or \$LSTR_GLOBAL_SET respectively as in

```
%RC = $LSTR_SESSION_SET(gname, value)
%RC = $LSTR_GLOBAL_SET(gname, value)
```

where

gname is the global or session name whose value is to be set.

value is the value to which the global or session LONGSTRING is to be set.

The session or global LONGSTRING being set need not have been set as such before these function invocations and could be bound to a local LONGSTRING.

The value of a session or global LONGSTRING can be retrieved with or \$LSTR_SESSION_GET and \$LSTR_GLOBAL_GET respectively as in

```
%VALUE = $LSTR_SESSION_GET(gname)
%VALUE = $LSTR_GLOBAL_GET(gname)
```

where

gname is the global or session name whose value is to be retrieved.

If the indicated session or global name has never been set or bound a null is returned.

Session or global LONGSTRINGS can be deleted with \$LSTR_SESSION_DEL and \$LSTR_GLOBAL_DEL respectively as in

```
%RC = $LSTR_SESSION_DEL(gname)
%RC = $LSTR_GLOBAL_DEL(gname)
```

where

gname is the global or session name to be deleted. *gname* can contain wildcard characters. For example, a *gname* of "*" would delete all global or session LONGSTRINGS.

These functions return the number of deleted session or global LONGSTRINGS. Any deleted global or session LONGSTRING is unbound from a local LONGSTRING to which it was bound via \$LSTR_SESSION or \$LSTR_GLOBAL.

7.8 **\$SIR_WILD**

Indicates whether the one string matches a Sirius-style pattern where a “*” matches any set of characters, a “?” matches any single character and a “” indicates that the following character is to be treated as a literal even if it is one of the three special wildcard characters, that is “*”, “?” or “”. The format of the \$SIR_WILD call is

```
%RC = $SIR_WILD(string, wildcard)
```

where

string Is the string to be tested for a match.

wildcard Is the string possibly containing wildcards against which it is to be tested.

%RC returns either a 1 indicating that the first string matches the second or 0 otherwise. For example

```
$SIR_WILD('Ahab', 'A*')
```

returns 1

```
$SIR_WILD('Starbuck', 'A*')
```

returns 0 and

```
$SIR_WILD('*LOOK', '*LOO?')
```

returns 1.

8.1 Shared DASD support for Sysplex and LPARs

SirSafe simplifies the operation of *Model 204* in Sysplex and multiple LPAR environments. The *SIRENQ* stand-alone utility allows members of a Sysplex or GRS ring to "see" each other, providing automatic recovery from a variety of shared DASD enqueueing conflicts that arise when *Model 204* jobs are scheduled across multiple systems. Extended monitoring and administration facilities within the *Model 204* environment simplify administration of Sysplex environments.

8.1.1 Shared DASD Background and Overview

A shared DASD configuration allows a particular DASD device to be accessed by more than one computer system or operating system instance. Shared DASD has become far more common with the advent of Logical PARTition (LPAR) support, fiber-optic (ESCON) channels with EMIF sharing between LPARs, and Sysplex clusters.

Model 204 utilizes a variety of mechanisms to serialize the access and update of database files among the various users in an online instance and between various instances of *Model 204*. The primary mechanism serializing database file accesses between various instances of *Model 204* is the operating system ENQ and DEQ facility. When a database file is opened, *Model 204* enqueues upon a resource name comprising the database file name, the volume serial for the first (or only) dataset of the file and the dataset name for the first dataset:

```
filename.volser.fully_qualified_dataset_name
```

The strength of enqueue (share or exclusive) depends upon the access intent for the file. In order to support deadlock-free reduction in access strength, *Model 204* uses three different "queue names": IFAMQA, IFAMQB and IFAMQC.

