
Notes for Sirius Mods Release 6.9

August, 2006



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 6, 2010

© 2010 Sirius Software, Inc.

Proprietary Notices

The following products:

- *Fast/Reload*
- *Janus Debugger*
- *Janus SOAP*
- *Janus Sockets*
- *Janus Web Server*
- *Sirius Debugger*
- *Sirius Functions*
- *SirTune*

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

Contents

Proprietary Notices	ii
Contents	iii
Chapter 1: Introduction	1
Chapter 2: Maintenance and Support	3
Model 204 support	3
Chapter 3: All or Multiple Products	5
FUNCOPTS system parameter X'02' bit	5
Argument name provided on request cancellation messages	5
Chapter 4: Fast/Reload	7
Chapter 5: Janus Debugger and Sirius Debugger	9
Chapter 6: Janus SOAP ULI	11
Narrowing Assignments and Class Tests	11
Chapter 7: Janus SOAP Stringlist Class	15
RegexCapture method	15
RegexLocate and RegexLocateUp methods	18
RegexReplaceCorresponding method	22
RegexSubset method	26
FindUp method	29
Chapter 8: Janus SOAP Xml Classes	31
DefaultURI function	31
PrefixURI function	32
InvalidCharacterPosition function	33
New DefaultURI argument to AddSubtree and InsertSubtreeBefore	33
Relaxation of Namespace property and AddSubtree/InsertSubtreeBefore	36
New SortCanonical option for serialization methods	36

Chapter 9: Janus Sockets	39
Telnet Server Support	39
Configuring a Janus Telnet Server	40
New JANUS DEFINE parameters	41
AUTOSYS subsys	41
TCPKEEPALIVE	41
WSFQUERY	42
TRUST parameter added to \$Sir_login	43
Chapter 10: Sirius Functions	45
New wait time parameter for \$Bind	45
\$RegexMatch: Whether string matches regex	45
Regex rules	48
\$RegexReplace: Replace matching strings	51
Updated mathematical functions	55
Chapter 11: SirTune	57
Compatibility	57
Changes	57
Chapter 12: Compatibility/Bug fixes	59
Backwards incompatibilities	59
Janus SOAP XML processing	59
XPath with “!=” and Element with more than 1 child	59
Fixed XPath “following” and “xx-sibling” axes	60
Corrected XPath results with unusual axes	60
Performance considerations: Document order, certain axes	62
Fixes in Sirius Mods 6.9 but not in 6.8	66
Reject invalid EndTag in Xml* deserialization methods	66
Accept UTF-16 input in Xml* deserialization methods	66
Version corequisites	66

CHAPTER 1 ***Introduction***

This document lists the enhancements and other changes contained in *Sirius Mods* version 6.9, which was released in October, 2006. The previous version of the *Sirius Mods*, 6.8, was available in November, 2005.

CHAPTER 2 *Maintenance and Support*

2.1 Model 204 support

Sirius Mods version 6.9 supports *Model 204 V6R1* only.

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 FUNCOPTS system parameter X'02' bit

The X'02' bit of the FUNCOPTS system parameter causes all \$list functions that encounter CCATEMP full conditions to act as if the LISTFC \$sirparm were set to 1, that is, to cancel the request with a CCATEMP full condition.

This feature was also implemented via a maintenance zap in version 6.8 of the *Sirius Mods*.

3.2 Argument name provided on request cancellation messages

Many internal Sirius methods (that is, those in the `System` namespace) provide an error message if an argument is valid. Previously, for named arguments, an argument number was provided to identify the argument. Now, the name of the argument is provided in the error message.

CHAPTER 4 ***Fast/Reload***

Although this fix has been provided via maintenance to all supported releases, it is worth noting that the handling of float values in *Fast/Reload* has been improved. The most significant aspect of this is that using UAI/LAI to reorganize a file with, in particular, FLOAT LEN 4 fields, and changing them to FLOAT LEN 8, now produces values which are correct according to standard *Model 204* float handling. The explanation of float field processing has been elaborated in the ***Fast/Reload Reference Manual***.

Janus Debugger and Sirius Debugger

First available in later releases of Version 6.8 of the *Sirius Mods*, these two products include some features that are available only with *Sirius Mods* 6.9.

The *Janus Debugger* is a tool designed for software developers who create and maintain Janus Web Server applications. The *Sirius Debugger* is designed to debug applications that use a 3270 terminal stream to interact with the user or applications that use a Batch 2 stream. Though the two debuggers use different threads on the host, they can run concurrently, and they both use the same Windows-based GUI client.

For more information about the debuggers, see the ***Janus/Sirius Debugger User's Guide*** on the Sirius web site Documentation page.

The Debugger features available only with version 6.9 and higher of the *Sirius Mods* are:

- **DebuggerTools class methods:** Under Version 6.9 of the *Sirius Mods*, you can use methods from the DebuggerTools system object in your User Language code to aid in the debugging process in either the *Janus Debugger* or the *Sirius Debugger*. Available now are the AmDebugging and Break methods.
- **Janus Web Legacy Support:** With Version 6.9 of the *Sirius Mods*, you may use the *Janus Debugger* against a *Janus Web Legacy Support* thread if you also have a license for the *Sirius Debugger*.
- **Setting variables:** From the Debugger GUI, you can now modify the current value of a scalar variable in the program you are debugging.
- **APSY details:** The information that identifies the procedure file and name to which a selected source code line belongs will now also identify the APSY subsystem within which the procedure was invoked, if any.
- **Wildcards on “Until” processing:** You can use a single trailing wildcard asterisk (*), for example, P.M*, to identify the procedure at which you want the Debugger to pause program execution.
- **DEBUGPAG:** The DEBUGPAG User 0 parameter defaults to 10 under *Sirius Mods* version 6.8. Under *Sirius Mods* version 6.9, the default is 100.

The following sections describe changes in the *Janus SOAP ULI* in this release.

6.1 Narrowing Assignments and Class Tests

Version 6.9 of *Sirius Mods* introduces support for narrowing assignments and class tests. As discussed in the *Janus SOAP Reference Manual*, these should not be done lightly and, in fact, should be avoided as much as possible. For this reason, by default, class tests and narrowing assignments are not allowed for any class. However, if class tests or narrowing assignments are unavoidable for a class, an `Allow Narrow` clause must be placed in the Public block of that base class.

For example, the Mammal class could have the following:

```
class mammal
  public
    allow narrow
    variable weight    is float
    variable name      is string len 32
    variable home      is string len 32
    ...
  end public
end class
```

Then, if you have a class Marsupial that extends the Mammal class, you could do something like the following:

```
%pouchy    is object marsupial
%furry     is object mammal
...
if %furry:name = 'Skippy' then
  %pouchy = %furry:(marsupial)
end if
```

In this example, if earlier lines in the program did not make `%furry` actually reference a Marsupial object, and it simply referenced a simple Mammal object or, perhaps, some other extension of class Mammal (say object Feline), the assignment of `%furry` to `%pouchy` would fail and the request would be cancelled.

Note that narrowing assignment cannot be done by simply assigning `%furry` to `%pouchy`. It has to be indicated by a pseudo-method of the target class name in parentheses. The term pseudo-method is used because no actual code is executed in the `(marsupial)`

operation and no transformation of the input object is performed — if %furry references a marsupial, the operation would succeed, otherwise it would fail. The narrowing pseudo-method will always fail if the source object is null.

With this syntax, it is possible to string member names to a narrowing assignment pseudo-method. For example, if the Marsupial class in the above example had a PouchSize member, one could do something like:

```
if %furry:name = 'Skippy' then
  print %furry:(marsupial):pouchSize
end if
```

One could also pass a narrowing assignment pseudo-method output as a method parameter:

```
%zoo:addMarsupial(%furry:(marsupial))
```

One could even do something as silly as

```
print %furry:(marsupial):(mammal)weight
```

This (marsupial) does a narrowing assignment of %furry to an internal work variable. Then, the (mammal)weight accesses the Mammal base class member Weight via that variable.

Of course, the above is identical to:

```
print %furry:weight
```

with the exception that the silly example would cancel the request if %furry did not reference a marsupial. It is worth emphasizing that the (marsupial) and the (mammal) in the above example serve two very different functions. The (marsupial) indicates a narrowing assignment to an extension class. The (mammal) indicates selection of a member of a base class.

One might wish to do narrowing assignments on the basis of the class of the object pointed to by an object variable. In the above example, %furry is declared as being a Mammal object but, via polymorphism, can be pointing at an object of some extension class, including, of course, class Marsupial. The `Is Instance Of` keywords are provided to facilitate testing whether the underlying object referenced by an object variable is of a specific class.

The following is an example using `Is Instance Of`:

```
%pouchy   is object marsupial
%furry    is object mammal
...
if %furry is instance of marsupial then
  %pouchy = %furry:(marsupial)
end if
```

In fact, one can perform class-specific processing based on such tests:

```
%furry    is object mammal
...
if %furry is instance of marsupial then
    %furry:home = 'Australia'
end if
```

This kind of code is **strongly** discouraged, and is a formula for long-term code maintainability problems. What will happen to the above code, for example, when used on an opossum which is one of the few marsupials that doesn't necessarily live in Australia?

The `Is Instance Of` keywords must be followed by the name of a class that is an extension class of the type of the object variable that precedes the keywords. Note that no system classes are currently Allow Narrow.

Janus SOAP Stringlist Class

The following sections describe changes in the *Janus SOAP* Stringlist class in this release.

7.1 **RegexCapture method**

This function applies a regular expression, or “regex,” to a given input string, and it obtains those characters in the string that match the “capturing groups” in the regex.

Any parenthesized sub-expression in the regex is a capturing group by default — unless its opening or closing parenthesis is escaped (preceded) by a backslash (\), or unless the opening parenthesis is followed by a question mark (?). Each captured set of characters is appended to the method object Stringlist as a separate item.

RegexCapture uses the “rules” of regular expression matching (information about which is provided in “[Regex rules](#)” on page 48).

<pre>[%retcode =] %s1:RegexCapture(inStr, regex, - [, Options=%str] - [, Status=%num])</pre>
--

RegexCapture Function

%retcode, if specified, is a number that is either 0 if the regular expression was invalid or no match was found, or the position of the character **after** the last character matched.

RegexCapture accepts two required and two optional arguments, and it returns a numeric value.

- The first argument is the input string, to which the regular expression *regex* is applied. This longstring is a required argument.
- The second argument is a string that is interpreted as a regular expression and is applied to the *inStr* argument to determine whether the regex matches *inStr*. This longstring is a required argument.
- The Options argument (name required) is an optional string of options. The options are single letters, which may be specified in uppercase or lowercase, in any combination, and separated by blanks or not separated:

! Do case-insensitive matching between *inStr* and *regex*.

