
Sirius Mods Release Notes Version 6.4

May, 2003



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*

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
Model 204 support	3
Documentation	3
Chapter 3: All or Multiple Products	5
SIRIUS MAINT command	5
LOBUFF truncation	5
New compression \$functions	5
NCMPBUF parameter	6
Chapter 4: Janus SOAP	7
\$XML_xxx functions	7
New \$XML_CHECK	7
New \$XML_COP	7
New \$XML_DEL	7
New \$XML_NODE_SING	8
Support for \$XML_PLC	8
Support for \$XML_RMV_NL	8
New \$XML_SER	8
XPath changes: predicates	8
[<n>] predicate	8
Location path within predicate	8
Existence test	9
Comparison test	9
Other changes	9
New NOEMPTYELT option	9
Remove limit on nodelists	9
contx_n checking	10
Chapter 5: Janus Network Security	11

Chapter 6: Janus Sockets	13
Chapter 7: Janus Web Server	15
URL Matching for Persistent Sessions	15
SCREEN and NOSCREEN parameters for JANUS DEFINE	15
Chapter 8: SirFact	17
APSY Maintenance Enhancements	17
Chapter 9: Compatibility/Bug fixes	23
Janus SOAP	23
Checking of contx_n arguments	23
Fixes in Sirius Mods 6.4 but not in 6.3	23
Version co-requisites	23

CHAPTER 1 ***Introduction***

This document lists the enhancements and other changes contained in *Sirius Mods* version 6.4 which was released in May, 2003. The previous generally released version of *Sirius Mods*, 6.3, was released in October, 2002.

CHAPTER 2 *Maintenance and Support***2.1 Model 204 support**

Sirius Mods version 6.4 includes support for *Model 204* version 5.3; and it no longer supports *Model 204* version V4R1.1.

2.2 Documentation

All the text in the *Sirius Mods* version 6.4 documentation is included in the Sirius Master Index, a searchable database you can download from the Sirius web site.

On the Documentation page of the web site, the download package contains a zipped collection of the Sirius PDF documents along with the Adobe-built, English-only, index (a PDX file and its supporting files), which you view with the Adobe Acrobat Reader.

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 as a section in the preface of each manual.

Since versions of *Sirius Mods* older than 5.4 are not supported, references to versions older than that have been removed from some manuals.

3.1 SIRIUS MAINT command

The SIRIUS MAINT command will display information about the current maintenance level of the *Model 204* load module.

3.2 LOBUFF truncation

Prior to version 6.4 of *Sirius Mods*, "print" output lines retrieved by \$LIST_CAPTURE or \$SOCK_CAPTURE, or redirected to a client browser by \$WEB_ON, are truncated if their length exceeds the value of the *Model 204* LOBUFF parameter setting. In version 6.4, this LOBUFF truncation remains only for \$LIST_CAPTURE.

\$LIST_CAPTURE is described in the *Sirius Functions Reference Manual*;
\$SOCK_CAPTURE is described in the *Janus Sockets Reference Manual*; \$WEB_ON is described in the *Janus Web Server Reference Manual*

3.3 New compression \$functions

\$DEFLATE and \$INFLATE are new functions which allow compression and decompression of longstrings using the "deflate" algorithm described by RFC 1951.

\$DEFLATE is described in the *Sirius Functions Reference Manual*; \$INFLATE is described in the *Sirius Functions Reference Manual*;

3.4 NCMPBUF parameter

The NCMPBUF User 0 parameter sets the number of compression buffers available for the duration of the online run.

The NCMPBUF parameter must be set by User 0 before the \$DEFLATE or \$INFLATE functions can be used or before *Janus Web Server* compression takes effect. For example, if \$DEFLATE or \$INFLATE is called with NCMPBUF = 0, the request is cancelled.

NCMPBUF should be set to the maximum number of users expected to concurrently use the compression feature. Compression buffers may also be used by Janus Web threads.

About 309K bytes are required for each NCMPBUF. These are allocated in 31-bit storage.

