

---

# Sirius Mods Release Notes Version 6.5

**May, 2004**

---



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](mailto:support@sirius-software.com)  
World Wide Web: <http://sirius-software.com>

November 17, 2004

© 2004 Sirius Software, Inc.

---

## *Proprietary Notices*

The following products:

- *Janus SOAP*
- *Janus TCP/IP Base*
- *SirFact*
- *Sirius Functions*

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:

**Computer Corporation of America**  
**500 Old Connecticut Path**  
**Framingham, Massachusetts 01701**  
**USA**

**SoftSpy™** is a proprietary product of Information Technology Systems:

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

---

## *Contents*

<b>Proprietary Notices</b> . . . . .	ii
<b>Contents</b> . . . . .	iii
<b>Chapter 1: Introduction</b> . . . . .	<b>1</b>
<b>Chapter 2: Maintenance and Support</b> . . . . .	<b>3</b>
Documentation . . . . .	3
<b>Chapter 3: All or Multiple Products</b> . . . . .	<b>5</b>
Callable \$functions . . . . .	5
Case-insensitive User Language . . . . .	5
<b>Chapter 4: Janus SOAP</b> . . . . .	<b>9</b>
Object-oriented syntax . . . . .	9
Additional XPath axes . . . . .	10
Performance implications and node order . . . . .	11
More non-syntactic Janus SOAP changes . . . . .	14
Namespace support . . . . .	16
Inaccessability of namespace declarations . . . . .	16
Implicit namespace declarations . . . . .	16
Performance . . . . .	17
<b>Chapter 5: SirFact</b> . . . . .	<b>19</b>
<b>Chapter 6: Sirius \$functions</b> . . . . .	<b>21</b>
<b>Chapter 7: Janus Web Server</b> . . . . .	<b>23</b>
SEFASTLOGIN and HIGHPRIORITY . . . . .	23
<b>Chapter 8: Janus Sockets</b> . . . . .	<b>25</b>
<b>Chapter 9: Fast/Unload User Language Interface</b> . . . . .	<b>27</b>
<b>Chapter 10: Fast/Reload</b> . . . . .	<b>29</b>
Procedure reloading . . . . .	29
Preparing the environment . . . . .	29

Setting PDSIZE . . . . .	30
<b>Chapter 11: Compatibility/Bug fixes . . . . .</b>	<b>31</b>
Backwards incompatibilities . . . . .	31
Janus SOAP . . . . .	31
Control characters disallowed . . . . .	31
No subtree copying with different Namespace . . . . .	31
Default Namespace On . . . . .	32
No “[>” in serialized result . . . . .	32
Deserialization returns error position . . . . .	32
\$XML_PRINT of ill-formed XmlDocument (conversion note) . . . . .	32
Exclusive/redundant serialization options . . . . .	32
Unicode XPath comparisons . . . . .	33
Mixed case property values (conversion note) . . . . .	33
Empty XPath result (conversion note) . . . . .	33
Janus TCP/IP Base . . . . .	34
Fixes in <b>Sirius Mods</b> 6.5 but not in 6.4 . . . . .	34
Janus SOAP . . . . .	34
Version co-requisites . . . . .	35

---

**CHAPTER 1** ***Introduction***

This document lists the enhancements and other changes contained in *Sirius Mods* version 6.5, which was released in May, 2004. The previous generally-released version of the *Sirius Mods*, 6.4, was available in May, 2003.



---

**CHAPTER 2** ***Maintenance and Support*****2.1 Documentation**

Although previously reported, it is well worth mentioning again that all the text in the *Sirius Mods* version 6.5 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 in a section in the preface of each manual.

### **3.1 Callable \$functions**

This release introduces the ability to call (using a User Language CALL statement) many of the existing Sirius \$functions, relaxing the requirement that they may only be invoked as assignments to %variables. 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.

The callable \$functions are indicated as such in their individual function descriptions in Sirius documentation. They are typically those that return no value other than an indicator of whether the function completed successfully or not.

### **3.2 Case-insensitive User Language**

This release introduces the ability to compose a User Language request in case-insensitive mode. When in this mode, all elements of a User Language program excepting “literal strings” are processed after converting lowercase letters to uppercase. The non-literal string elements treated in this case-insensitive manner are:

#### **User Language keywords**

e.g., For, If, Then, eq, ...