S DOTALL mode. If this mode is **not** specified, a dot (.), also called a point, matches any single character except X'0D' (carriage return) and X'25' (linefeed). In DOTALL mode, a dot (.) also matches carriage return and linefeed characters.

Note: This is different from Perl, which does not consider a carriage return an end-of-line character. In Perl, a dot matches a carriage return even when not in DOTALL mode.

M Multi-line mode. If this mode is **not** specified, a caret (^) or a not sign (¬) — whichever key your keyboard program translates to X'5F' — matches only the position at the very start of the string, and dollar sign (\$) matches only the position at the very end. (This documentation uses the caret.)

The caret and dollar sign are position-identifying characters known as “anchors,” which match the beginning and end, respectively, of a line or string. They do not match any text.

In M mode, a caret **also** matches the position immediately after any end-of-line indicator (carriage return, linefeed, carriage return linefeed), and a dollar sign **also** matches the position immediately before any end-of-line indicator.

C XML Schema mode. Do the match according to XML Schema rules: In a regex, no characters serve as anchors, and any regex is always assumed to be anchored at both ends. For example, the regex `X` in C mode is equivalent to `^(?:X)$` in non-C mode, where the `(?:` indicates a “non-capturing” group.

In addition, contrary to non-C mode:

- The usual anchoring-atoms, `^` and `$`, are treated as ordinary characters in a regex, and you may not escape them (the escaping rules do not change).
- The two-character sequence `(?` is not valid in a regex.
- The dot metacharacter `.` matches neither a carriage return character nor a linefeed.

You may want to use C mode for testing a regex for inclusion in a schema document (an XML document that constitutes an XML schema).

If you specify an invalid option, the request is cancelled.

- The Status argument (name required) is optional; if specified, it is set to an integer code. These values are possible:
 - >0 A successful match was obtained. The position of the character **after** the last character matched.

- 0** No match: *inStr* not matched by *regex*.
- 1** The pattern in *regex* is invalid.

Note: If you omit this argument and a negative Status value is to be returned, the run is cancelled.

RegexCapture is callable.

Notes:

- RegexCapture requires considerable user stack space. It is recommended that you increase the value of the *Model 204* LPDLST parameter to a minimum of 8000 (using, for example, `UTABLE LPDLST 8000`).
- If *%retcode* is 0, either *regex* did not match *inStr*, or there was an error in the regex. The Status argument returns additional information: If it is negative, it indicates an error. If it is zero, it indicates there was no error, but the regex did not match.
- Even with a Status value of 1, which indicates a successful match, it is possible that zero items were added to the method argument Stringlist. This is the case if the regex contains no capturing groups. Otherwise, each capturing group in the regex creates an item in the Stringlist, even if that item contains only the null string.
- It is indistinguishable whether an empty item in the output Stringlist represents a capturing group in the regex that was applied but matched no characters, or represents a capturing group that was not applied for some reason (for example, an earlier alternative made the match).
- A sub-expression that is a validly formed capturing group that is nested within a non-capturing sub-expression is still a capturing group.

In the following code fragment, the regex, which has two capturing groups, matches the string. Two items are added to the method Stringlist, only one of which is non-empty:

```

...
%sl = new
%regex = 'a(b)(c?)'
%inStr = 'ab'
%rc = %sl:RegexCapture (%inStr, %regex, Status=%st)

If not %rc then
    Print 'Status from RegexCapture is ' %st
Else
    Print %regex ' matches ' %inStr
End If
For %i from 1 to %sl:Count
    Print 'Captured item ' %i ' is: ' %sl:Item(%i)
End For
...

```

This code would print the following:

```
a(b)(c?) matches ab
Captured item 1 is: b
Captured item 2 is:
```

7.2 **RegexLocate and RegexLocateUp methods**

These functions use a regular expression (regex) to locate a string in the method Stringlist. They return the item number of the first item that the regex matches. Information about the regular expression matching rules observed is provided in [“Regex rules” on page 48](#).

Both functions proceed item-by-item through the Stringlist. `RegexLocate` starts by default from item 1, the first item in the Stringlist, or it starts from the item you specify. `RegexLocateUp` starts from the item `n` you specify and proceeds “upward” to item `n-1`, then `n-2`, and so on.

<pre>[%itemnum =] %s1:RegexLocate[Up] (regex, [%startitem], - [, StartCol=%s] - [, EndCol=%e] - [, Options=%str] - [, Status=%num])</pre>
--

RegexLocate and RegexLocateUp

`%itemnum`, if specified, contains the number of the `%s1` item matched by `regex`, or it is 0 if no items are matched.

`RegexLocate` and `RegexLocateUp` accept one required and five optional arguments, and they return a float.

- The first argument is a string that is interpreted as a regular expression and is applied to the `%s1` method Stringlist items to determine whether the regex finds a match. This longstring is a required argument.
- The second argument identifies the item number from which the regex matching is to begin. If this item is not matched, the method attempts to match the item with the next higher (if `RegexLocate`) or lower (if `RegexLocateUp`) item number, and so on.

This numeric value is an optional argument that defaults to the first list item (if `RegexLocate`) or the last item (if `RegexLocateUp`). Setting this argument to 0 is the same as setting it to 1.

- The `StartCol` argument (name required) is an optional number that specifies the starting column of the range of columns in which the matched string must be located.

If specified, `%s` must be greater than or equal to 1 and less than or equal to the `EndCol` argument value. If the argument is omitted, its default value is 1.

If you specify a `%s` value that is greater than the length of a particular `%s/` item, `regex` is matched against the empty string for that item.

- The `EndCol` argument (name required) is an optional number that specifies the ending column of the range of columns in which the matched string must be located.

If specified, `%e` must be greater than or equal to 1, greater than or equal to the `StartCol` argument value, and less than or equal to the lesser of 6124 or the length of `inStr`.

If the `EndCol` argument is omitted, its default value is 6124, but effectively it becomes the length of the `s/` item, for items less than 6124 bytes.

Note: If the `EndCol` argument is omitted and a `s/` item exceeds 6124 bytes, the run is cancelled.

- The `Options` argument (name required) is an optional string of options. The options are single letters, which may be specified in uppercase or lowercase, in any combination, and separated by blanks or not separated:

I Do case-insensitive matching between `%s/` items and `regex`.

S DOTALL mode. If this mode is **not** specified, a dot (`.`), also called a point, matches any single character except `X'0D'` (carriage return) and `X'25'` (linefeed). In DOTALL mode, a dot (`.`) also matches carriage return and linefeed characters.

Note: This is different from Perl, which does not consider a carriage return an end-of-line character. In Perl, a dot matches a carriage return even when not in DOTALL mode.

M Multi-line mode. If this mode is **not** specified, a caret (`^`) or a not sign (`¬`) — whichever key your keyboard program translates to `X'5F'` — matches only the position at the very start of the string, and dollar sign (`$`) matches only the position at the very end. (This documentation uses the caret.)

The caret and dollar sign are position-identifying characters known as “anchors,” which match the beginning and end, respectively, of a line or string. They do not match any text.

In M mode, a caret **also** matches the position immediately after any end-of-line indicator (carriage return, linefeed, carriage return linefeed), and a dollar sign **also** matches the position immediately before any end-of-line indicator.

- C** XML Schema mode. Do the match according to XML Schema rules: In a regex, no characters serve as anchors, and any regex is always assumed to be anchored at both ends. For example, the regex *X* in C mode is equivalent to `^(?:X)$` in non-C mode, where the `(?:` indicates a “non-capturing” group.

In addition, contrary to non-C mode:

- The usual anchoring-atoms, `^` and `$`, are treated as ordinary characters in a regex, and you may not escape them.
- The two-character sequence `(?` is not valid in a regex.
- The dot metacharacter `.` matches neither a carriage return character nor a linefeed.

You may want to use C mode for testing a regex for inclusion in a schema document (an XML document that constitutes an XML schema).

If you specify an invalid option, the request is cancelled.

- The Status argument (name required) is optional; if specified, it is set to an integer code. These values are possible:
 - n* The number of the `%s/` item that is first matched.
 - 0** No match: no items in `%s/` were matched by *regex*.
 - 1** The pattern in *regex* is invalid.
 - 2** The `%startitem` value is invalid (either a negative value or a value greater than the number of list items), or the method Stringlist is empty.

Note: If you omit this argument and a negative Status value is to be returned, the run is cancelled.

RegexLocate and RegexLocateUp are callable.

Notes:

- RegexLocate and RegexLocateUp require considerable user stack space. It is recommended that you increase the value of the *Model 204* LPDLST parameter to a minimum of 8000 (using, for example, `UTABLE LPDLST 8000`).
- The regex matching is limited to the first 6124 bytes of each item.
- The request is cancelled in any of these cases:
 - You fail to specify a required argument.
 - You specify an invalid argument option.

- You omit the Status argument and a situation occurs that would cause a negative Status value.
- You omit the EndCol argument and *inStr* exceeds 6124 bytes.

In the following code fragment, `RegexLocate` is applied to the method `Stringlist %s1` to find the first occurrence of the string `'CUST'`. If this is successful, the containing item is copied to a new `Stringlist`.

The matching is done case-insensitively, and the starting item is the beginning of the list, by default. Substituting `RegexLocateUp` for `RegexLocate` in the example would find the last occurrence of `'CUST'`.

```
...
%s1 = new
text to %s1
  %n = %list:findImageItem(%cust:ssn)
  if %n then
    %list:replaceImage(%n, 'CUST')
  else
    %list:addimage('CUST')
  end if
end text

%opt='i'
%regex = ''cust''
%itemnum = %s1:RegexLocate(%regex, Options=%opt, Status=%st)
  If (%itemnum EQ 0) then
    Print 'Status from RegexLocate[Up] is ' %st
  Else
    Print %regex ' matches item ' %itemnum
    Print 'Now for %s12 . . . '
    %s12 = new
    %i = %s12:CopyItems(%s1, %itemnum, '1')
    %i = %s12:Print
  End If
...

```

7.3 **RegexReplaceCorresponding method**

This function searches a given string for matches to one of multiple regular expressions contained in a list, and it replaces found matches with or according to one of multiple replacement strings contained in a list that corresponds to the regex list.

The regex list items are treated as mutually exclusive alternatives, and the function stops as soon as an item matches and the replacement is made. A “global” option is also available to continue searching and replacing within the given string using the matching regex item until no more matches are found.

RegexReplaceCorresponding uses the “rules” of regular expression matching (information about which is provided in [“Regex rules” on page 48](#)).

```
outStr = %regList:RegexReplaceCorresponding( -
                                     inStr, -
                                     replacementList -
                                     [, Options=%str] -
                                     [, Status=%num] )
```

RegexReplaceCorresponding Function

outStr is a longstring set to the value of *inStr* with each matched substring replaced by the value of the *replacementList* item that corresponds to the matching *%regList* item.

RegexReplaceCorresponding accepts two required and two optional arguments, and it returns a longstring.