This mechanism works well for cataloged and uncataloged datasets referenced only from one operating system instance. However, it breaks down when two instances of *Model 204* running on separate systems access the same file via shared DASD. This is because by default the two operating system instances will not share information about the enqueues used by *Model 204*. Thus, two separate instances of *Model 204*, each running on different systems can simultaneously hold exclusive enqueues for the same resource.

The shared DASD support in *Model 204* is based upon the concept of a "Shared DASD Enqueue List". This enqueue list is a structure contained in the first page of the first (or only) dataset of each database file, the *File Parameter List* page. An entry is placed in a

file's enqueue list for every *Model 204* instance that opens the file. The entry is deleted when the file is physically closed by the instance and the entry is updated if the instance changes its access intent for the file.

The shared DASD enqueue list is used during the *Model 204* database file open process to identify conflicts with *Model 204* instances running under other operating system instances. Each entry in the enqueue list contains the following information:

- **System name**
A four-character name that identifies the system that is hosting the *Model 204* instance, identified as the first field in message M204.0061. For instances running under MVS or OS/390, this ID will be the SMF system ID, set by the parameter SID in a SMFPRMxx parmlib member. For instances running under CMS, this ID will be "CMS".
- **Access intent**
This is either EXCL when a file is opened for update, else SHR.
- **Jobname**
For most operating systems this is the job name of the *Model 204* instance. For instances of *Model 204* running under CMS, it is the ID of the *Model 204* virtual machine.
- **Stepname**
For most operating systems this is the step name of the *Model 204* instance. For instances of *Model 204* running under CMS, it is the six-character CPU serial number padded on the right with two blanks.
- **Date/Time**
The date and time when the entry was added or last updated.

The open logic also removes obsolete enqueue entries that were established, but no longer needed by *Model 204* instances executing under the current operating system instance. Obsolete entries can result from system crashes, x22 ABENDs and other infrequent (but common) events. The following logic is used:

1. Standard operating system ENQ/DEQ logic is used to enqueue upon the specific database file. If the enqueue fails, the open is rejected with *file is in use*.
2. A hardware **RESERVE** is placed upon the volume containing the FPL page. This prevents other systems from accessing the device and potentially changing the enqueue list.
3. The FPL is read and the shared DASD enqueue list is scanned for a conflict with the access being requested by the current open. If no conflict is found, the list is updated and the FPL is written back out to disk.
 - a. If an entry is found that conflicts with our access, but the entry was established by a *Model 204* instance under our system, delete the entry because it is obsolete. We know the entry is obsolete because the standard ENQ/DEQ mechanism under our operating system gave us the requested access.

- b. If a conflicting entry was established under a **different** operating system instance, reject the open with M204.0582 *access prevented by* and M204.0584 *file is in use* messages.
 - c. Otherwise, add or modify the appropriate entry for our open and write out the FPL page with an updated enqueue list.
4. Release the hardware **RESERVE** on the volume containing the FPL page.

If an OPEN request is prevented by an obsolete entry from a different operating system instance, the *Model 204* **ENQCTL** command can be used to clear out the offending entry. If **ENQCTL** is used to delete a non-obsolete entry, then an M204.0585 *list overlaid* message will result when the file is closed on the system whose entry was deleted.

8.1.2 Global Resource Serialization (GRS)

Global Resource Serialization (GRS) allows two or more operating system instances to share information about serialized resources. GRS can be used to support ENQ/DEQ serialization between address spaces executing under different instances of an OS/390 operating system. GRS is mandatory between members of a Sysplex and optional between systems that are not members of the same Sysplex.

Once a GRS ring is established, ENQ/DEQ requests between all members of the ring can be serialized as if the entire ring was a single operating system image. The ENQ/DEQ services accept an argument that specifies the scope of the resource being enqueued:

SYSTEM The current ENQ/DEQ applies just to the current operating system instance. ENQ/DEQ requests with a scope of SYSTEM are not visible to other members of a GRS ring.

SYSTEMS The current ENQ/DEQ applies to the GRS ring of the current system. ENQ/DEQ requests with a scope of SYSTEMS are not considered to be the same as requests with a scope of SYSTEM.