#### **variable elements**

e.g., %variables, Screen/Image names and items, Class/Structure members; hence in this mode you cannot define two variable elements whose names differ only by some letters being different case

### labels and subroutine names

hence in this mode you cannot define two labels whose names differ only by some letters being different case

### field names

hence in this mode you cannot access field names containing lowercase letters

### \$functions

hence in this mode you cannot access \$functions containing lowercase letters

### procedure names (on the Include statement)

hence in this mode you cannot include procedures whose names contain lowercase letters

Literal strings, which are **not** affected by case-insensitive processing, are the following elements in a User Language request:

### Quoted strings

e.g., 'Caps help with veryLongNames'

### Literal contents of the Text/Html block

in the Text/Html block, case is preserved for all elements except those enclosed in the “evaluation” brackets; in the following example, `empName` and `empAddress` are references to fields named “NAME” and “ADDRESS”, respectively:

```
Html
Name: {empName}
Address: {empAddress}
End Html
```

It should be noted that case-insensitive processing only applies to program elements encountered during compilation. This applies to a multitude of string values in use in the program. For example, when setting a value for a fieldname variable or for Screen or Image indirection, the value must exactly match the name. That is, if you have a field named EMPNAME and you reference it with a fieldname variable, the value used must be all uppercase:

```
Begin
%x String Len 20
%x = 'EMPNAME'
FR
  Print %%x
End For
End
```

The case-insensitive mode of User Language processing is entered at the start of a request if the `BEGIN` command has any lowercase letters. For example, the following two requests are valid:

```
b
Print 'b is an abbreviation for begin'
End

Begin
Print 'We capitalize the first letter' And -
      'of "non-programmer defined"' And -
      'elements only by convention'
End
```

However, the following causes a compilation error, because in case-**sensitive** mode, `Print` is an unknown User Language keyword:

```
B
Print 'A single B will not do'
End
```

In addition to lower case letters in the `Begin` command which trigger case-insensitive processing, the Sirius Case compiler directive can be used to control it.

The following statement in a User Language request will turn on case-insensitive processing:

```
Sirius Case toUpper
```

The following statement in a User Language request will turn off case-insensitive processing:

```
Sirius Case Leave
```

Note that other than the `Begin` command, case-insensitive processing of User Language does not apply to *Model 204* command processing. However, this all usually works well if you use `*UPPER` for command input, and let the modifications to the *Model 204* editor (including the `SIREEDIT` parameter and the editor `CASE` command) be in effect for working on User Language.



There have been significant changes in the *Janus SOAP* API in this release. Most of these are due to changes described in “[Object-oriented syntax](#)”.

In addition to these changes, two major enhancements are described in “[Additional XPath axes](#)” on page 10 and “[Namespace support](#)” on page 16.

Other functional enhancements are described in “[More non-syntactic Janus SOAP changes](#)” on page 14, and some performance enhancements are described in “[Performance](#)” on page 17.

## 4.1 Object-oriented syntax

The new XML API for *Janus SOAP* is based on **objects**, which are acted upon or described by members such as **functions** and **properties**. For example, here is a fragment using version 6.4:

```
%D = $XML_DOC
%X = $XML_STR2DOC(%D, '<a>Hello, world!</a>')
%N = $XML_NL
%X = $XML_NODE_SING(%N, %D, '/*')
PRINT $XML_VAL(%N)
```

This could be written using version 6.5 as:

```
%d Is Object XmlDocument
%d = %d:New
Call %d:LoadXml('<a>Hello, world!</a>')
%n Is Object XmlNode
%n = %d:SelectSingleNode('/*')
Print %n:Value
```

Note: in practice, this would be simplified, because the Value property supports an XPath argument (as did \$XML\_VAL). So, you'd likely combine the last three lines as:

```
Print %d:Value('/*')
```

The new version of the *Janus SOAP Reference Manual* is being written entirely using the new form of the API. The old form of the API is obsolesced; it will be supported only for existing customers of *Janus SOAP*. These Release Notes will sometimes describe

*Janus SOAP* XML API changes in terms of the new API (especially if they are only available there) and sometimes in terms of the old API, if that seems more appropriate.

The conceptual foundation for the *Janus SOAP* object-oriented approach is described in the *Janus SOAP Reference Manual*.

## 4.2 Additional XPath axes

This version now supports all axes in XPath expressions except the `namespace` and `preceding` axes. The following are axes which were not previously supported:

**following-sibling** This could be useful to insert a node *after* the current node, e.g.:

```
%nod2 = %nod1 -
    :SelectSingleNode( -
        'following-sibling::node()')
If %nod2 Is Null Then
    %nod2 = %nod1 -
        :SelectSingleNode('..')
    Call %nod2:AddElement('foo')
Else
    Call %nod2 -
        :InsertElementBefore('foo')
End If
```

**preceding-sibling** This axis is the “mirror image” of `following-sibling`. It is not needed to accomplish “positioned insert”, because the `Insert...Before` members accomplish this. See the comments below about performance implications and node order.

**descendant-or-self** This axis is commonly employed by `//`, which is an abbreviation for the step consisting of `descendant-or-self::node()`. This axis can be used, for example, to locate an element by its name, if the “path” to it is not known:

```
//foo
```

Notes:

- The nodes in this axis are the context nodes, its children, their children, etc. Remember that the children of a node do not include its attributes, so this axis does not include any attributes; therefore, this is not a “sub-tree”, strictly speaking.
- Although this axis may seem convenient, it should be used with care; the processing time to search an `XmlDoc` using `//` can be very large.