- The first argument is the input string, to which the regular expressions in *%regList* are applied. This longstring is a required argument.
- The second argument is a Stringlist, each of whose items is a potential replacement string for the substring of *inStr* that is matched by the corresponding item of *%regList*.

This is a required argument. Regex subexpressions may be enclosed in parentheses, but the parentheses are treated as meaningless.

Except when the **A** option is specified (as described below for the Options argument), you can include `$0` markers in *replacementList* items as placeholders for the substring of *inStr* that the item matches.

`xxx$0` is an example of a valid replacement string, and `xxx` followed by the portion of *inStr* that gets matched (by the corresponding *%regList* item) comprises the replacement string.

Any character after the dollar sign other than a zero is an error. Multiple zeroes (as many as 9) are permitted.

You can also use the format `$m0`, where *m* is one of the following modifiers:

- U or u** Specifies that the matched substring should be uppercased when inserted.
- L or l** Indicates that the matched substring should be lowercased when inserted.

The only characters you can escape in a replacement string are dollar sign (`$`), backslash (`\`), and the digits `0` through `9`. So only these escapes are respected: `\\`, `\$`, and `\0` through `\9`. No other escapes are allowed in a replacement string, and an “unaccompanied” backslash (`\`) is an error.

To indicate that the first matched substring should go between the numbers 1 and 0 in the replacement string, use `1$0\0`.

- The Options argument (name required) is an optional string of options. The options are single letters, which may be specified in uppercase or lowercase, in any combination, and separated by blanks or not separated:

I Do case-insensitive matching between *inStr* and *%regList*.

S DOTALL mode. If this mode is **not** specified, a dot (`.`), also called a point, matches any single character except `X'0D'` (carriage return) and `X'25'` (linefeed). In DOTALL mode, a dot (`.`) also matches carriage return and linefeed characters.

Note: This is different from Perl, which does not consider a carriage return an end-of-line character. In Perl, a dot matches a carriage return even when not in DOTALL mode.

M Multi-line mode. If this mode is **not** specified, a caret (`^`) or a not sign (`¬`) — whichever key your keyboard program translates to `X'5F'` — matches only the position at the very start of the string, and dollar sign (`$`) matches only the position at the very end. (This documentation uses the caret.)

The caret and dollar sign are position-identifying characters known as “anchors,” which match the beginning and end, respectively, of a line or string. They do not match any text.

In M mode, a caret **also** matches the position immediately after any end-of-line indicator (carriage return, linefeed, carriage return linefeed), and a dollar sign **also** matches the position immediately before any end-of-line indicator.

C XML Schema mode. Do the match according to XML Schema rules: In a regex, no characters serve as anchors, and any regex is always assumed to

be anchored at both ends. For example, the regex `X` in C mode is equivalent to `^(?:X)$` in non-C mode, where the `(?:` indicates a “non-capturing” group.

In addition, contrary to non-C mode:

- The usual anchoring-atoms, `^` and `$`, are treated as ordinary characters in a regex, and you may not escape them (the escaping rules do not change).
- The two-character sequence `(?` is not valid in a regex.
- The dot metacharacter `.` matches neither a carriage return character nor a linefeed.

You may want to use C mode for testing a regex for inclusion in a schema document (an XML document that constitutes an XML schema).

- G** Global replace. If this mode is **not** specified, the *replacementList* string replaces the first matched substring only. In G mode, any additional substrings matched by the *regList* item also replaced by the *replacementList* string.
- A** As is. If this mode is specified, the *replacementList* item string is copied as is. No escapes are recognized; a `$0` combination is interpreted as a literal and **not** as a special marker; and so on.

If you specify an invalid option, the request is cancelled.

- The Status argument (name required) is optional; if specified, it is set to an integer code. These values are possible:
 - n* The number of replacements made. A value greater than 1 indicates option G was in effect.
 - 0** No match: *inStr* not matched by any *%regList* items.
 - 1** A regex in *%regList* is invalid.
 - 2** Syntax or other error: for example, the number of items in *%regList* does not equal the number in *replacementList*; or a *%regList* item exceeds 6124 bytes; or *%regList* is empty.
 - 5** An invalid string in a *replacementList* item. For example, an invalid escape sequence, or a `$` followed by any characters other than one or more (but no more than 9) zeroes.

Note: If you omit this argument and a negative Status value is to be returned, the run is cancelled.

Notes:

- `RegexReplaceCorresponding` requires considerable user stack space. It is recommended that you increase the value of the *Model 204* `LPDLST` parameter to a minimum of 8000 (using, for example, `UTABLE LPDLST 8000`).
- Items in `%regList` must **not** exceed 6124 bytes. However, the `inStr` value and items in `replacementList` may exceed 6124 bytes.

In the following code fragment, the second item in regex list `%regList` is the first to match the input string `inStr`. Since `%opt='g'` is specified, three replacements are made (using the corresponding, second, item in `%repList`):

```
...
%regList = new
text to %regList
  abcx
  a(bc?)
  abcd
end text

%repList = new
text to %repList
  &
  &&
  &&&
end text

%inStr = 'abc1abc2abcd'
%opt='g'
%outStr = %regList:RegexReplaceCorresponding (%inStr,
%repList, Options=%opt, Status=%st)

Print 'Status from ReplaceCorresponding is ' %st
Print 'OutputString: ' %outStr
...
```

The result would be:

```
Status from ReplaceCorresponding is 3
OutputString: &&1&&2&&d
```

7.4 **RegexSubset method**

This function returns a Stringlist that is a subset of the method Stringlist. The subset contains copies of all the items in the method Stringlist that are matched by a specified regex. Information about the regular expression matching rules observed is provided in “Regex rules” on page 48.

```
%subset = %sl:RegexSubset( regex, [ , StartCol=%s] -  
                                [ , EndCol=%e] -  
                                [ , Options=%str] -  
                                [ , Status=%num] )
```

RegexSubset Function

%subset is a Stringlist that contains the %sl items matched by *regex*.

RegexSubset accepts one required and four optional arguments, and it returns a Stringlist.

- The first argument is a string that is interpreted as a regular expression and is applied to the %sl method Stringlist items to determine whether the regex finds a match. This longstring is a required argument.
- The StartCol argument (name required) is an optional number that specifies the starting column of the range of columns in which the matched string must be located.

If specified, %s must be greater than or equal to 1 and less than or equal to the EndCol argument value. If the argument is omitted, its default value is 1.

If you specify a %s value that is greater than the length of a particular %sl item, *regex* is matched against the empty string for that item.

- The EndCol argument (name required) is an optional number that specifies the ending column of the range of columns in which the matched string must be located.

If specified, %e must be greater than or equal to 1, greater than or equal to the StartCol argument value, and less than or equal to the lesser of 6124 or the length of *inStr*.

If the EndCol argument is omitted, its default value is 6124, but effectively it becomes the length of the *sl* item, for items less than 6124 bytes.

Note: If the EndCol argument is omitted and a *sl* item exceeds 6124 bytes, the run is cancelled.

- The Options argument (name required) is an optional string of options. The options are single letters, which may be specified in uppercase or lowercase, in any combination, and separated by blanks or not separated:

- I** Do case-insensitive matching between *%s/* items and *regex*.
- S** DOTALL mode. If this mode is **not** specified, a dot (`.`), also called a point, matches any single character except `X'0D'` (carriage return) and `X'25'` (linefeed). In DOTALL mode, a dot (`.`) also matches carriage return and linefeed characters.

Note: This is different from Perl, which does not consider a carriage return an end-of-line character. In Perl, a dot matches a carriage return even when not in DOTALL mode.

- M** Multi-line mode. If this mode is **not** specified, a caret (`^`) or a not sign (`¬`) — whichever key your keyboard program translates to `X'5F'` — matches only the position at the very start of the string, and dollar sign (`$`) matches only the position at the very end. (This documentation uses the caret.)

The caret and dollar sign are position-identifying characters known as “anchors,” which match the beginning and end, respectively, of a line or string. They do not match any text.

In M mode, a caret **also** matches the position immediately after any end-of-line indicator (carriage return, linefeed, carriage return linefeed), and a dollar sign **also** matches the position immediately before any end-of-line indicator.

- C** XML Schema mode. Do the match according to XML Schema rules: In a regex, no characters serve as anchors, and any regex is always assumed to be anchored at both ends. For example, the regex *X* in C mode is equivalent to `^(?:X)$` in non-C mode, where the `(?:` indicates a “non-capturing” group.

In addition, contrary to non-C mode:

- The usual anchoring-atoms, `^` and `$`, are treated as ordinary characters in a regex, and you may not escape them.
- The two-character sequence `(?` is not valid in a regex.
- The dot metacharacter (`.`) matches neither a carriage return character nor a linefeed.

You may want to use C mode for testing a regex for inclusion in a schema document (an XML document that constitutes an XML schema).

If you specify an invalid option, the request is cancelled.

- The Status argument (name required) is optional; if specified, it is set to an integer code. These values are possible:

n The number of *%s/* items that are matched.

- 0 No match: no items in *%s/* were matched by *regex*.
- 1 The pattern in *regex* is invalid.

Note: If you omit this argument and a negative Status value is to be returned, the run is cancelled.

Notes:

- RegexSubset requires considerable user stack space. It is recommended that you increase the value of the *Model 204* LPDLST parameter to a minimum of 8000 (using, for example, `UTABLE LPDLST 8000`).
- The regex matching is limited to the first 6124 bytes of each item, but a matched item is copied in its entirety to the output subset.
- Prior to copying matched items to *%sub*, any preexisting contents of that Stringlist are deleted.

In the following code fragment, RegexSubset is applied to the method Stringlist *%s1* to find the *%s1* items that are matched by the regex `%\[a-z]*\`. The regex is designed to find items that contain shared methods whose class names contain only upper and lowercase letters.

```
...
%s1 = new
text to %s1
  b
  %doc is object xmlDoc
  %(daemon):getInputObject(%doc)
  %doc:selectSingleNode('/outer/inner'):addAttribute('foo','bar')
  %(daemon):returnObject(%doc)
end
end text

%regex = '%\[a-z]*\'
%opt='i'
%s12 = %s1:RegexSubset (%regex, Options=%opt, Status=%st)

If (%st EQ 0) then
  Print 'Status from RegexSubset is ' %st
Else
  Print %regex ' matches the following items:'
End If
For %i from 1 to %s12:Count
  Print 'Matching item ' %i ' is: ' %s12:Item(%i)
End For
...
```

This code would print the following:

```
%\[a-z]*\) matches the following items:  
Matching item 1 is: %(daemon):getInputObject(%doc)  
Matching item 2 is: %(daemon):returnObject(%doc)
```

7.5 FindUp method

The FindUp method is a variation of the existing Find method. Like Find, FindUp locates a Stringlist item that exactly matches a specified string. The difference between Find and FindUp is the direction of the search: Find searches from the starting point in ascending item number order, while FindUp searches in descending item number order.

```
%rc = %sl:FindUp(string, [startitem])
```

FindUp syntax

Syntax Terms

- %rc** A numeric variable that is set to the number of the first item in the Stringlist that matches the search string, or it is set to 0 if the string is not found.
- %sl** A Stringlist object.
- string** The string to be matched. This is a required argument.
- startitem** A number that indicates the item number at which the search is to begin. If this argument is not specified, searching begins at the last item in the Stringlist.