The following sections describe new or changed features in *Janus SOAP*.

Note that we are planning for some pervasive changes for the Janus SOAP \$functions API in version 6.5; one of these will be to remove the use of a *contx_n* argument in \$functions that have an XPath expression argument.

As a start in this direction, we have implemented \$XML_PLC and \$XML_DEL with an XPath expression argument, but no *contx_n* argument.

Part of this change will also probably change the names of \$functions. We haven't finished the new nameset, but to "leave room" for new names, we've names of \$functions formerly proposed, e.g., \$XML_PLACE is, for now, \$XML_PLC, and \$XML_REMOVE_NL is, for now, \$XML_RMV_NL.

4.1 **\$XML_xxx functions**

This section describes \$XML_xxx functions which are introduced in version 6.4 or which had been documented in version 6.3 but are only now supported.

4.1.1 **New \$XML_CHECK**

This \$function allows you to check whether an XMLDoc or nodelist ID is valid.

4.1.2 **New \$XML_COP**

This \$function allows you to copy an XMLDoc sub-tree as the child of another node, either in the same XMLDoc or in another one.

4.1.3 **New \$XML_DEL**

This \$function allows you to delete an XMLDoc sub-tree; note that if any deleted nodes had been referenced in a nodelist, they may not subsequently be referenced using a relative XPath expression.

4.1.4 New \$XML_NODE_SING

This \$function is the same as \$XML_NODES, except that it only gets the first node matching the XPath expression argument; therefore it can be more efficient for a large document, if you only need one node in the target nodelist.

4.1.5 Support for \$XML_PLC

This \$function is now supported.

4.1.6 Support for \$XML_RMV_NL

This \$function is now supported; it can be useful to remove items from a nodelist for nodes deleted by \$XML_DEL, so that the whole nodelist is still usable.

4.1.7 New \$XML_SER

This \$function serializes a sub-tree of a document, and provides an EBCDIC option. (Note that this \$function, without the EBCDIC option, was also introduced by zap in *Janus SOAP* version 6.3.)

4.2 XPath changes: predicates

This section describes forms of XPath predicates which are now supported in version 6.4.

Within a single step, only one predicate may occur, and location paths within predicates cannot themselves contain predicates.

4.2.1 [<n>] predicate

This XPath predicate is now supported; for example:

```
/book/chapter[2]/section[9]/paragraph[3]
```

4.2.2 Location path within predicate

You can now use a location path in a predicate of an XPath expression. This can take either of two forms, an “existence” test, and a “value comparison” test; simple examples of these are shown in this section. Note that the location path in a predicate can be any supported location path, including multi-step and absolute expressions.

4.2.3 Existence test

You can now use a location path as a predicate of an XPath expression; the predicate evaluates as true if the resulting nodeset is non-empty. The **usual** (but not only) purpose of this predicate is to select a node if it has at least one attribute or element of a given name; for instance:

```
/active/cust[invoice]/contact[fax]
```

The above example selects all contact children of cust elements, if the cust element has an invoice child element and the contact has a fax child element.

4.2.4 Comparison test

You can now use a location path followed by a comparison operator and literal as a predicate of an XPath expression; the predicate evaluates as true if the comparison is true of at least one node in the resulting nodeset. The **usual** (but not only) purpose of this predicate is to select a node if it has at least one attribute or element whose value bears the requested relationship to the literal; for instance (if the following two lines are combined as one XPath expression):

```
/bills/cust[@bill_dt<"20030101"]/  
contact[state="MA"]
```

The above example selects all contact children of cust elements billed before 2003, if the contact has a state child element with the value "MA".

4.3 Other changes

4.3.1 New NOEMPTYELT option

This option of the serialization \$functions specifies that an empty element be serialized with a start tag and end tag (e.g., "<foo></foo>") rather than an empty-element tag ("<foo/>"). (Note that this was also introduced by zap in *Janus SOAP* version 6.3.)

4.3.2 Remove limit on nodelists

The initial version of *Janus SOAP* had a limit on the number of nodes in a nodelist; this has been removed.