- See the comments below about performance implications and node order.

<b>descendant</b>	This axis is similar to <code>descendant-or-self</code> , except that it does not include the context node. See all comments for <code>descendant-or-self</code> .
<b>ancestor-or-self</b>	This axis contains the context node, its parent, its parent, and so on, to and including the Root node of the XmlDocument. See the comments below about performance implications and node order.
<b>ancestor</b>	This axis is similar to <code>ancestor-or-self</code> , except that it does not include the context node. See the comments below about performance implications and node order.
<b>following</b>	This axis contains all nodes which are after the context node, in document order.

Notes:

- Unlike the `descendant` and `descendant-or-self` axes, this will contain any attribute nodes which are in that part of the document.
- This axis should be used with care; the processing time to search an XmlDocument using it can be very large.
- See the comments below about performance implications and node order.

In addition, although the `child` axis was supported in version 6.4, note that it interacts with the new axes in terms of performance implications and node order.

### 4.2.1 Performance implications and node order

Before going into detail, node order issues can be summarized as:

#### Simple axes

Use of only the `child`, `attribute`, `self`, and `following-sibling` axes, or the `parent` axis only as the first axis, presents no issues for XPath processing performance.

#### Other axes, in certain combinations:

- can increase processing time;
- lose the usual efficiency offered by single node selection (e.g., the `SelectSingleNode` member, or `Value` or various other members which have an XPath argument); this is because some cases require

searching the entire document, even after some node is selected, in case that node is not the first node in the document matching the XPath expression.

As discussed in the *Janus SOAP Reference Manual*, the items in an XmlNodeList are in **document order**, which is simply the order in which the node (or its start-tag, in the case of an Element node) occurs in the serialization of the XmlDocument. For each step in an XPath expression, there is also an order implied by the axis, which is important for the `position()` and `last()` predicate functions. For most axes, this can be considered the same as document order, except for `preceding` (which we don't support), `ancestor`, `ancestor-or-self`, and `preceding-sibling`, which occur in the reverse of document order. For example, using the following document:

```
<top>
  <a/>
  <b/>
  <c/>
  <d/>
  <e/>
</top>
```

The result of the XPath expression `/*/c/preceding-sibling::*` is the two elements named a and b, in that order, and the result of the XPath expression `/*/c/following-sibling::*` is the two elements named d and e, in that order. The result of the XPath expression `/*/c/following-sibling::*[1]` is the element named d. However, the result of the XPath expression `/*/c/preceding-sibling::*[1]` is the element named b.

*Janus SOAP* implements these semantics correctly for all XPath expressions. Sometimes XPath is intrinsically used to designate a set of nodes, that is, by the XPath expression arguments of the `SelectNodes`, `UnionSelected`, `DifferenceSelected`, and `IntersectionSelected` members. In addition to these “set-valued” members, XPath expressions can be used in many *Janus SOAP* members to operate on a single node which satisfies the expression. For example, the `SelectSingleNode` member explicitly returns one node matching its XPath expression argument, and there are a number of members (e.g., `Value`, `DeleteSubtree`, `QName`, etc.) which operate on a single node.

The “single node” XPath selection by *Janus SOAP* will return the first node in document order, but in some cases this is not the same as the **order of its internal selection algorithm**. In those cases, *Janus SOAP* will examine the entire document to find the first, in document order, node selected by the XPath expression. For example, use the following document:

```
<top>
  <a>
    <b x="1" />
  </a>
  <b x="2" />
</top>
```

- When, say, `Value('/**/b/@x')` is evaluated, the document search ends when the first match is found (and the `Value` method returns “1”).
- But when `Value('//b/@x')` is evaluated, the document search first matches on `x=2`, and it will continue searching the entire document. In this case, it turns out that the first match is not the first selected node (in document order), so continuing the search is necessary to get the correct result (and the `Value` method returns “1”).

This extra processing in single node selection can occur in the following cases:

1. the presence of the `preceding-sibling` axis
2. the presence of the `ancestor` axis, if it is not the first axis in the expression
3. the presence of the `ancestor-or-self` axis, if it is not the first axis in the expression
4. the presence of any of the following axes:
  - `descendant`
  - `descendant-or-self`
  - `following`
  - `parent`, if it is not the first axis in the expression

if that axis is followed, in a subsequent step, by any of the following axes:

- `parent`
- `child`
- `following-sibling`
- `descendant`
- `descendant-or-self`
- `following`

Note that the above considerations only apply to the “outer” XPath expression; they do not apply to any expression within a predicate. That is, there is no efficiency concern when an XPath predicate, such as the following, has an axis combination which is processed out of order internally:

```
Print %d:Value('/book/chapter' With -  
    '[./credit/details/@auth="Dave"]')
```

## 4.3 More non-syntactic Janus SOAP changes

A major enhancement to *Janus SOAP* is shown in the sub-section “[Namespace support](#)” on [page 16](#). In addition to (and sometimes as a consequence of) this, the following features are new or changed in *Janus SOAP*:

### Control characters disallowed

The deserialization and node insert functions now check for invalid characters in an XML document, e.g., “&#x19;”. As noted in “[Control characters disallowed](#)” on [page 31](#), this may cause a backward incompatibility.