This last problem means that all of the *Model 204* instances executing on an operating instance must use the same specification for scope. In practice this can be very hard to ensure, and errors could compromise file integrity.

Fortunately, a GRSRNLxx parmlib member can be used to override the scope specification for ENQ/DEQ requests. This makes it possible for all of the *Model 204* file control ENQ/DEQ requests on a system to be promoted from a scope of SYSTEM to a scope of SYSTEMS. This is accomplished by the following entries in a GRSRNLxx member:

```
RNLDEF RNL(INCL) TYPE(GENERIC) QNAME(IFAMQA)
RNLDEF RNL(INCL) TYPE(GENERIC) QNAME(IFAMQB)
RNLDEF RNL(INCL) TYPE(GENERIC) QNAME(IFAMQC)
```

If the entries above are active for a particular instance of OS/390, then all of the ENQ/DEQ requests used by *Model 204* to serialize database file access will be promoted to the SYSTEMS scope. This guarantees that the current *Model 204* logic for shared DASD will continue to work, but in and of itself does not extend the functionality.

8.1.3 SIRENQ

The *SIRENQ* utility is designed to be run as a started task or batch job under all of the OS/390 instances that can host *Model 204* jobs. The purpose of *SIRENQ* is to let each *Model 204* instance determine if it is a member of a Sysplex or GRS ring and if so to determine the other members of the ring that are promoting IFAMQx enqueues. *Model 204* can then treat all systems so identified as a single system image, performing automatic cleanup of obsolete shared DASD enqueue list entries. This approach provides significant benefit to Sysplex and GRS ring members, while still maintaining the usefulness of the current shared DASD approach for other systems, e.g. a monoplex used for testing.

SIRENQ establishes three special enqueues, each with a scope of SYSTEM:

```
IFAMQA  MODEL204.SIRENQ.smfid
IFAMQB  MODEL204.SIRENQ.smfid
IFAMQC  MODEL204.SIRENQ.smfid
```

Where *smfid* is the SMF system ID for the current system. *SIRENQ* then uses the GQSCAN system service to build a list of all OS/390 instances that have enqueues on these resources with a scope of SYSTEMS. By examining the last portion of the resource name, *SIRENQ* can determine if the IFAMQx enqueues from other systems will be “visible”. *SIRENQ* produces informational messages to the operator's console when other OS/390 instances either become visible or cease to be visible. Then *SIRENQ* sleeps for an interval (default 5 minutes) and then rechecks its view of the GRS environment and updates its list of other OS/390 instances running *SIRENQ*.

SIRENQ accepts the following parameters from the PARM field of its EXEC statement:

INTERVAL=nnn

Number of minutes to wait before reissuing GQSCAN. Useful for avoiding 522 ABENDs. The default is 5 minutes, while a value of 0 eliminates the timer and *SirENQ* just waits for a MODIFY or STOP command from the operator.

SMFID=cccc

If specified this parameter provides the SMF system ID to be used by this copy of *SIRENQ*. If not specified the SMF system ID of the current system is used. This is useful in certain hot recovery environments to avoid the need for ENQCTL commands to clear obsolete shared DASD enqueues.

SIRENQ listens for operating system **STOP** or **MODIFY** commands. If a STOP command is received, *SIRENQ* drops its enqueues and exits, causing the current

OS/390 instance to become invisible. *SIRENQ* processes the following MODIFY commands:

REFRESH

Immediately rebuild the list of *SIRENQ* copies seen on other systems, without waiting for the completion of the current time interval, if any.

EOD or EXIT

Processed as a STOP command - immediately drop the enqueues and exit.