4.3.3 **contx_n checking**

The “contx_n” argument is now always checked, if you provide one explicitly: it must either be 1, if it corresponds to an XMLDoc ID argument, or less than or equal to the size of its corresponding nodelist argument, and greater than zero.

A common error which this will now detect is an “options” argument in the wrong position with an absolute XPath expression; for example:

```
* It should be:  
* .. $XML_PRINT(%D, '/', , 'INDENT 1')  
%X = $XML_PRINT(%D, '/', 'INDENT 1')
```

Note that this introduces a minor backward incompatibility (“[Checking of contx_n arguments](#)” on page 23), because such errors were not detected in version 6.3.

The following features are new or changed in *Janus Network Security*:

New security protocol option: TLS

Prior to this release, *Janus Network Security* offered support only for the SSL (Secure Sockets Layer) protocol, versions 2 and 3, created by Netscape. This release adds support for the TLS (Transport Layer Security) protocol, developed by the IETF Internet standards group. TLS is a more secure extension to SSL V3, and it is supported for both client and server Janus ports.

Since TLS is an extension and not a fundamental change to SSL, general references to SSL in the ***Janus Network Security Reference Manual*** now imply TLS as well, or they are stated as "SSL/TLS" or as "SSL-like." Explicit references to TLS are included where appropriate.

TLS support introduces only the following visible changes to Janus security implementation:

- The SSLPROT parameter of the JANUS DEFINE command has two new bit settings:
 - X'04', if a port will support only TLS.
 - X'07', if a port will support TLS, SSL v3, and SSL v2, and offers them in that order of preference. This is the new SSLPROT default.
- The \$WEB_PROTOCOL web server function may now return "4" (which corresponds to the new SSLPROT value), if TLS is being used for the connection.

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

FINCLOSE parameter for \$SOCK_CONN

This new option on \$SOCK_CONN indicates to *Janus Sockets* that if the other side of a connection closes the connection, even if it does so “cleanly” (that is with FIN rather than RESET), the connection is to be closed immediately. This is useful in situations where FIN is as good as RESET for rendering a connection unusable, and where it's important to know that a FIN has been sent to avoid wasted processing or even a hung connection.

One specific application for this capability is a *Janus Sockets* application communicating with a web server that is using a keep-alive facility (multiple requests over the same TCP/IP connection). In such a situation, the web server could close the connection between any pair of requests, and it appears most web servers do this with a FIN. Without FINCLOSE, the Janus thread that made the connection to the web server would remain in-use until the *Janus Sockets* application tried to receive data on the connection and so noticed the closed connection.

FINCLOSE and its inverse NOFINCLOSE can be specific on a \$SOCK_CONN call or, to set a port-wide default, on the JANUS DEFINE command for a port. This parameter was actually made available on the \$SOCK_CONN call as a ZAP (ZAP6386) under *Sirius Mods* version 6.3.

LOBUFF truncation

"Print" output lines retrieved by \$SOCK_CAPTURE are no longer truncated if their length exceeds the value of the *Model 204* LOBUFF parameter setting. In addition, long print lines no longer wrap at the *Model 204* OUTCCC limit, nor are they truncated at the *Model 204* OUTMRL limit.

A long print line is captured as a single “logical” line (with a single trailing LINEND string, as specified on the JANUS DEFINE command or \$SOCK_SET for the socket).

New PUSH keyword for \$SOCK_SEND and \$SOCK_SENDLN

Specifiable as the third argument for these functions, PUSH ensures that the data being sent on the socket is immediately sent to the receiver. Normally, data to be sent is buffered and may not be sent immediately.

Using PUSH is **not** necessary if:

- You are specifying FIN, since a push operation is implied by FIN.

- You are alternating sends and receives on a socket, since receive-processing functions (`$SOCK_RECV` and `$SOCK_RECVPRS`) always do a push before receiving.