### Default Namespace On

The default value of an XmlDocument's Namespace property is 'On'. As noted in “[Default Namespace On](#)” on [page 32](#), this may cause a backward incompatibility. You may find that some applications which used Namespace Ignore are now handled better (because of the increased error checking, and the improved use of prefixes with XPath) using the default Namespace property (On).

### Mixed case node type

All node type names (value of the Type property) are now mixed case, that is:

- Attribute
- Comment
- Element
- PI
- Root (note - this was formerly the DOCUMENT node)
- Text

Note: The \$Xml\_Type functions still returns all uppercase values, and the old name for the root node: ATTRIBUTE, COMMENT, ELEMENT, PI, DOCUMENT, and TEXT.

### Mixed case property values

Various property values which are of a fixed set of strings are now mixed case. When setting these values, any case variation can be used, but if you compare the value of the property to a string whose source is hand-coded, you must compare to the string exactly as follows to achieve the expected results:

- Namespace property: On, None, Ignore  
Note: The \$Xml\_Set and \$Xml\_Info functions still return all uppercase values: ON, NONE, IGNORE, but \$Xml\_Set can be passed a value in any case.
- XPathOrder property: EBCDIC, Unicode

### **Any case for option arguments**

Various option arguments, such as “ErrRet” for the LoadXml member and “NoXmlDecl” for the Serial member, can now be specified using any mixture of upper and lower case. In addition, the options to the deserialization members, such as “DTDIgnore” and “WspPreserve” for LoadXml, can now be specified without the underscore previously used. The underscore is still accepted, but it is considered vestigial.

(All these forms of options are also accepted by the corresponding \$Xml\_xxx functions).

### **Boolean predicates**

The **and** and **or** Boolean operators can be used in XPath predicates, for example:

```
Print %n:SelectSingleNode( -  
    'pay[@per="day" and @due="late"]')-  
:Value
```

### **Full position() support**

The XPath position() function is supported (not simply the “abbreviation” of a number in square brackets, e.g., [3]), and it may be followed in a predicate with a comparison operator (e.g., >=) and a numeric value. The numeric value used for position (both with explicit position() and the “abbreviated” form) may now be any integer, including zero and negative integers.

### **AddSubtree disallowed with different Namespace**

It is illegal to copy a subtree from one XmlDocument to another which has a different value of the Namespace property. As noted in [“No subtree copying with different Namespace” on page 31](#), this may cause a backward incompatibility.

### **No “]]>” in serialized result**

In accord with the XML Recommendation, an Element's content representation (in the serialized form, of course) cannot contain the character string “]]>”. Previously, this could occur. As noted in [“No “\]\]>” in serialized result” on page 32](#), this bug fix may cause a backward incompatibility.

### **Print of “incomplete” XmlDocument**

The Print subroutine allows printing from the root node of an XmlDocument which does not contain an Element. This is useful since Print is used for debugging, and so it should not fail in such cases.

### **Deserialization returns error position**

The deserialization functions return the position of an input character which is at or after the cause of a parsing error (when the ErrRet option is used). In version 6.4, the \$function forms returned the value “1” to indicate a parse

error. As noted in “[Deserialization returns error position](#)” on page 32, this may cause a backward incompatibility.

#### **Deserialization errors provide element/attribute name**

During deserialization, several types of errors refer to an element start-tag or attribute which preceded the discovery of the error, for example, a mismatched element end-tag. The messages for these errors now include the name of the element or attribute involved in the error, thus making the error easier to find.

## **4.4 Namespace support**

*Janus SOAP* now supports namespaces. Although version 6.4 did not, there was description of how the support would work. There have been several changes to this; some are straightforward. For instance, the `AddAttribute` function is only used to insert Attribute nodes; the version 6.4 documentation assumed that it would also be used to insert Namespace nodes, which are instead inserted using `AddNamespace`.

This section explains changes to the proposed handling of namespaces which go beyond such straightforward changes.

### **4.4.1 Inaccessibility of namespace declarations**

One aspect of the XPath 2 specification which is implemented in *Janus SOAP* namespace support is that namespace declarations are not directly accessible. That is, the namespace axis is not supported; also, there is no need to identify a node type (formerly called Namespace) for them. You can obtain the information provided by namespace declarations by various `Xml*` class members, for example, the `URI` property of an `XmlNode`.

### **4.4.2 Implicit namespace declarations**

When an Element node (for example, with `AddElement`) or Attribute node (with `AddAttribute`) is inserted in an `XmlDoc`, specifying the `URI` argument indicates the namespace URI of the node (including “no namespace”, when the null string is given as the `URI` argument, which is different from omitting the argument).