Janus SOAP Xml Classes

The following sections describe changes in the *Janus SOAP Xml** classes (the XML API) in this release.

8.1 DefaultURI function

This function gets the Uniform Resource Identifier (URI) associated with the default namespace declaration which is in scope at a node which is the head of an XPath result.

```
%uri = nr:DefaultURI([XPath])
```

DefaultURI syntax

Syntax Terms

- %uri* A string variable for the returned URI.
- nr* An XmlDocument or XmlNode, used as the context node for the *XPath* expression. If an XmlDocument, the Root node is the context node.
- XPath* An XPath expression that results in a nodelist, the head of which is the node to process. An optional argument, its default is a period (.), that is, the node referenced by the method object (*nr*).

In the following example:

```
Begin
%d Object XmlDocument Auto New
%n Object XmlNode
%n2 Object XmlNode
%n = %d:AddElement('p:top', , 'urn:top')
%n2 = %n:AddElement('inner', , 'urn:default')
%n:Print
Print 'URI of p:top default namespace = ' And %n:DefaultURI
Print 'URI of inner default namespace = ' And %n2:DefaultURI
End
```

the following lines are printed:

```
<p:top xmlns:p="urn:top">
  <inner xmlns="urn:default"/>
</p:top>
URI of p:top default namespace =
URI of inner default namespace = urn:default
```

The request cancellation conditions are the same as for the URI function.

8.2 PrefixURI function

This function gets the Uniform Resource Identifier (URI) associated with a prefix, or the default URI, in the context of a node which is the head of an XPath result.

`%uri = nr:PrefixURI(prefix, [XPath])`

PrefixURI syntax

Syntax Terms

- %uri* A string variable for the returned URI.
- nr* An XmlDocument or XmlNode, used as the context node for the *XPath* expression. If an XmlDocument, the Root node is the context node.
- prefix* A string which is the prefix to “lookup” in the XmlDocument. If this is the null string, then the %uri result is the default namespace at the node to process. Otherwise, the %uri result is the URI bound to *prefix* by a namespace declaration in scope at the node to process. In either case, the %uri result is null if there is no namespace associated with *prefix*.
- XPath* An XPath expression that results in a nodelist, the head of which is the node to process. An optional argument, its default is a period (.), that is, the node referenced by the method object (*nr*).

In the following example:

```
Begin
%d Object XmlDocument Auto New
%n Object XmlNode
%n = %d:AddElement('x', , 'urn:default')
%n:AddNamespace('foo', 'urn:foo')
%n:AddNamespace('bar', 'urn:bar')
%n:Print
Print 'Default namespace URI =' And %n:PrefixURI('')
Print 'URI bound to prefix "foo" =' And %n:PrefixURI('foo')
Print 'URI bound to prefix "bar" =' And %n:PrefixURI('bar')
End
```

the following lines are printed:

```
<x xmlns="urn:default" xmlns:foo="urn:foo"
xmlns:bar="urn:bar"/>
Default namespace URI = urn:default
URI bound to prefix "foo" = urn:foo
URI bound to prefix "bar" = urn:bar
```

The request cancellation conditions are the same as for the URI function.

8.3 InvalidCharacterPosition function

The following shared function in the XmlDocument class:

```
%pos = xmlDocQual:InvalidCharacterPosition(string)
```

checks the characters in the *string* argument and returns 0 if they are all valid as characters to be used as the value of an Element or Attribute node, or returns the position of the first invalid character otherwise.

Notes:

- The invalid characters of an Element or Attribute node are the “control characters”.
- Like any shared method, the method object can be an object of the class of the method (XmlDoc), null or not, or a class name qualifier:

```
* Example of first type, %myDoc may be null or non-null:
%pos = %myDoc:InvalidCharacterPosition(%str)
* Example of second type:
%pos = %(XmlDoc):InvalidCharacterPosition(%str)
```

- With the introduction of this method, the IsValidString function is being deprecated.

8.4 New DefaultURI argument to AddSubtree and InsertSubtreeBefore

The `DefaultURI` named argument may be specified to the `AddSubtree` or `InsertSubtreeBefore` functions, to control the URI of the default namespace of some of the Element nodes copied to the XmlDocument of the object method. Most of this section will describe the argument using `AddSubtree`, but it applies equally to `InsertSubtreeBefore`.

The `DefaultURI` argument takes a string value. It is optional; if it is **omitted**, the normal operation of `AddSubtree` occurs: every node added to the XmlDocument is added with the

same URI property as it has in the source XmlDocument. For example, in the following User Language fragment:

```
%out = %outDoc:AddElement('target', , 'urn:target')
%src = %inDoc:AddElement('sourceWrap', , 'urn:source')
%src = %src:AddElement('source')
%out:AddSubtree(%src)
%out:Print
```

The result is:

```
<target xmlns="urn:target">
  <source xmlns="urn:source">
</target>
```

This is because the default namespace URI in scope at the “source” element as a child of “sourceWrap” is `urn:source`, and that is retained in the normal AddSubtree copy operation.

When the DefaultURI argument is specified, its value is the URI that is used for all unprefixed elements in the source subtree that are not in the scope of a default namespace declaration that is also **within** the source subtree.

For example, in the following User Language fragment:

```
%out = %outDoc:AddElement('target', , 'urn:target')
%src = %inDoc:AddElement('sourceWrap', , 'urn:source')
%src = %src:AddElement('source')
%out:AddSubtree(%src, DefaultURI='urn:target')
%out:Print
```

The result is:

```
<target xmlns="urn:target">
  <source/>
</target>
```

This is because the declaration of the default namespace in scope at the “source” element is outside the source subtree, and so the URI used in the copy is the value of the DefaultURI argument.

The namespace URI of an unprefixed node is preserved if the declaration is within the copied subtree, whether the DefaultURI argument is specified or not.

For example, in the following User Language fragment:

```
%out = %outDoc:AddElement('target', , 'urn:target')
%src = %inDoc:AddElement('source', , 'urn:source')
%out:AddSubtree(%src, DefaultURI='urn:target')
%out:Print
```

The result is:

```
<target xmlns="urn:target">
  <source xmlns="urn:source"/>
</target>
```

You can use the DefaultURI **property** of the AddSubtree method object if you wish the copied subtree to “inherit” the default namespace of that node.

For example, in the following User Language fragment:

```
%out = %outDoc:AddElement('target', , 'urn:target')
%src = %inDoc:AddElement('sourceWrap', , 'urn:source')
%src = %src:AddElement('source')
%out:AddSubtree(%src, DefaultURI=%out:DefaultURI)
%out:Print
```

The result is:

```
<target xmlns="urn:target">
  <source/>
</target>
```

As mentioned, the DefaultURI argument is also available to the InsetSubtreeBefore function, and it operates in the same fashion, although in the example of “inheriting” the default namespace declaration, you need to obtain the DefaultURI property of the **parent** of the method object.

For example, in the following User Language fragment:

```
%top = %outDoc:AddElement('target', , 'urn:target')
%sib = %top:AddElement('sibling')
%src = %inDoc:AddElement('sourceWrap', , 'urn:source')
%src = %src:AddElement('source')

%sib:InsertSubtreeBefore(%src, DefaultURI=%top:DefaultURI)
%top:Print
```

The result is:

```
<target xmlns="urn:target">
  <source/>
  <sibling/>
</target>
```

The following method call would also produce the same result:

```
%sib:InsertSubtreeBefore(%src,
                          DefaultURI=%sib:DefaultURI('..'))
```

If the `DefaultURI` argument is specified, the value of the `Namespace` property of the method object of `AddSubtree` must be `On`.

8.4.1 Relaxation of Namespace property and AddSubtree/InsertSubtreeBefore

In previous versions of the *Sirius Mods*, the `AddSubtree` operation (and `InsertSubtreeBefore`) are both allowed between two different `XmlDocs` only if the `Namespace` property of both `XmlDocs` is the same.

This has been relaxed, so that the following cases are also allowed:

- The source `XmlDoc` has `Namespace=None` and the target `XmlDoc` has `Namespace=Ignore`.
- The source `XmlDoc` has `Namespace=None` and the target `XmlDoc` has `Namespace=On`, if the `DefaultURI` argument is specified.

8.5 New SortCanonical option for serialization methods

All serialization methods now accept the `SortCanonical` option, to indicate that namespace declarations (based on the prefix being declared) and attributes (based on the namespace URI followed by the local name) are serialized in sorted order. This can be useful, for instance, when using the `Serial` method to serialize a portion of an XML document for a signature.

The sort order for namespace declarations and attributes is from lowest to highest, and it uses the Unicode code ordering (for example, numbers are lower than letters).

Consider the following example fragment:

```
Text To %s1
<top p:abc="p" q:xyz="q"
  xmlns:p="urn:p" xmlns:q="http://q.com" xmlns="urn:default"
  name="t" id="z15"
/>
End Text
%d:LoadXml(%s1)
Print %d:Serial(, 'EBCDIC SortCanonical NoEmptyElt')
```

The result is (some newlines added for emphasis):

```
<top
  xmlns="urn:default"
  xmlns:p="urn:p" xmlns:q="http://q.com"
  id="z15" name="t"
  q:xyz="q" p:abc="p"
>
</top>
```

Notes:

- The reason for the name (`SortCanonical`) is that this ordering is part of the specification of the **Canonical XML Recommendation** (<http://www.w3.org/TR/xml-c14n>) and the **Exclusive Canonical XML Recommendation** (<http://www.w3.org/TR/xml-exc-c14n>), which constrain serializations to facilitate processing such as digital signatures.
- Notice the use of the `NoEmptyElt` option above. This produces an STag ("`<top ...>`") and an ETag ("`</top>`") for an empty element, rather than just the `EmptyElemTag` ("`<top ...>`").

The canonical XML recommendations state that an STag/ETag pair, rather than an `EmptyElemTag`, is the serialization for an empty element.