CHAPTER 7 *Janus Web Server*

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

7.1 URL Matching for Persistent Sessions

Janus Web Server now only requires the initial part of the URL to match the original URL for a persistent session, that is, one that uses `$WEB_FORM_DONE`. The first `$WEB_FORM_DONE` establishes the base URL, and any request with a URL whose beginning matches this base URL (assuming `userid` and other basic criteria also match) will be considered a response to the `$WEB_FORM_DONE`.

This feature allows URL parameters to be added to a base URL in a persistent session and allows hypertext links to continue a persistent session on one of its pages. For example, if a persistent session was created by URL `/cust/update?crn=54321`, a page generated for this session could have a link like the following:

```
<a href="/cust/update?crn=54321&view=history">History</a>
```

When the user clicks on this link, the `$WEB_FORM_DONE` for the session would return. At that point, the request could examine the URL parameter (using `$WEB_ISINDEX_PARM` or `$WEB_PARM`), determine that the URL parameter `view` is set to “history”, and take appropriate action.

7.2 SCREEN and NOSCREEN parameters for JANUS DEFINE

These parameters indicate whether or not Janus Web Legacy screens will be allowed to appear on requests to the port.

`SCREEN`, the default, specifies that Janus Web Legacy support automatically renders any 3270 screen presented by an application as part of a Janus Web Legacy session. In cases where this might be viewed as a security risk — 3270 screens not intended for web user consumption might accidentally appear on a browser because of an application error — `NOSCREEN` ensures that an attempt to display a 3270 screen on a web request will result in a user restart.

You can override the `SCREEN` and `NOSCREEN` parameters on the `JANUS DEFINE` command on a URL basis using the like-named parameters in `JANUS WEB ON` rules.

The following features are new or changed in *SirFact*:

8.1 APSY Maintenance Enhancements

There is a host of new facilities available to *SirFact* customers for simplifying and improving APSY subsystem maintenance. The availability of this feature is controlled by new bits in the SIRFACT system parameter which must be set in the user0 parameters in the CCAIN stream:

- X'40'** Causes all procedures to be copied to CCATEMP when included. After the copy the share enqueue on the procedure is released. The upside of this feature is that it prevents users who are including a procedure from preventing an update to a procedure. This can be important if in a production, test, or development environment there is a need to replace a procedure to correct a problem, but one or more users may currently have the procedure included. The downside of this feature is, of course, that it incurs some extra overhead in doing the copy, though early experience suggests that the overhead may be barely measurable, if at all. Note that this feature can be useful whether a procedure is INCLUDE'd as part of an APSY subsystem or directly from *Model 204* command mode.
- X'80'** Enables the variety of APSY subsystem processing enhancements that are described below.

The subsystem enhancements can be controlled on a subsystem basis, though it is possible to change the system-wide default settings for these enhancements. Specification of the enhancements is done via a typical *Model 204* bitmask where the bits are:

- X'01'** Allows compilations where the outer or an inner procedure is in an unlocked procedure file in a procedure group to be saved, that is pre-compiled. If the outer proc is changed or added to an unlocked file in the procedure group, the procedure is re-compiled and that compilation saved.

This feature has no effect on subsystems using procedure files instead of a procedure group, and no effect on subsystems not using unlocked files in a procedure group. The feature will also pre-compile a procedure with a pre-compile prefix that was not present in the procedure group when the subsystem was started.

- X'02'** Keeps track of included procedures for pre-compiled procedures in a subsystem with unlocked procedure files in a procedure group, that is, if an included procedure is changed for a pre-compiled procedure, that procedure will be recompiled.

- X'04'** Keeps track of all the pages allocated to pre-compiled procedures in a subsystem in a bitmap. When the subsystem is stopped, this bitmap is used to free the pages rather than chaining through them. While this has a little more overhead (though probably not measurable) while saving compilations, it can make the STOP SUBSYSTEM process significantly faster. Since the bitmap is subsystem-wide and not procedure-specific, it will not improve the speed of discarding CCATEMP pages associated with a compilation that is being replaced.

Note that for the purposes of both the X'01' and X'02' bit 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.