8.1.4 Modified Shared DASD List Maintenance

All systems that are “*SirENQ visible*” to the current system are considered to be the same system for purposes of deleting obsolete shared DASD enqueue list entries. For example, if a *Model 204* instance is running on system “V210” with a copy of *SIRENQ* and *SIRENQ* is also running on another GRS ring member with an SMF system ID of “Z140”, then the first instance will automatically detect and delete any obsolete enqueue entries for Z140.

This is possible because the *Model 204* instances on V210 know that their ENQ/DEQ calls have been promoted to the SYSTEMS scope and they also know that the ENQ/DEQ calls on Z140 have been promoted. Further, *Model 204* instances on the two systems know that the GRS ring between the two instances is functional.

Note that if a third system is either not in the GRS ring or does not have a copy of *SIRENQ* active, then any *Model 204* instances running on that system will not be able to detect obsolete entries from V210 or Z140 and they in turn will not be able to detect obsolete entries from the third system. However, no loss in file integrity will result from sharing files among the three systems.

8.1.5 Enhancements to the AUTHCTL Command

The AUTHCTL command has been enhanced to better support the monitoring and management of shared DASD enqueueing within *Model 204*. The AUTHCTL VIEW command now indicates whether enhanced shared DASD enqueueing is active, and if so lists the systems which are visible and those which have become invisible:

```
*** MSIR.0688: SirSafe enhanced shared DASD active on V210 (SIRIJES2)
*** MSIR.0689: SirSafe Shared DASD visible system:  V210  (SIRIJES2)
*** MSIR.0689: SirSafe Shared DASD visible system: + Z140  (SIRIUSZ0)
*** MSIR.0689: SirSafe Shared DASD visible system: - TEST  (SIRITEST)
```

The above example shows that enhanced shared DASD enqueueing is active for the current system, that the system Z140 has become visible since the last AUTHCTL VIEW or AUTHCTL REFRESH, and the the system TEST was visible but is currently not visible. Note that the eight-character “system name” is shown as well as the older four-character SMF ID for each system.

Suppose that a *Model 204* online being tested under system TEST had several files open when the testing LPAR was reset. Rather than track down all the obsolete shared DASD enqueue list entries and delete them with an ENQCTL command, a second copy of *SIRENQ* can be run on the V210 system, with a parameter of "SMFID=TEST". This will cause any *Model 204* instance on V210 to automatically clear the obsolete entries from TEST. While this second copy of *SIRENQ* is active, an AUTHCTL VIEW command would yield:

```
*** MSIR.0688: SirSafe enhanced shared DASD active on V210 (SIRIJES2)
*** MSIR.0689: SirSafe Shared DASD visible system:  Z140 (SIRIUSZ0)
*** MSIR.0689: SirSafe Shared DASD visible system:  TEST (SIRIJES2)
```

The presence of the system name makes it easy to spot duplicate copies of *SIRENQ* used to facilitate recovery. Note that if the system SIRITEST was restarted and rejoined the Sysplex or GRS ring, then when its copy of *SIRENQ* was started, the operator of SIRITEST would receive an error notification indicating that SIRIJES2 was running a spoofing copy of *SIRENQ*.

Each *Model 204* instance maintains its view of visible systems in an efficient manner. A new AUTHCTL command is available for causing *Model 204* to manually rebuild its view of visible systems:

```
AUTHCTL REFRESH [CLEAR]
```

The CLEAR option directs *Model 204* to remove from its list any systems that were once visible, but are now invisible (i.e., those entries preceded by a minus sign). If a *Model 204* instance is started without a copy of *SIRENQ* running, then an AUTHCTL REFRESH command is required to enable enhanced shared DASD enqueueing.