- Of the multiple serialization methods, the `SortCanonical` option is expected to be most often used with the `Serial` method. If you are using a different serialization method to obtain a canonical representation, you might want to use the `NoXmlDecl` option as well. This excludes the XML version declaration from the serialization, which the `Serial` method does by default.
- Some features of canonical XML are not included in version 6.9 of *Janus SOAP*; for example:
 - Attribute values must be framed by quotation marks (which is *usually* the case for *Janus SOAP*), never by apostrophes (which is *sometimes* the case for *Janus SOAP*).
 - Canonical handling of namespace declarations, particularly in the serialization of a subtree (which is usually the case for digital signatures) that may use declarations outside the subtree, is not performed as of version 6.9 of *Janus SOAP*. Support of the **Exclusive Canonical XML Recommendation** rules for namespace declarations will be provided in a future version of *Janus SOAP*.
 - There may be other details not supported; we have not completed an exhaustive study of the recommendations.
- Although `SortCanonical` uses the "Unicode" sort sequence, it should be noted that this is limited to Unicode values less than 256 (as of version 6.9 of *Janus SOAP*), so it is accomplished with an 8-byte EBCDIC to 8-byte Unicode table, which is (for all intents and purposes) merely an EBCDIC-to-ASCII translation.

- Although the example above uses the `EBCDIC` option to display the result, when digital signature processing is employed, you will probably want to omit the `EBCDIC` option, and use the default result of `Serial`, which is UTF-8.

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

9.1 Telnet Server Support

Janus Telnet support allows you to set up one or more telnet servers within a *Model 204* Online. JANUS commands are used to define and start a TCP/IP listening port for each Janus Telnet Server. The Janus Telnet Server only supports the tn3270 subset of the telnet protocol. That is, it will only serve clients that provide 3270 emulation via telnet. Janus Telnet servers can be accessed with telnet, or more specifically a tn3270 client of your choice, to gain 3270 access to *Model 204*. A tn3270 client will typically be running on an end-user's workstation.

Nowadays, almost all access to mainframe applications is via tn3270. Typically, clients connect to a tn3270 server provided by the TCP/IP stack for the operating system. While most of these stacks are now provided by IBM, there are still a few third-party stacks available, most of which provide their own telnet servers. For *Model 204* access, most tn3270 clients first connect to a service such as VTAM or CMS, which then provides access to *Model 204*.

Using the Janus Telnet Server has some advantages over using the telnet server provided by the TCP/IP stack:

- It eliminates the extra layer between TCP/IP and *Model 204* — either VTAM or CMS. This reduces complexity and, possibly, CPU overhead. For end-users, the extra step of connecting to *Model 204* after connecting to VTAM or logging on to CMS is eliminated — the user is connected directly to *Model 204*.
- It makes it much easier to determine the IP address of a 3270 user. When tn3270 access is via VTAM, *Model 204* only sees the VTAM LU of the terminal. Mapping this to an IP address can be difficult. Maintaining a fixed IP address to LU name mapping in TCP/IP makes this more feasible, but it is a maintenance headache.
- It reduces the number of thread types required in an Online. Since Telnet Server threads run in daemon threads, no special 3270 threads are required for 3270 access. In fact, since CMS and TSO both provide tn3270 clients, no special thread type is required for full screen access from CMS or TSO, either.
- The configuration of a Janus Telnet Server is simpler and more dynamic than the telnet server provided by the TCP/IP stack. Janus Telnet Server ports can be started and stopped at will with the JANUS START and JANUS STOP commands.

In addition, the simple JANUS DEFINE command structure makes it easy to set things like BINDADDR or TCPKEEPALIVE that are difficult, if not impossible, to set via the TCP/IP stack telnet server.

- For *Janus Network Security* customers, the Janus Telnet Server can be trivially configured to use SSL.
- The Janus tracing facilities may be used to trace 3270 datastreams. While obviously a somewhat esoteric capability, it can be quite useful on occasion. When tracing 3270 datastreams using JANUS TRACE, one should be aware that the 3270 datastreams are wrapped in telnet/tn3270 datastreams, so they will contain extra “stuff.” The telnet layer is described by RFC 854 (<http://www.faqs.org/rfcs/rfc854.html>) and tn3270 by RFC 1576 (<http://www.faqs.org/rfcs/rfc1576.html>).

There are a couple of disadvantages to using a Janus Telnet Server:

- The Janus Telnet Server currently provides no facility comparable to the VTAM transfer facility available for VTAM terminals.
- If display of data in SirScan is limited by IODEV type (that is, IODEV15), the display will pick up Janus Telnet Server connections, too, which might not be desirable. Of course, this is easy enough to correct by specifying `JAN:` followed by the port name, instead of using IODEV15 to limit SirScan display. For example, `JAN:WEB` would limit display to connections on port WEB.

9.2 Configuring a Janus Telnet Server

Janus Telnet Server is only available in *Sirius Mods* version 6.9.

Even though Janus Telnet Server runs full screen requests on daemon threads (iodev number set by the SDAEMDEV system parameter), *Model 204* typically considers an iodev's type to be either line-mode or full-screen. As such, the daemon threads running telnet requests dynamically switch to a different iodev number while there's a connection. Like SDAEMDEV, the iodev number to be used for telnet requests must be indicated in the user 0 CCAIN parameters.

The TNDEV parameter indicates the telnet iodev number. Since iodev 21 is currently unused, this is a good candidate, so adding a “TNDEV=21” to the user 0 parameters would enable telnet support.

If support for arbitrary geometry telnet clients is also desired, then “SIRTERM=1” should also be specified in the user 0 parameters, if SIRTERM is not already set there.

Once the prerequisite TNDEV parameter is set, all that is required for telnet support is to define a JANUS port that will provide telnet service. As for all other Janus port types,

this is done via the JANUS DEFINE command. The following command illustrates a typical JANUS DEFINE command for a telnet server port:

```
JANUS DEFINE TNSERV 777 TNSERV 10 WSFQUERY
```

A JANUS START for the port will then start the port and make direct telnet access to *Model 204* available.

As with any other port type, if a port number below 1024 is to be used, the port might need to be reserved for the Online in the TCP/IP configuration. The standard port number for telnet is 23. This port number will almost certainly be in-use on any mainframe on which *Model 204* will be running. However, on a multi-homed mainframe, it might be possible to use port 23 on a specific host IP address for Janus. The BINDADDR JANUS DEFINE parameter is required to provide service on a specific IP address:

```
JANUS DEFINE TNSERV 23 TNSERV 10 WSFQUERY BINDADDR  
198.242.244.13
```

9.3 New JANUS DEFINE parameters

The following new parameters have been added for the JANUS DEFINE command:

9.3.1 AUTOSYS subsys

This parameter sets the *Model 204* AUTOSYS parameter to the indicated value when a connection is received on a TNSERV port.

The AUTOSYS parameter is useful for providing functionality comparable to the equivalent user parameter for traditional *Model 204* full-screen threads. Since TNSERV connections run on daemon threads, and since it would not be common for AUTOSYS to be set for daemon threads, this parameter makes it possible to provide AUTOSYS functionality specifically for telnet connections.

The AUTOSYS parameter set via JANUS DEFINE remains intact over the lifetime of a telnet connection, even after a logoff and logon.

The AUTOSYS parameter must be followed by the name of a subsystem to be invoked after a login on the Telnet Server.

9.3.2 TCPKEEPALIVE

This parameter specifies that connections on the port should use TCP keep alives. TCP keepalives request that the TCP stack send periodic “keepalive” packets to the

communications partner to see if it is still there. The time interval between these packets cannot be set by Janus, but is set in the TCP/IP stack configuration. For example, with the IBM stacks, the keepalive interval is set in the TCPCONFIG INTERVAL parameter for BPX (IBM Communications Server) and the KEEPALIVEOPTIONS INTERVAL parameter for VM TCP/IP.

In some sense, the term keepalive is a misnomer — keepalive packets that are not responded to cause a connection to be closed, so keepalives will actually cause connections to be closed faster than they might be, otherwise.

TCPKEEPALIVE probably only makes sense for ports where connections are held open for long periods of time. TNSERV ports are the most likely candidate. For these ports, TCPKEEPALIVE might be useful for two reasons:

1. It can detect connections lost due to a client failure (say, a turned off workstation), reducing threads wasted for connections to lost clients.
2. It can reassure certain routers, especially those doing network address translation (NAT), that the connection is still active. Some routers will stop routing packets for connections on which no activity is seen for some period of time. Keepalives ensure that there is periodic activity on a connection, even if there is no user interaction.

Of course, for this to be successful, the TCP/IP stack's keepalive interval must be less than any applicable router's inactivity timeout. For this particular application, keepalives live up to their name.

Since the TCP/IP stack does the keepalives, the overhead in *Model 204* for setting this parameter is virtually zero.

9.3.3 WSFQUERY

This parameter indicates that Janus is to issue a “Write Structured Field Query” request to any telnet client that connects to it. This is useful if a connecting client is not using one of the standard screen geometries — Model 2 (24 by 80), Model 3 (32 by 80), Model 4 (43 by 80), or Model 5 (27 by 132). *Model 204* and many clients support more or less arbitrary geometries, but *Model 204* cannot detect a non-standard geometry on a telnet client without the WSFQUERY option.

If all connecting clients will be using the standard geometries, the tn3270 protocol correctly sends the model number, telling *Model 204* the screen geometry. However, even if there are no non-standard screen sizes connecting to a Janus TNSERV port, there is little harm in specifying the WSFQUERY parameter, other than the slight delay in the WSF query being sent and the response received.

If the 3270 clients at a site support arbitrary geometries, it might be well worth experimenting with them, as there is no particular reason that any of the standard geometries are ideally suited to a particular workstation. The *Model 204* command line

and editor, as well as many *UL/SPF* applications, take advantage of all the space available on non-standard geometry 3270 emulators. To enable support for arbitrary 3270 geometries, the SIRTERM system parameter's X'01' bit must be set.

9.4 TRUST parameter added to \$Sir_login

The fifth argument for \$sir_login can now contain the word TRUST. This indicates that the userid for which the login is done can do subsequent trusted logins by simply typing LOGON (with no userid) on the *Model 204* command line.

This feature is probably most useful in a development Online where end-users have access to the *Model 204* command line and might want to do a quick logon/logoff sequence to reset everything. Being able to simply type LOGON eliminates the need for re-entering a password, so it makes quick logon/logoffs much simpler than it would be otherwise. And since the user already logged on to the thread once, there is no security hole in doing a subsequent trusted login.

The TRUST parameter is only allowed for applications running on a TNSERV port.

CHAPTER 10 *Sirius Functions*

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

10.1 New wait time parameter for \$Bind

\$Bind now has a second parameter that indicates the number of milliseconds to wait for the resource name being bound, if it is not immediately available. This allows \$bind to be used to control access to resources that are held for relatively short periods of time, so shouldn't require user intervention to resolve the conflicts. For example, the following program binds the resource called 'BURNS', waiting up to 10 seconds (10,000 milliseconds) for the resource to become available.

```
b
%rc = $bind('BURNS', 10000)
end
```

10.2 \$RegexMatch: Whether string matches regex

This function determines whether a given pattern (regular expression, or “regex”) matches within a given string according to the “rules” of regular expression matching (information about the rules observed is provided in “[Regex rules](#)” on page 48).

```
[%rc =] $RegexMatch(inStr, regex, [options], [%status])
```

\$RegexMatch Function

%rc, if specified, is a number that is either 0 if the regular expression was invalid or no match was found, or the position of the character **after** the last character matched.