Also, when using temporary procedure groups, a request compilation will not be 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 re-compiled. If an inner procedure but not the outer is found in a file not in the subsystem's permanent group the procedure might or might not be re-compiled if the X'02' bit is not on so might produce confusing behavior. If the X'02' bit is on the procedure will always be recompiled in such a case. As such it is recommended that where temporary procedure groups are to be used, the X'02' bit is to be set.

The bits described here can be set on a system-wide basis with the SIRAPSYF system parameter which must be set in the user0 system parameters. To override the specified or default value (X'00') of SIRAPSYF the deferred update DD name for the procedure file or group must be set in CCASYS. This should be set under "Deferred Name" under "Fileuse" in SUBSYSMGMT or be set in the APSFDN field in the SCLS records for the subsystem in CCASYS. The value of the deferred update DD name must start with the characters "TAPE*" and be followed by two hexadecimal digits that indicate the subsystem-specific bit settings for the *SirFact* APSY enhancements. For example, setting the deferred index update DD name to "TAPE*07" would use bits X'07' for the *SirFact* APSY enhancements. Note that setting a deferred index update file name is completely harmless in systems that don't have the *SirFact* APSY facility or don't have it enabled (via the SIRFACT X'80' bit).

If the SIRFACT X'80' bit is set, enabling the possibility of using the *SirFact* APSY enhancements, one feature is automatically used for all procedures. This feature releases 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. This subtle difference can make a big difference in the enqueueing behavior of procedures that end in an END statement. If a procedure ends in an END statement, that procedure is locked in share mode until procedure evaluation completes and *Model 204* attempts to get the next line

from the procedure. If the evaluation should take a long time either because it does terminal I/O or because it contains a long-running program or some other reason, that procedure would ordinarily not be updateable until evaluation completed. With that SIRFACT X'80' bit turned on this is no longer the case though cosmetic things like blank lines or comments after an END could defeat this enhancement. There is an outside possibility that a procedure might depend on the enqueue to be held on itself because it does a \$RDPROC, \$PROCOPN or something similar against itself and depends on itself not being deleted. If such a case is present at a site, it can be dealt with by simply not using the feature by turning off the SIRFACT X'80' bit or by simply adding a blank or comment line to the end of any procedure that depends on this. Note that if the SIRFACT X'40' bit is used, all procedures are dequeued even before they evaluate, so in such a case procedures cannot depend on \$RDPROC's or \$PROCOPN's against themselves working unless they can depend on this because of local policies or rules.

All these facilities make it easy to update procedures in an APSY subsystem on-the-fly, that is while the subsystem is up and in use and still have update procedures be pre-compiled. Unfortunately, if many users are inside a subsystem when changes are being made two problems can arise:

- A user can start compiling a procedure and pick up a mix of old and new versions of its inner and outer procedure. This can result in compilation errors or, perhaps even worse, incorrect application behavior.
- A user compiling a request and so doing includes can hold share locks on procedures that would prevent an update of those procedures making it impossible to fully update the subsystem's procedures. While the SIRFACT X'40' bit, which causes included procedures to be quickly copied to CCATEMP before being included, ameliorates this problem to a large degree it doesn't completely eliminate it — it is still possible for a procedure update operation to be blocked by a user in the process of copying a procedure from its source file to CCATEMP.

To help deal with these problems, *SirFact* has introduced the concept of *quiescing* a subsystem that is getting all users into a state where they are not likely to block procedure update operations or compile inconsistent copies of inner and outer procedures. While there is no obvious definition for the meaning of “quiesced” for a subsystem *SirFact* uses one based on experience in the way most subsystems are implemented that is probably good enough for most applications. This definition states that a subsystem is quiesced if all users are either evaluating a pre-compiled request (as determined by the request's prefix not its true pre-compileability) or between procedures in the subsystem driver loop. Another way to think of this is that a subsystem is quiesced if no users are inside a non-pre-compiled procedure or compiling pre-compiled procedures.