If necessary, a namespace declaration is implicitly inserted in the newly inserted Element node (or, when inserting an Attribute, in its Element parent). This is one of the two styles of namespace control which you can use when inserting nodes (the other is to omit the `URI` argument). Whichever style you use, or if you create an `XmlDoc` with one of the deserialization functions, the `URI` which is set for an Element or Attribute node remains the `URI` of that node. For example, if you copy or move the node, that `URI` remains as the `URI`, and if an implicit namespace declaration is needed as part of the copy or move operation, one is inserted.

There could be occasions in your application when you insert a node with the URI argument, and so an implicit declaration may be inserted on an Element. Subsequently, you may want to ensure a specific declaration on that Element, for example, if you are inserting descendants of it using the “no URI argument style”. Therefore,

- When AddNamespace is called with the same prefix and URI as a declaration which already exists for the Element, no error occurs.

## 4.5 Performance

There have been several changes to *Janus SOAP* to enhance its performance. Besides overall changes, the following two deserve mention:

**Hashing** All strings (names or values) are stored in an XmlDocument as part of a hash table. The hashing algorithm and its implementation have been improved to reduce path length and to reduce hash collisions. This will improve serialization, node insertion, and XPath operation.

**Depth-first XPath** The approach to evaluating XPath expressions has been changed so that now it is in depth-first order of the XmlDocument tree, which usually will coincide with the stored order of XmlDocument nodes, and will also reduce the number of needless nodes searched when a “single node” member is invoked.

In addition to the above improvements, you may be able to improve the performance of your *Janus SOAP* applications by ensuring that the *Model 204* **MAXOBUF** parameter is large enough; it should probably be a minimum of 6 if your XmlDocuments are heavily used or contain a number of nodes and values in the thousands or more.

MAXOBUF can be extremely beneficial for a large number of *Model 204* applications, not just those involving *Janus SOAP*.



---

 CHAPTER 5 *SirFact*

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

### INFO.VERSIONS

You can display INFO.VERSIONS to get the *Model 204* version, the *Sirius Mods* version, and the *SirFact* internal dump format version in effect when the *SirFact* dump was produced.

### Display XmlDocument and XmlNode objects

You can use the VALUE and ATTRIBUTE classes to display an XmlDocument or XmlNode object. The VALUE class of display, which can also be used to display one or more elements of an array of XmlDocument or XmlNode objects, consists of the same output lines as the result of the Print method.

### Display XmlNodeList objects

You can use the VALUE and ATTRIBUTE classes to display an XmlNodeList object. The VALUE class of display consists of the same output lines as the result of the Print method applied to the Item method; in order to specify which item of the XmlNodeList to display, use a colon followed by the item number as the suffix for the specification. The following example displays the XmlDocument subtree starting at the node which is the third item in the %nodlis XmlNodeList:

```
D %NODLIS:3
```

To request the count of items in an XmlNodeList, use a colon and “number sign” (“#”) as a suffix. The following example displays the number of items in the %nodlis XmlNodeList:

```
D %NODLIS:#
```

The VALUE class of display can also be used to display one or more elements of an array of XmlNodeList objects; for that, the colon and item number or “#” is specified after the parenthesized array subscripts.

### Colon syntax for \$list items

The same “:<number>” and “:.” suffix notation can be used with the LIST display class, as an alternative to a single item in parentheses. So, to display a single \$list item:

```
D L.%LIS:3
```

To display the number of items in a \$list:

D L.%LIS:#

---

**CHAPTER 6** *Sirius \$functions*

The following are new or changed features in the *Sirius Functions*:

**New \$Lstr\_Translate function**

This function takes a string or longstring input and replaces characters indicated by an “input table” with corresponding characters in an “output table.”



---

**CHAPTER 7**    *Janus Web Server*

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

## **7.1    SESFASTLOGIN and HIGHPRIORITY**

These JANUS DEFINE parameters are valid only for Janus Web ports. They are designed, respectively, to reduce the login security overhead for Janus Web threads, and to have web threads start at the upper end of their priority range rather than the default start.

### **SESFASTLOGIN**

Indicates that a login performed for a continued session will be a “fast” login. A fast login does no CCASTAT lookup or external authorizer (RACF, ACF2, Top Secret) lookup for the userid.

The advantages of this parameter are:

- It avoids the overhead of heavy external-authorizer login traffic for web threads, which do a login for every protected page requested. The overhead of CCASTAT lookup is also avoided for users in CCASTAT, but this overhead is usually relatively small.
- It avoids the external authorizer's logging of “last logged in” times for every page a user accesses. This presents an inaccurate picture of the time of last login, and it produces large amounts of useless external authorizer logging.

The disadvantage of this setting is that access to an external-authorizer protected resource or to command privileges will only be available to the first URL request in a session.