\$RegexMatch accepts two required and two optional arguments, and it returns a numeric value.

- The first argument is the input string, to which the regular expression *regex* is applied. This longstring is a required argument.

- The second argument is a string that is interpreted as a regular expression and is applied to the *inStr* argument to determine whether the regex matches *inStr*. This longstring is a required argument.
- The third argument is an optional string of options. The options are single letters, which may be specified in uppercase or lowercase, in any combination, and separated by blanks or not separated:

I Do case-insensitive matching between *string* and *regex*.

S DOTALL mode. If this mode is **not** specified, a dot (.), also called a point, matches any single character *except* X'0D' (carriage return) and X'25' (linefeed). In DOTALL mode, a dot (.) also matches carriage return and linefeed characters.

Note: This is different from Perl, which does not consider a carriage return an end-of-line character. In Perl, a dot matches a carriage return even when not in DOTALL mode.

M Multi-line mode. If this mode is **not** specified, a caret (^) or a not sign (¬) — whichever key your keyboard program translates to EBCDIC X'5F' — matches only the position at the very start of the string, and dollar sign (\$) matches only the position at the very end. (This documentation uses the caret.)

The caret and dollar sign are position-identifying characters known as “anchors,” which match the beginning and end, respectively, of a line or string. They do not match any text.

In M mode, a caret **also** matches the position immediately after any end-of-line indicator (carriage return, linefeed, carriage return linefeed), and a dollar sign **also** matches the position immediately before any end-of-line indicator.

C XML Schema mode. Do the match according to XML Schema rules: In a regex, no characters serve as anchors, and any regex is always assumed to be anchored at both ends. For example, the regex *X* in C mode is equivalent to *^(?:X)\$* in non-C mode, where the *(?:* indicates a “non-capturing” group.

In addition, contrary to non-C mode:

- The usual anchoring-atoms, ^ and \$, are treated as ordinary characters in a regex, and you may not escape them.
- The two-character sequence (? is not valid in a regex.
- The dot metacharacter (.) matches neither a carriage return character nor a linefeed.

You may want to use this mode for testing a regex for inclusion in a schema document, an XML document that constitutes an XML schema.

- The fourth argument is optional; if specified, it is set to an integer status value. If you omit this argument and a negative *%status* value is to be returned, the run is cancelled. These values are possible:
 - 1 A successful match was obtained.
 - 0 No match: *inStr* was not matched by *regex*.
 - 1 The pattern in *regex* is invalid.

\$RegexMatch is callable.

Notes:

- \$RegexMatch requires considerable user stack space. It is recommended that you increase the value of the *Model 204* LPDLST parameter to a minimum of 8000 (using, for example, `UTABLE LPDLST 8000`).
- If *%rc* is zero, either *regex* did not match *inStr*, or there was an error in the regex. The *%status* argument returns additional information. If it is negative, it indicates an error. If it is zero, it indicates there was no error, but the regex did not match.
- If you fail to specify a required argument, or if you specify an invalid argument option, the request is cancelled.
- \$RegexMatch has related Stringlist class methods:
 - `RegexCapture` (“[RegexCapture method](#)” on page 15) has the same functionality as \$RegexMatch, and it also saves (“captures”) the matching values of any regex sub-expressions you enclose in parentheses.
 - `RegexLocate` and `RegexLocateUp` (“[RegexLocate and RegexLocateUp methods](#)” on page 18) use regex matching to find a Stringlist item.
 - `RegexSubset` (“[RegexSubset method](#)” on page 26) uses a regex to copy matching Stringlist items to a new Stringlist.

`$RegexReplace` (“[\\$RegexReplace: Replace matching strings](#)” on page 51) uses a regex to search a string and replace the characters that match the regex.

The following example tests whether the regex `*bc?[5-8]` matches the string `a*b6`. If the return code is 0 (no match), the status variable is checked for more information.

```
Begin
  %rc float
  %regex Longstring
  %String Longstring
  %Options string len 10
  %status float

  %Options = ''
  %regex = '\*bc?[5-8]'
  %String = 'a*b6'

  %rc = $RegexMatch (%String, %regex, %Options, %status)
  If (%rc EQ 0) then
    Print 'Status from $RegexMatch is ' %status
  Else
    Print %regex ' matches ' %String
  End If
End
```

The regex matches the input string; the example result is:

```
\*bc?[5-8] matches a*b6
```

This regex demonstrates the following:

- To match a string, a regex pattern must merely “fit” a substring of the string.
- Metacharacters, in this case star (*), must be escaped.
- An optional character (c?) may fail to find a match, but this does not prevent the success of the overall match.
- The character class range ([5-8]) matches the 6 in the input string.

10.2.1 Regex rules

When a regular expression is said to “match a string,” what is meant is that a substring of characters within the string fit (are matched by) the pattern specified by the regex. The “rules” observed by Sirius for regex formation and matching are primarily those followed by the Perl programming language (as described, for example, in *Programming Perl*, by Larry Wall et al, published by O'Reilly Media, Inc.; 3rd edition, July 14, 2000). An additional reference is *Mastering Regular Expressions*, by Jeffrey E. F. Friedl, published by O'Reilly Media, Inc. (2nd edition, July 15, 2002).

Exceptions to the Perl “standard” include those specifically noted in the *options* argument in the `$RegexMatch` function description above, as well as the following:

- In **character classes** (which “match any character in the square brackets”), the only ranges allowed are subsets of uppercase letters, lowercase letters, or digits. For example, `[A-z]` is illegal; `[A-Za-z]` **is** legal; `[a-9]` is **not** legal.

Because of the gaps in the EBCDIC encoding, you can specify `[A-Z]`, but internally that is converted to `[A-IJ-RS-Z]`; and similarly for `[a-z]`.

- The only **escape sequences** allowed in a Sirius regex are those for metacharacters and those that are “shorthands” for special characters or character classes, as specified below.

These metacharacter escapes are allowed anywhere except within replacement strings:

<code>\.</code>	Dot
<code>\[</code>	Left square bracket
<code>\]</code>	Right square bracket
<code>\(</code>	Left, or opening, parenthesis
<code>\)</code>	Right, or closing, parenthesis
<code>*</code>	Star, or asterisk
<code>\-</code>	Hyphen
<code>\{</code>	Left curly bracket
<code>\}</code>	Right curly bracket
<code>\ </code>	Vertical bar
<code>\\</code>	Backslash
<code>\+</code>	Plus sign
<code>\?</code>	Question mark
<code>\\$</code>	Dollar sign (not allowed in XML Schema mode of regex functions/methods)
<code>\^</code>	Caret, or circumflex (not allowed in XML Schema mode)

These “shorthand” escape sequences are allowed *outside of* character classes and replacement strings:

<code>\r</code>	Carriage return (X'0D')
<code>\n</code>	Linefeed (X'25')
<code>\t</code>	Horizontal tab (X'05')
<code>\s</code>	Whitespace character; equivalent to <code>[\r\n\t]</code>
<code>\S</code>	Non-whitespace; equivalent to <code>[^\r\n\t]</code>
<code>\d</code>	Digit; equivalent to <code>[0-9]</code>
<code>\D</code>	Non-digit; equivalent to <code>[^0-9]</code>
<code>\i</code>	Legal start-of-name character; equivalent to <code>[_:A-Za-z]</code>
<code>\I</code>	Non-legal start-of-name character; equivalent to <code>[^_:A-Za-z]</code>
<code>\c</code>	Legal name character; equivalent to <code>[\-_:.A-Za-z0-9]</code>
<code>\C</code>	Non-legal name character; equivalent to <code>[^\-_:.A-Za-z0-9]</code>

Only these escapes are allowed within replacement strings:

<code>\\$</code>	Dollar sign
<code>\\</code>	Backslash

`\0 \1 ... \9` Digits 0 through 9

- **Look-behind** sub-expressions in a regex are **not** supported. The only supported parenthesized sub-expression sequences that begin with a question mark are the following:

`(?:` Denotes a non-capturing group

`(?=` Denotes a positive look-ahead

`(?!` Denotes a negative look-ahead

- **Unicode** is not supported.
- Only “greedy” matching is supported. That is, if there is more than one plausible match, **greedy matching** selects the longest one.

Greediness is introduced by the quantifiers (like `*`, `+`, `?` etc.), which govern how many input string characters the preceding regex item may try to match. The contrasting “lazy” quantifiers (like `*?`, `+`, `??` etc.) are **not** supported.

The quantifiers have rules for the minimum and maximum number of characters they potentially can match. In cases where both the minimum and maximum are possible, a choice between them must be made. Greediness refers to choosing the maximum.

For example, the regex `(a+)ab` matches the input string `aaaab`. In Sirius methods and \$functions, the parenthesized sub-expression, `a+`, matches `aaa`, although its set of plausible matches also includes strings of one and two `a`'s.

Understanding this greediness becomes more important when the string that `a?` matches is being replaced by another string. For example, to convert some HTML code to its stricter relative, XHTML, you want to change `` to ``.

You can use the following regex, which looks for `img` tags that have attributes — and use the `$RegexReplace` function (“[\\$RegexReplace](#)” on page 51):

```
(<img .*>)
```

However, the same regex applied to the string `<body></body>` does not insert the end tag `` after the first closing angle bracket (`>`) after `"24"` as you want. Instead, the matched string greedily extends to the second closing angle bracket, and the tag `` is positioned at the end:

```
</body></img>
```

One remedy for this situation is to use the following regex, which employs a negated character class to match non-closing-bracket characters. This regex does not extend beyond the first closing angle bracket in the input string:

```
(<img [^>]*>)
```

Although not exceptions to Perl, these features of Sirius regex support are worth emphasis:

- **Alternatives** (indicated by `|`) are evaluated from left to right, and evaluation is “short-circuited” (that is, it stops as soon as it finds a match).
- **Empty expressions**, for example, empty alternatives, are supported. The following regex matches `A9`, `B9`, and `9`, capturing respectively `A`, `B`, and the null string:

```
(A|B|)9
```

An empty alternative (like the `|`, above, that is followed only by the closing parenthesis) is always `True`.

Note: Sirius and Perl make a special case of a regex that has an empty alternative on the left. In such a regex, the empty alternative on the left is evaluated as the last alternative instead of as the first. For example, the regex `(|A|B)9` also matches `A9`, `B9`, and `9`. However, because the evaluation of the empty alternative is implicitly postponed until the other alternatives are tried, this regex captures, respectively, `A`, `B`, and the null string.

- Positive and negative **look-aheads** are supported. Their specifications begin with `(?=` and `(?!`, respectively.