8.2 **`$SIR_CHECK_ACCESS` function**

The `$SIR_CHECK_ACCESS` function allows installations to code sophisticated interfaces between *Model 204* applications and a System Authorization Facility such as RACF or ACF2. `$SIR_CHECK_ACCESS` lets a system security administrator maintain controls over "dataset names" that identify intended application actions. By calling the function at strategic points a User Language program can provide security with arbitrary granularity. The format of the `$SIR_CHECK_ACCESS` call is

```
%RC = $SIR_CHECK_ACCESS( dsn, prefix, access, log)
```

where

dsn an upper case "dataset name", following the usual rules. Note: if a prefix is provided, the prefix will be concatenated to the front of dsn with a separating dot ('.') and the resulting string must be less than 44 characters.

prefix a string indicating whether the provided dataset name is to be prefixed. Valid values are:

NONE no prefix is to be provided (the default).

AUTH the same HLQ used by *SirSafe* determined by the external authorizer in use:

RACF the RACF control group name in effect for the run, default M204RACF.

ACF2 The character "R" with the ACF2 resource type appended, default R204.

JOB the current job name.

JOB.STEP the current job name and step name separated by a dot.

AUTH.JOB the SirSafe HLQ and job name separated by a dot.

CCASYS the dataset name for CCASYS

access

flag indicated desired access, either "R" for read or "W" for write. The default is R.

log flag indicating whether failed access checks should be logged, either "Y" or "N". Default is "N", which suppresses logging.

\$SIR_CHECK_ACCESS returns an integer value as follows:

- 2** no authorizer running, call ignored
- 1** access not allowed
- 0** access allowed
- 1** resulting dsname (prefix.dsname) invalid
- 2** prefix argument invalid
- 3** read/write flag invalid
- 4** log/nolog flag invalid

Compatibility/Bug fixes

This chapter lists any compatibility issues with prior versions of the *Sirius Mods* and any bugs which have been fixed in this version of the *Sirius Mods* but had not, as of the date of this release, been fixed in the immediately prior version (6.2).

The first sections (that is, all sections before “Fixes in *Sirius Mods* 6.3 but not in 6.2” on [page 40](#)) list, by product, any backwards incompatibility issues, that is, any differences in processing that result from successful execution with *Sirius Mods* version 6.2, as compared with the same inputs to *Sirius Mods* version 6.3.

9.1 Janus Sockets

9.1.1 \$SOCK_SENDx argument values longer than 255

Since the *Janus Sockets* \$functions are now LONGSTRING capable, any argument values longer than 255 bytes are passed in their entirety to the \$function. In previous versions, such values would be truncated at 255 bytes. \$SOCK_SEND and \$SOCK_SENDDLN present different behavior for this; for example, if %A and %B are STRINGS and their combined length exceeds 255

```
%RC = $SOCK_SEND(%SOK, %A WITH %B)
```

would send the first 255 bytes of the concatenated results prior to *Sirius Mods* 6.3 but as of this version it will send the entire concatenated result.

9.1.2 \$SOCK_RECV result

\$SOCK_RECV now returns the number of bytes received when the RECVLIM is hit or a FIN is received. Before *Sirius Mods* 6.3 \$SOCK_RECV used to return a 0 in such a case.

9.2 Sirius Functions

9.2.1 \$COMMAND and \$COMMNDL output width parameter

\$COMMAND and \$COMMNDL allow an output line width to be specified that is greater than 256. The maximum line width allowable in a \$COMMAND and \$COMMNDL is now the smallest LOBUFF value for any sdaemon in the Online. In addition, where these

functions would silently set the line length to 80 and 256 respectively if an invalid line width was specified, as of *Sirius Mods* 6.3 these functions now cancel the request on an invalid line width value.

9.3 Fixes in *Sirius Mods* 6.3 but not in 6.2

This section lists other fixes to functionality existing in the *Sirius Mods* version 6.2 but which, in absence of customer problems, have not, as of the date of the release, been fixed in that version.

There are no fixes in *Sirius Mods* 6.3 that are not available in previous versions.

9.4 Version co-requisites

This section lists any restrictions on usage of various products (including *Sirius Mods* itself) which will be imposed by use of version 6.3 of *Sirius Mods*.

There are no corequisites associated with *Sirius Mods* 6.3.