**Note:** This parameter has no effect unless login sessions are being maintained by SESCOOKIE or SSLSES.

The WEBLOGHOLD parameter accomplishes many of the same things as SESFASTLOGIN, but it has the drawback that an sdaemon thread is tied up for each held login. It has the advantage that there is no difference privilege-wise between an initial user URL request and a later one.

**HIGHPRIORITY**

Indicates that upon login, a web user's priority is set to the top of its priority range. *Model 204* has three basic priority classes:

**HIGH**                Ranges from 80 to 127

**STANDARD**        Ranges from 32 to 79

**LOW**                Ranges from 0 to 47

By default, when a user logs in, they get a privilege 16 greater than the bottom of its class. This can be a problem for web threads, because the priorities of 3270 threads often “drift” to the top of their range. As a result, the web threads ultimately have lower priorities than 3270 threads. If you set **HIGHPRIORITY**, web threads will start out at the top of their range, so they will tend to get as good or better service than 3270 threads.

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

### FTP support

*Janus Sockets* FTP support allows you to set up one or more FTP servers within a *Model 204* online. Janus FTP servers can be accessed with the FTP client of your choice to copy procedures into and out of *Model 204* procedure files. The FTP client may be running on any platform that can make a TCP/IP connection to the online.

Janus FTP features include:

- Adding, replacing, deleting, and renaming *Model 204* procedures with any FTP client.
- Viewable procedure lists (the FTP ls and dir commands).
- EBCDIC/ASCII translation in both directions.
- Binary file transfers (FTP TYPE I).
- Mapping of the standard UNIX folder structure to *Model 204* procedures and procedure files, enabling navigation with the standard FTP CWD command.
- FTP User authentication based on *Model 204* user ids and passwords using your security package.
- Anonymous FTP.
- Active and passive FTP.
- Support for MVS, VM, and VSE.
- Multiple FTP servers running on different port numbers in an online, not necessarily using the default port (21).
- Letting the procedure name suffix (say, .HTML) control the transfer mode of a file (text vs. binary).

### Socket object support

*Janus Sockets* Socket object support offers a coding alternative to the \$SOCK\_xxxx functions for both client and server applications that spares you from having to manage or refer to socket numbers.

Essentially an object style reference to a Socket number, the Sirius Socket object is designed for *Janus SOAP* applications only. Socket objects and \$functions may be used in the same code, and transition methods are available for converting \$function applications to object-oriented applications.

### HTTP Helper object support

The *Janus Sockets* HTTP Helper is a simple object-based API that lets you write a User Language HTTP client without knowledge of socket level

programming or the format of HTTP requests and responses. You can construct HTTP requests for any HTTP server (including Janus Web server).

For SOAP applications, the HTTP Helper supports the posting of XML data and parsing an XML document in an HTTP response. HTTP Get and Post requests are supported as are proxy servers and full HTTP header access.

The following feature is new in the *Fast/Unload User Language Interface*:

#### **\$FUNLOAD and multiple output streams**

The multiple output feature was introduced in *Fast/Unload* 4.1, but not supported for the *Fast/Unload User Language Interface* until *Fast/Unload* version 4.2 (and *Sirius Mods* 6.5).

To use \$FUNLOAD to invoke a multiple output version 4.2 *Fast/Unload*, you specify any character string (eight-character maximum and not a %variable or asterisk) as the \$FUNLOAD fourth argument. This string is not used, but it indicates that the output data is to be sent to the data sets specified in the FUEL program.

Prior to *Fast/Unload* 4.2 and *Sirius Mods* 6.5, such a character string in a \$FUNLOAD fourth argument was assumed to name an output data set specified in the FUEL program.

Specifying multiple outputs in a *Fast/Unload* 4.1 FUEL program that is invoked by \$FUNLOAD is an error, regardless of the *Sirius Mods* version.



---

---

## CHAPTER 10 *Fast/Reload*

The following feature is new in *Fast/Reload*:

### 10.1 Procedure reloading

As of *Fast/Unload* version 4.2 and *Sirius Mods* 6.5, the UAI and LAI statements support the reorganization of procedures and procedure aliases. Procedures and aliases present in the TAPEI input data set for a *Fast/Reload* run are loaded as is by LAI. The UAI unload and LAI reload preserve their associations.

If procedure or alias names clash with names in the target file, the reload stops. If you want the unloaded procedures and aliases not to be reloaded, you specify the NOPROCS option of the LAI statement.

The procedure reloading feature likely entails some environmental adjustments (especially more storage buffers), and it offers an automatic tuning of the target file's procedure dictionary settings.

#### 10.1.1 Preparing the environment

When you are LAI-reloading a file that has procedures, take into consideration the following recommendations:

- More CSECTs are required than in the typical BATCH204 module.
- Instead of maintaining separate BATCH204 and ONLINE modules, combine their CSECTs into a single ONLINE module, and use that for *Fast/Reload*.
- Ensure sufficient disk buffers to minimize the elapsed time for the reload:

If your storage capacity is plentiful, 10,000 to 15,000 buffers should ensure maximum throughput.