10.3 **\$RegexReplace: Replace matching strings**

This function searches a given string for matches of a regular expression, and it replaces found matches with or according to a specified replacement string. The function stops after the first match and replace, or it can continue searching and replacing until no more matches are found.

Matches are obtained according to the “rules” of regular expression matching (information about the rules observed is provided in [“Regex rules” on page 48](#)).

```
outStr = $RegexReplace(inStr, regex, replacement, -  
                      [options], [%status])
```

\$RegexReplace Function

outStr is a longstring set to the value of *inStr* with each matched substring replaced by the value of *replacement*.

`$RegexReplace` accepts three required and two optional arguments, and it returns a longstring.

- The first argument is the input string, to which the regular expression *regex* is applied. This longstring is a required argument.
- The second argument is a string that is interpreted as a regular expression and that is applied to the *inStr* argument to find the one or more *inStr* substrings matched by *regex*. This longstring is a required argument.
- The third argument is the string that replaces the substrings of *inStr* that *regex* matches. This longstring is a required argument.

Except when the **A** option is specified (as described below for the fourth argument), you can include markers in *replacement* to indicate where to insert corresponding captured strings. As in Perl, these markers are in the form `$n`, where *n* is the number of the capture group, and 1 is the number of the first capture group. *n* must not be 0 or contain more than 9 digits.

`xxx$1` is an example of a valid replacement string, and `$0yyy` is an example of a **non**-valid one.

Or you can use the format `$mn`, where *m* is one of the following modifiers:

- U or u** Specifies that the specified captured string should be uppercased when inserted.
- L or l** Indicates that the captured string should be lowercased when inserted.

The only characters you can escape in a replacement string are dollar sign (`$`), backslash (`\`), and the digits `0` through `9`. So only these escapes are respected: `\\`, `\$`, and `\0` through `\9`. No other escapes are allowed in a replacement string, and an “unaccompanied” backslash (`\`) is an error.

The scan for the number that accompanies the meta-`$` stops at the first non-numeric, so to indicate that the first captured string should go between the numbers 1 and 2 in the replacement string, use `1$1\2`.

- The fourth argument is an optional string of options. The options are single letters, which may be specified in uppercase or lowercase, in any combination, and separated by blanks or not separated:

I Do case-insensitive matching between *string* and *regex*.

S DOTALL mode. If this mode is **not** specified, a dot (`.`), also called a point, matches any single character except `X'0D'` (carriage return) and `X'25'` (linefeed). In DOTALL mode, a dot (`.`) also matches carriage return and linefeed characters.

Note: This is different from Perl, which does not consider a carriage return an end-of-line character. In Perl, a dot matches a carriage return even when not in DOTALL mode.

M Multi-line mode. If this mode is **not** specified, a caret (^) or a not sign (¬) — whichever key your keyboard program translates to X'5F' — matches only the position at the very start of the string, and dollar sign (\$) matches only the position at the very end. (This documentation uses the caret.)

The caret and dollar sign are position-identifying characters known as “anchors,” which match the beginning and end, respectively, of a line or string. They do not match any text.

In M mode, a caret **also** matches the position immediately after any end-of-line indicator (carriage return, linefeed, carriage return linefeed), and a dollar sign **also** matches the position immediately before any end-of-line indicator.

C XML Schema mode. Do the match according to XML Schema rules: In a regex, no characters serve as anchors, and any regex is always assumed to be anchored at both ends. For example, the regex *X* in C mode is equivalent to `^(?:X)$` in non-C mode, where the `(?:` indicates a “non-capturing” group.

In addition, contrary to non-C mode:

- The usual anchoring-atoms, ^ and \$, are treated as ordinary characters in a regex, and you may not escape them.
- The two-character sequence (? is not valid in a regex.
- The dot metacharacter (.) matches neither a carriage return character nor a linefeed.

You may want to use this mode for testing a regex for inclusion in a schema document, an XML document that constitutes an XML schema.

G Global replace. If this mode is **not** specified, the *replacement* string replaces the first matched substring only. In G mode, every occurrence of the match is replaced.

A As is. If this mode is specified, the *replacement* string is copied as is. No escapes are recognized; a \$*n* combination is interpreted as a literal and **not** as a special marker; and so on.

- The fifth argument is optional; if specified, it is set to an integer error code. These values are possible:

n The number of replacements made.

0 No match: *inStr* was not matched by *regex*.

-1 The pattern in *regex* is invalid.

- 5 An invalid *replacement* string. For example, an invalid escape sequence, or a \$ followed by a non-number, by a 0 or by no digits, or by more than 9 digits.

Note: If you omit this argument and a negative Status value is to be returned, the run is cancelled.

Notes:

- \$RegexReplace requires considerable user stack space. It is recommended that you increase the value of the *Model 204* LPDLST parameter to a minimum of 8000 (using, for example, `UTABLE LPDLST 8000`).
- Any parenthesized sub-expression in the regex is a capturing group unless its opening or closing parenthesis is escaped (preceded) by a backslash (\), or unless the opening parenthesis is followed by a question mark (?). A capturing group that is nested within a non-capturing sub-expression is still a capturing group.
- If a capturing group makes no matches (is positional, for example), or if there was no *n*th capture group corresponding to the \$*n* marker in a replacement string, the value of \$*n* used in the replacement string is the empty string.
- In Perl, \$*n* markers (\$1, for example) enclosed in single quotes are treated as literals instead of as “that which was captured by the first capturing parentheses.” \$RegexReplace uses the `A` option of the `Option` argument for this purpose.
- If this is the *regex*:

`9([A-Z])*9`

And this is the input string:

`xxx9ABCDEF9yyy`

\$RegexReplace replaces “ABCDEF”, the concatenation of the multiple matches.

In the following example, the regex (5.) is applied repeatedly (global option) to the string 5A5B5C5D5E to replace the uppercase letters with their lowercase counterparts. The \$L1 *%replacement* value makes the replacement string equal to whatever is matched by the capturing group, (5.), in the regex (the L causes the lowercase versions of the captured letters to be used).

```
Begin
  %regex Longstring
  %inStr Longstring
  %replacement Longstring
  %outStr Longstring
  %opt string len 10
  %status float

  %inStr='5A5B5C5D5E'
  %regex='(5.)'
  %replacement='$L1'
  %opt='g'
  %outStr = $RegexReplace (%inStr, %regex, %replacement, -
                          %opt, %status)
  Print '%RegexReplace: status = ' %status
  Print 'OutputString: ' %outStr
End
```

The example result is:

```
%RegexReplace: status = 5
OutputString: 5a5b5c5d5e
```

10.4 Updated mathematical functions

Sirius Mods version 6.9 includes high-performance, high-precision versions of most of the IBM mathematical functions available via User Language.

The following functions are added to the *Sirius Functions*. With the exception of \$MIN and \$MAX, they are functionally equivalent to the versions documented in the *Model 204 User Language Manual*. The *Sirius Functions* version of \$MIN and \$MAX accept as many as eight parameters instead of the the maximum of five supported in the CCA version.

\$ABS(x)	Absolute value
\$ARCCOS(x)	Inverse cosine
\$ARCSIN(x)	Inverse sine
\$ARCTAN(x)	Arctangent
\$ARCTAN2(x,y)	Arctangent of x/y
\$COS(x)	Cosine of x radians
\$COSH(x)	Hyperbolic cosine

\$COTAN(x)	Cotangent of x radians
\$ERF(x)	Error function
\$ERFC(x)	Complement error function
\$GAMMA(x)	Gamma function
\$IXPI(x,y)	Integer base raised to integer exponent
\$LGAMMA(x)	Log gamma function
\$MAX(x1,x2,x3,x4,x5,x6,x7,x8)	Maximum of specified arguments
\$MIN(x1,x2,x3,x4,x5,x6,x7,x8)	Minimum of specified arguments
\$PI	Value of pi
\$RXPI(x,y)	Real base raised to integer exponent
\$RXPR(x,y)	Real base raised to real exponent
\$SIN(x)	Sine of x radians
\$SINH(x)	Hyperbolic sine of x radians
\$SQRT(x)	Square root of positive argument
\$TAN(x)	Tangent of x radians
\$TANH(x)	Hyperbolic tangent of x radians

CHAPTER 11 *SirTune*

As of *Sirius Mods* 6.9, *SirTune* data collector functionality is integrated into the *Sirius Mods* product. You no longer need to install *SirTune* as a separate product. The *SirTune* Report Generator (SIRTUNER), however, is still installed separately. See the *SirTune Reference Manual* for installation details.

11.1 Compatibility

Existing configuration information does not require any modification. SIRTUNEI configuration statements and the SIRTUNED dataset may be used as-is with *Sirius Mods* 6.9

The SIRTUNEO dataset is obsolete under *Sirius Mods* 6.9. It is ignored if it is allocated. *SirTune* CPU authorization and patch usage messages are now written to the *Model 204* journal with the normal *Sirius Mods* initialization messages.

11.2 Changes

- Under previous releases of *SirTune*, the SIRTUNE module was invoked instead of the *Model 204* load module during initialization. Under z/OS or OS/390, you did this as follows:

```
//ONLINE EXEC PGM=SIRTUNE,...
```

This format is no longer necessary — in fact, it will prevent the integrated *SirTune* program from running. *SirTune* invocation no longer requires JCL modification. The EXEC statement in your JCL should directly invoke the *Model 204* load module:

```
//ONLINE EXEC PGM=ONLINE,...
```

- If your site is authorized for *SirTune*, the *SirTune* data collector gets initialized by default, with these exceptions:
 - Setting the new SIRTUNE parameter (described on the following page) to 0 disables initialization of the integrated *SirTune* product.
 - Under z/OS or OS/390, if no SIRTUNED DD statement is present during initialization, *SirTune* does not get initialized.

- It is possible to run an earlier version of *SirTune* with *Sirius Mods* 6.9, but this configuration is not recommended, since older versions of *SirTune* will no longer be upgraded to work with new versions of *Sirius Mods* or *Model 204*
- The SIRTUNEI data set does not require modification to run with the integrated *SirTune* module. However, the PGM statement is ignored, since *SirTune* no longer loads the *Model 204* ONLINE or BATCH204 load module.
- The SIRTUNE parameter statement is added to allow control over whether the *SirTune* data collector is initialized at the start of a *Model 204* run.

The SIRTUNE parameter, which is only allowed as a parameter in the EXEC JCL statement or CMS EXEC statement, can be set to 0 or 1. Setting it to 0 disables initialization of the integrated *SirTune* product for a particular run. Setting it to 1, the default, enables it.

An example of the parameter's specification follows:

```
//ONLINE EXEC PGM=ONLINE, PARM='SIRTUNE=0'
```

For CMS, the following format (or equivalent) is necessary in the ONLINE or BATCH204 invocation EXEC:

```
M204CMS M204ONLN ( SIRTUNE 0 ...
```

CHAPTER 12 *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 (6.8).