The reason that users in the evaluator inside non-pre-compiled procedures are not considered “quiesced” is because it is likely that such users are one or several include levels deep and so hold procedure enqueues (though this is less likely with SIRFACT X'40') and because typically non-pre-compiled procedure evaluations are short-lived and have complex interactions with other compilations and commands in the same request

and that some of these compilations and commands might be in different procedures, introducing the possibility of procedure inconsistencies if procedures are updated while a user is evaluating inside a non-pre-compiled procedure. Subsequent releases of *SirFact* might relax the meaning of quiesced for users in non-pre-compiled procedures if experience demonstrates this to be too restrictive but for now that's the definition.

With this definition in hand, a subsystem can be quiesced with the SIRFACT QUIESCE command and then “unquiesced” or resumed with the SIRFACT RESUME command. The format of these commands is

```
SIRFACT QUIESE subsys [WAIT sec] [BUMP]
SIRFACT RESUME subsys
```

where

subsys is the name of the subsystem to be quiesced or resumed.

sec indicates the number of seconds to wait for the subsystem to be quiesced. If the subsystem is quiesced before this time the command returns at that point. If the subsystem is still not quiesced after this time it will return with a message indicating that the subsystem is not quiesced yet unless BUMP is specified. The default wait time is 3 seconds and cannot be set to 0.

BUMP indicates that any users preventing the subsystem from being quiesced after the waittime seconds are to be BUMP'ed. After BUMP'ing these users the command will wait one more second for these users to (hopefully) notice the bump.

When a SIRFACT QUIESCE is issued for a subsystem the subsystem enters a quiescing state. While quiescing, any user that is going to switch to the next procedure in the subsystem driver will be stopped until the subsystem is resumed. In addition, any user that exits a pre-compiled procedure will be stopped at end of request. Other users will continue to run until they hit one of these two points. A user evaluating a pre-compiled request or in between procedures is already considered quiesced.

It is quite likely that even if a SIRFACT QUIESCE doesn't completely quiesce a subsystem it will do what it needs to do to allow for safe update and replacement of procedures for a subsystem — any user that hasn't quiesced within any reasonably long time frame will probably not cause or encounter any problems during an update operation especially if the SIRFACT X'40' bit is set. If despite this one wants to be even more sure that subsystem users are safely “out of the way” when procedure updates are performed, the BUMP option can be used on SIRFACT QUIESCE to bump users holding up the quiesce after a certain amount of time. Note that bumped users will not actually disappear from the system because the BUMP will cause the user's next procedure to be set to the APSY error procedure but the error procedure won't actually be run because the subsystem is quiescing — in a quiesce state users never go to the next procedure in a subsystem.

SIRFACT QUIESCE and RESUME are not additive and can both be issued as many times as desired. SIRFACT QUIESCE simply puts a subsystem into the quiesced or quiescing state and SIRFACT RESUME takes it out. That is one can issue multiple SIRFACT QUIESCE commands for the same subsystem or SIRFACT QUIESCE commands can be issued by multiple users but only one SIRFACT RESUME is required to resume the subsystem. If one user is waiting on 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.

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 (6.3).

The first sections (that is, all sections before “Fixes in ***Sirius Mods*** 6.4 but not in 6.3”) list, by product, any backwards incompatibilities, that is, any differences in processing that result from successful execution with *Sirius Mods* version 6.3, as compared with the same inputs to *Sirius Mods* version 6.4.

9.1 Janus SOAP

9.1.1 Checking of *ctx_n* arguments

The ***ctx_n*** argument is now always checked, as described in “[ctx_n checking](#)” on [page 10](#). This might cause some programs which used an invalid ***ctx_n*** argument to be cancelled.

9.2 Fixes in *Sirius Mods* 6.4 but not in 6.3

This section lists other fixes to functionality existing in the *Sirius Mods* version 6.3 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* 6.4 that are not available in previous versions.

9.3 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 6.4 of *Sirius Mods*.

- There are no corequisites associated with *Sirius Mods* 6.4.