- Set the *Model 204* parameter MINBUF to at least 10000.
- Set the LDKBMWND parameter to at least 30.

Otherwise, if storage capacity is a concern, set the MAXBUF parameter to the number of pages in the unloaded procedure dictionary, plus some overhead:

$$\text{MAXBUF} = \text{PDpgs} + \text{LDKBMWND} + 50 + \min(10, \text{nprocs}) * \text{avgProcpgs}$$

Where:

- `PDpgs` (total number of pages in the unloaded procedure dictionary), `nprocs` (number of unloaded procedures), and `avgProcpgs` (average procedure length in pages) are available from the FSTATS statistics in the UAI report data set.
- The `LDKBMWND` parameter is set to at least 30.

### 10.1.2 Setting PDSIZE

Procedure loading includes an automatic PDSIZE-setting feature. This auto-sizing is weighted toward creating a new procedure dictionary whose total size is nearly the same as the old, but which typically is a product of a larger PDSIZE and fewer page-blocks than the old.

The sizing feature is invoked only when you are loading into a file that has no procedures, the file's PDSIZE value is the default size (3), and the number of pages of UAI unloaded procedures and aliases is greater than the default PDSIZE.

When the preceding conditions exist, the PDSIZE of the reload target file is set to one of the following:

- The number of pages in the unloaded file's procedure dictionary, if that number of pages is less than or equal to 255.
- The smallest number less than 255 that will yield the smallest number of page-blocks, if the number of pages in the unloaded procedure dictionary exceeds 255.

To arrive at these smallest numbers, (the rounded-up, integral result of) the number of pages in the unloaded procedure dictionary is divided by the initial number of page blocks in the new dictionary. This page-blocks number is the quotient of the old page count and 255, rounded up to the next integral value.

For example, for an unloaded dictionary that has 800 pages, the new page-block count would be 4, and the new PDSIZE would be 200. For an unloaded dictionary that has 1332 pages, the new page-block count would be 6, and the new PDSIZE would be 222.

The target file's existing value for PDSIZE is used in any of the following cases:

- A procedure dictionary already exists.
- PDSIZE is already set to a value other than 3.
- The unloaded procedures and aliases used no more than 3 pages.

**Note:** The PDSIZE optimization feature is only activated if the *Fast/Unload* user has the FSTATS feature enabled. If FSTATS is **not** enabled, the optimization information is not written to the unloaded data set, nor are procedure statistics displayed in the report data set.

---

**CHAPTER 11** *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 (6.4).

In general, backward incompatibility means that an operation which was previously performed without any indication of error, now operates, given the same inputs and conditions, in a different manner. We may not list as backwards incompatibilities those cases in which the previous behaviour, although not indicating an error, was “clearly and obviously” incorrect, and which are introduced as normal bug fixes (whether or not they had been fixed with previous maintenance).

## **11.1 Backwards incompatibilities**

### **11.1.1 Janus SOAP**

Version 6.5 introduces a new API that uses object-oriented methods instead of \$functions. Many of the methods perform the same or nearly the same operation as, and have names that closely resemble, their \$function counterparts. The subsections below include some cases where version 6.5 methods differ slightly but significantly from their otherwise matching \$functions.

#### **11.1.1.1 Control characters disallowed**

The deserialization and \$XML\_INSxxx functions now check for invalid characters in an XML document, e.g., “&#x19;”. This is in accordance with the XML Recommendation, but since previous versions of *Janus SOAP* did not check for valid characters, you may now obtain a request-cancelling error message due to an invalid character, in an application which formerly inserted the illegal character without any error message.

#### **11.1.1.2 No subtree copying with different Namespace**

It is illegal to copy a subtree from one XmlDocument to another with different values of the Namespace property. Since previous versions of *Janus SOAP* did not enforce this, this may cause a backward incompatibility.

### **11.1.1.3 Default Namespace On**

The default value of an XmlDocument's Namespace property is 'On'. In previous versions of *Janus SOAP*, OFF was the default. The only real backward incompatibility this causes is if you display or otherwise process the value of the Namespace property, because the other consequence of the default is that previously the use of namespaces was rejected.

Moreover, you may find that some applications which were handled using Namespace Ignore are now handled better (because of the increased error checking, and the improved use of prefixes with XPath) using the default Namespace property (On).

### **11.1.1.4 No “]]>” in serialized result**

In accord with the XML Recommendation, an Element's content representation (in the serialized form, of course) cannot contain the character string “]]>”. If an XML document contains this string, it was previously incorrectly serialized as “]]>” but now is serialized as “]]&gt;”.