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).

12.1 Backwards incompatibilities

Backwards incompatibilities are described per product in the following sections.

12.1.1 Janus SOAP XML processing

The following backwards compatibility issues have been introduced in the *Janus SOAP* Xml* methods.

12.1.1.1 XPath with “!=” and Element with more than 1 child

The processing of the **not equals** comparison in XPath predicates was, in some cases, incorrect. If the comparison was done to an Element with more than one child and in fact the value of the Element was different from the literal being compared, the Element was not included in the result, although it should be. For example, given the following XML document:

```
<top>
  <emphasis>Crazy</emphasis> bug
</top>
```

The result of the following statement:

```
Print %d:Exists('*[. != "OK"]'):ToString
```

should be `True`; however, prior to this bug fix, it was `False`.

This change was not delivered with the initial version 6.9 *Sirius Mods*, but rather it was delivered as maintenance (ZAP69D0). Although Sirius does not usually update Release Notes retrospectively with changes delivered by a zap, since there are multiple zaps changing XPath processing, we have included these XPath changes compatibility notes. This section will also be inserted into the version 7.0 *Sirius Mods* Release Notes.

12.1.1.2 Fixed XPath “following” and “xx-sibling” axes

The change described in this section was not delivered with the initial version 6.9 *Sirius Mods*, but rather it was delivered as maintenance (ZAP69B9). Although Sirius does not usually update Release Notes retrospectively with changes delivered by a zap, for the sake of completeness this section contains information copied from the ZAP69B9 description, since some XPath changes are discussed (in “[Corrected XPath results with unusual axes](#)”). This section will also be inserted into the version 7.0 *Sirius Mods* Release Notes.

These behaviors, which did not conform to the XPath standard, have been fixed:

- Previously, using XPath's `following` axis (for example, `following::*`) resulted in the incorrect selection of extra nodes, in particular, Attribute and descendant nodes of the step's context node(s). Selecting these nodes is explicitly excluded by the XPath standard.
- Previously, using the `following-sibling` and `preceding-sibling` axes with an Attribute context node (for example, '@x/following-sibling::*' or '@x/preceding-sibling::*'), which by the standard should *not* select any nodes, might incorrectly select nodes.

In version 7.0 of the *Sirius Mods*, the ability to navigate from one Attribute node to its sibling (and in turn, the rest of the sibling Attribute nodes), will be provided by the Next and Previous methods (which can be used from nodes other than Attribute nodes, as well).

For information about the performance implications of the axes discussed above, see “[Performance considerations: Document order, certain axes](#)” on page 62

12.1.1.3 Corrected XPath results with unusual axes

Prior to this correction, errors might occur in some cases using these axes:

```
following
descendant
descendant-or-self (which can be obtained with //)
ancestor
ancestor-or-self
preceding-sibling
```

and in some unusual cases using these axes:

`parent` (which can be obtained with `..`)
`following-sibling`

The errors that can occur are:

- Return of an incorrect node, or return of a node other than the first node in document order

Occurs with methods that return the first selected node, such as `Value` and `SelectSingleNode`

- Incorrect, duplicate, and/or out-of-order result nodes

Occurs with methods that return an `XmlNodeList` (`SelectNodes` and `UnionSelected`)

For example, given the following document:

```
<t>
  <a a="aa" n="0">
    <a a="ab" n="1">
      <b n="1"/>
      <b x="1"/>
    </a>
  <b n="2"/>
  <c n="2"/>
</a>
</t>
```

The result of `Serial('//a[2]')` before the code fix was:

```
<a a="ab" n="1"><b n="1"/><b x="1"/></a>
```

This result element is the first child of its parent; it is not the second "a" child as specified in the XPath expression. In fact, the request in this example should be cancelled, because the XPath result is empty — there is no second "a" child of any element in the document.

With the same document, the result of `Serial('//self::node()[@a]/b')` before the code fix was:

```
<b n="2"/>
```

However, this is not the first element satisfying the XPath expression; the correct result is the element `<b n="1">`.

The result of `SelectNodes('t/a/a/@*/following:*')` with the same document before the code fix was:

```
<b n="1" />
<b x="1" />
<b n="2" />
<c n="2" />
<b n="1" />
<b x="1" />
<b n="2" />
<c n="2" />
```

Notice that the last four items in the `XmlNodeList` (duplicates of the first four) should not be present.

For information about the the performance implications of using the axes discussed above, see [“Performance considerations: Document order, certain axes”](#).

This change was not delivered with the initial version 6.9 *Sirius Mods*, but rather it was delivered as maintenance (ZAP69B9). Although Sirius does not usually update Release Notes retrospectively with changes delivered by a zap, this is an exception, in case you want to see this more expansive description when you apply the zap. This section will also be inserted into the version 7.0 *Sirius Mods* Release Notes.

12.1.1.4 Performance considerations: Document order, certain axes

This section of the Release Notes is a copy of the revised section (titled “A.5. Order of nodes: performance,” as of November 28, 2006) in the *Janus SOAP Reference Manual*. This section will also be inserted into the version 7.0 *Sirius Mods* Release Notes.

The changes in the revised section from the previous version of the manual are:

1. In addition to the axes and axis combinations specified in the preceding appendix subsections, extra processing can occur for these axes and circumstances:

<i>Axis</i>	<i>Circumstances</i>
ancestor, ancestor-or-self	Regardless of anything else in the XPath expression
attribute	If it is subsequent to a <code>parent</code> axis that is not the first step in the expression
following	If it is not the first step in the expression

2. There is a special case that is an exception to the general rule for the `descendant-or-self` axis followed by `child`. An example is `//chapter`.

Revised version of “A.5 Order of nodes: performance”

This section discusses the performance implications of evaluating certain XPath expressions. The expressions of concern have a common characteristic — they are not **simple XPath expressions**.

Simple XPath expressions, which have **no special performance considerations**, are any of these:

- One or more steps containing only `child`, `attribute`, or `self` axes.
- A `parent` axis used in the first step, after which may be one or more steps containing only `child`, `attribute`, or `self` axes.
- A `following-sibling` axis *used alone*.

The rest of this section considers XPath expressions that are not simple and therefore might have negative performance implications. If your use of XPath is confined to the simple expressions defined above, the following discussion is not your concern.

The XPath expression arguments of methods like `SelectNodes` and `UnionSelected` in the `XmlNodeList` class designate a set of nodes. In addition to these “set-valued” methods, XPath expressions can be used in many *Janus SOAP* XML document methods (`SelectSingleNode`, `Value`, `DeleteSubtree`, `QName`, and more) to operate on a single node that satisfies the expression. For the simple XPath expressions (described above), the “single node” methods scan fewer than or as many nodes before determining the desired node, thus give better performance than the set-valued methods.

The single-node XPath selection by *Janus SOAP* returns the first node in document order, but with non-simple XPath expressions, this is not the same as the first node found by the XPath **internal selection algorithm**, which may visit nodes in a different order. In those cases, *Janus SOAP* examines an entire subtree to determine the first node, in document order, that the XPath expression selects.

In other words, given an XPath expression *expr* that uses any of the axis cases described below in “[The extra-processing expressions](#)” on page 64, and given any single-node selection method *XMeth*, this expression:

```
%obj:XMeth(expr)
```

scans as many nodes as:

```
%obj:SelectNodes(expr):Item(1):XMeth
```

For all other XPath expressions, the number of nodes scanned by the first of these two approaches may be significantly lower, because the first node internally selected will also be the first node in document order.

For example, consider 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 finds the match `x=2`, then it continues searching the entire document for all matches, to ensure that the match which is lowest in document order (`x=1`) is the result.

The performance implications of the expressions that involve extra processing apply to the set-valued methods as well. The set methods must produce their results in document order, but the nodes selected during XPath evaluation may be selected in an order (due to the selection algorithm) that differs from document order.

The extra-processing expressions

Extra processing can occur in the following cases:

1. The presence of the `preceding-sibling` axis
2. The presence of the `ancestor` axis
3. The presence of the `ancestor-or-self` axis
4. The presence of the `following` axis, if it is not the first step in the expression
5. The presence of any of these axis-combinations:

One of the following axes:

- `descendant`
- `descendant-or-self`
- `following`
- `parent`, if it is not the first step in the expression

followed, in a subsequent step, by any of these axes:

- `parent`
- `child`
- `following-sibling`
- `descendant`

- `descendant-or-self`

The

- `parent` axis, if it is not the first step in the expression,

followed, in a subsequent step, by the

- `attribute` axis.

In addition to the cost of the actual XPath search performed with the above expressions, they can incur an additional cost for XPath evaluation. If the document has been modified in such a way that the internal order of the nodes cannot be guaranteed to be the same as document order (this will always happen with any of the XML `Insert..Before` methods, and usually will happen with any of the `Add..` methods), then the entire document (not only the subtree being searched) must be scanned so that the order is adjusted. This does not involve any internal movement of the nodes, but does require a full scan.

Note:

1. One important exception to the above rules is the `descendant-or-self::node()` **step** followed immediately by the `child` axis without any predicate. An example of the usual way to specify this is:

```
//chapter
```

In this case, the internal node selection algorithm operates in document order, and no extra processing is incurred.

Even with this special case, it is better to avoid the `descendant-or-self` step (specified explicitly or by using `//`) if your document structure lends itself to explicitly specifying the “intermediate” elements (and, even better, their names) that should be matched.

2. The considerations described in this section only apply to the “outer” XPath expression; they do not apply to any expression within a predicate. Although it is still better, for the sake of efficiency, to prune the search by explicitly specifying “intermediate” elements rather than using `//`, there is no efficiency concern due to the internal order of node selection with an XPath predicate such as the following:

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

3. In conclusion, except when you must use the “`//chapter`” exception discussed in Note 1, above, *avoid these extra-processing axes and axis combinations* (especially in outer XPath expressions) if your documents are relatively large and performance is a consideration.

12.2 Fixes in Sirius Mods 6.9 but not in 6.8

This section lists fixes to functionality existing in the *Sirius Mods* version 6.8 but which, due to the absence of customer problems, have not, as of the date of the release, been fixed in that version.

12.2.1 Reject invalid EndTag in Xml* deserialization methods

Previously, some invalid forms of end tag were accepted during XML deserialization. For example, the following was accepted:

```
%doc:LoadXml('<t></t foo="bar">')
```

Such invalid end tags are no longer accepted.

12.2.2 Accept UTF-16 input in Xml* deserialization methods

Previously, most UTF-16 inputs were not allowed during XML deserialization (for example, the `LoadXml` method). UTF-16 input is now accepted.

12.3 Version corequisites

This section lists any restrictions on usage of various products (including *Sirius Mods* itself) that will be imposed by use of version 6.9 of *Sirius Mods*.

- There are no corequisites associated with *Sirius Mods* 6.9.