### **11.1.1.5 Deserialization returns error position**

The deserialization functions return the position of an input character which is at or before the cause of a parsing error (when the ErrRet option is used). In version 6.4, the \$function forms returned the value “1” to indicate a parse error. Although it was documented that a positive number indicates a parsing error, this could cause a backwards incompatibility if you check for the value “1”, rather than testing for a non-0 value.

### **11.1.1.6 \$XML\_PRINT of ill-formed XmlDocument (conversion note)**

In version 6.4, the \$XML\_PRINT function was not allowed if the top of the subtree to be printed was the Root node, and the XmlDocument did not contain an Element (an XmlDocument in this form is called “not well-formed” - every well-formed XmlDocument must contain an Element node).

Since the \$XML\_PRINT function was designed to support debugging, not serialization of an XmlDocument, this is now allowed in the version 6.5 Print method, the replacement for \$XML\_PRINT.

### **11.1.1.7 Exclusive/redundant serialization options**

In version 6.4, the serialization \$functions (e.g., \$XML\_PRINT) allowed an option to be specified multiple times, for example:

```
%X = $XML_PRINT(%D, , , 'NOXMLDECL NOXMLDECL')
```

Mutually exclusive options were also allowed, with the last one having effect:

```
%X = $XML_PRINT(%D, , , 'NOXMLDECL ALLOWXMLDECL')
```

Both of these cases are now treated as errors.

#### **11.1.1.8 Unicode XPath comparisons**

In accord with the XPath Recommendation, an ordered comparison in an XPath expression is performed using the Unicode collating sequence. In some cases, this differs from the EBCDIC collating sequence, which was used in version 6.4:

- Lower-case letters are greater than uppercase in Unicode.
- Letters are greater than numbers in Unicode.

The above orderings are the reverse in EBCDIC; also the following orderings are different in Unicode and EBCDIC:

- ordering among non-alphanumeric characters
- ordering between non-alphanumeric and alphanumeric characters

If you want all XPath expressions on an XmlDocument to use EBCDIC collation, set the XPathOrder property to `EBCDIC`. You can change it back to use Unicode collation by setting the XPathOrder property to `Unicode`.

#### **11.1.1.9 Mixed case property values (conversion note)**

Although this not actually a compatibility issue, you should take note, if you are converting from `$Xml_Type` to the `Type` property, or from `$Xml_Info('Namespace')` or `$Xml_Set('Namespace', value)` to the `Namespace` property, that these properties return values with the initial letter capitalized and the rest of the value lowercase (and, for `Type`, the name of the root of an XmlDocument tree is 'Root', rather than the former 'DOCUMENT').

#### **11.1.1.10 Empty XPath result (conversion note)**

Although this not actually a compatibility issue, you should take note, that an empty XPath result causes request cancellation with the following methods:

- Length
- LocalName
- Prefix
- QName
- URI

- Print
- Serial
- Type
- Value (however, the method ValueDefault does not)

The like-named \$Xml\_xxx functions did not cancel the request in these circumstances, but rather had a special result defined for the empty XPath result; for example, the following statement will print the null string (because the Root node does not have a parent):

```
Print $Xml_Type('/..')
```

### 11.1.2 Janus TCP/IP Base

The following backwards incompatibilities are introduced in *Janus TCP/IP Base*:

#### **JANUS DEBUG not allowed**

Formerly, you could use `JANUS DEBUG` as an alias of the `JANUS TRACE` command (which specifies what kind of audit trail tracing to perform on one or more JANUS ports or on some selected connections on the ports). Use of this alias is no longer available.

## 11.2 Fixes in *Sirius Mods 6.5* but not in 6.4

This section lists fixes to functionality existing in the *Sirius Mods* version 6.4 but which, due to the absence of customer problems, have not, as of the date of the release, been fixed in that version.

### 11.2.1 Janus SOAP

#### **Retain Namespace property after parse failure**

Previously, an error during deserialization of an XML document would completely re-initialize the XmlDocument, causing the Namespace setting to be lost. Deserialization errors now will retain the value of the Namespace property.

#### **Detect “?>” in all \$XML\_INSPI cases**

Previously, if the value (argument 3) of \$XML\_INSPI contained the characters “?>” (which are invalid in that context), for a very small number of such cases, the error was not detected.

#### **Duplicate \$XML\_NODES/PLC/COUNT with “..”**

Previously, use of the `parent` axis in an XPath expression could produce duplicate nodes as the result of \$XML\_NODES or \$XML\_PLC, or in an incorrectly large value as the result of \$XML\_COUNT.

**Control characters disallowed**

This bug fix is mentioned as a compatibility issue in “Control characters disallowed” on page 31.

**No “[>” in serialized result**

This bug fix is mentioned as a compatibility issue in “No “[>” in serialized result” on page 32.

**Unicode XPath comparisons**

This bug fix is mentioned as a compatibility issue in “Unicode XPath comparisons” on page 33.

## **11.3 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 6.5 of *Sirius Mods*.

- There are no corequisites associated with *Sirius Mods* 6.5.

