
Notes for Sirius Mods Release 7.3

September, 2008



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 25, 2008

© 2008 Sirius Software, Inc.

Proprietary Notices

The following products:

- *Janus SOAP*
- *Sirius Functions*

are proprietary products of Sirius Software, Inc.:

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

Model 204™ is a proprietary product of Computer Corporation of America:

Computer Corporation of America
200 West St.
3rd Floor West
Waltham, MA 02451
USA

Contents

Proprietary Notices **ii**

Contents **iii**

Chapter 1: Introduction **1**

Chapter 2: Maintenance and Support **3**

 Documentation 3

Chapter 3: All or Multiple Products **5**

 Enhancement to Sirius regex 5

Chapter 4: Janus SOAP ULI **7**

 Utility classes 7

 RandomNumberGenerator class 7

 New constructor 8

 UpdateSeed subroutine 8

 Value function 9

 CharacterMap class 9

 Copy function 10

 DeepCopy function 10

 New constructor 10

 Update subroutine 10

 Constant methods 11

 Intrinsic Float class 11

 Absolute function 11

 Div function 12

 Modulus function 12

 Intrinsic String class 12

 Center function 13

 Char function 13

 OneOf function 13

 Replace and Remove functions 14

 Reverse function 14

 ToUpper and ToLower functions 14

 Translate function 14

 X function 15

 Screen classes 15

 SCRNSTBL user parameter 15

Chapter 5: Sirius Functions	17
\$Lstr_Word may return word longer than 255 bytes	17
Chapter 6: Compatibility/Bug fixes	19
Backwards incompatibilities	19
Janus SOAP ULI Intrinsic methods	19
Fixes in Sirius Mods 7.3 but not in 7.2	20
Version corequisites	20

CHAPTER 1 *Introduction*

This is a work-in-progress document describing possible contents of a software release.

Until the the commercial release of the software, we reserve the right to remove or change anything described herein.

This document lists the enhancements and other changes contained in *Sirius Mods* version 7.3, which is still under development. The previous version of the *Sirius Mods*, 7.2, was available in February, 2008.

CHAPTER 2 ***Maintenance and Support***

Sirius Mods version 7.2 supports only Model 204 version V6R1. *Sirius Mods* version 7.3 supports only *Model 204* version V6R1.0.

2.1 Documentation

- The *Janus Network Security Reference Manual* and the HTML pages of the SSL certificate management application are copyedited as well as updated to include information about client certificates and intermediate server certificates.
- The `FloatNamedArray` class has been available since the earliest releases of the *Janus SOAP ULI*, but its documentation (in the *Janus SOAP Reference Manual*) was not available until this release of the *Sirius Mods*.

3.1 Enhancement to Sirius regex

The following feature is added to the Sirius regular expression support (the regex methods in the `StringList` and intrinsic `String` class, and the `$RegexMatch` and `$RegexReplace` \$functions). The regex support is summarized in the “Regex Processing” chapters in the *Janus SOAP Reference Manual* and *Sirius Functions Reference Manual*.

- Support is added for the `\b` and `\B` multi-character escape sequences. They operate as they do under the Perl language.

Note: They are not supported in the following cases:

- Within a character class specification.
- Within values in replacement-string arguments in those methods and \$functions that have such arguments.

\b A word boundary anchor: a *position* where there is a word character (member of the `\w` character class) preceded by a non-word-character, or where there is a word character followed by a non-word-character. That is, `\b` succeeds at the word break that starts or ends a word. The `\w` character class is any letter (uppercase or lowercase), any digit, or the underscore.

For example, `\b` could match the following input string at the locations shown:

```
=hello 987
```

Matching locations:

- between “=” and “h”
- between “o” and blank
- between blank and “9”
- after “7”

\B The inverse of `\b`: a position that is not a word boundary.

For example, given the `=hello 987` string, `\B` could match between “h” and “e” but not between “o” and blank.

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

4.1 Utility classes

Two new system classes are the first members of the group of *Janus SOAP* “utility” classes: the `RandomNumberGenerator` class (“[RandomNumberGenerator class](#)”) and the `CharacterMap` class (“[CharacterMap class](#)” on page 9).

4.1.1 `RandomNumberGenerator` class

As its name suggests, the `RandomNumberGenerator` class is designed to generate random numbers. It is patterned after the Sirius functions `$Random` and `$Random_Seed`, described in the *Sirius Functions Reference Manual*.

The class differs from the functions in these important ways:

- For flexibility, the seeds specified to the methods are arbitrary length longstrings, not numbers. The longstrings are MD5 hashed to be used as a seed.
- Salt values are available on some methods to provide additional random data for creating a random seed. Like the seed, the salt is MD5 hashed and then included as part of the initial random data.

The methods in the class include a `New` constructor to instantiate an object, a `Value` method for printing as well as constraining the random number value range, and an `UpdateSeed` method to reset the value of an object.

As an example, the methods in this request create a reproducible sequence of ten numbers between 0 and 99:

```
b

%rand is object randomNumberGenerator
%i    is float

%rand = new(seed='abcd')
  for %i from 1 to 10
    printtext {~} = {%rand:value(0, 99)}
  end for

end
```

The individual methods are described briefly in the following subsections.

4.1.1.1 New constructor

The New method creates a random number. The kind of random number depends on the parameters specified when New is invoked:

- If no parameters are specified, New returns an unpredictable random number each time it is called.
- If a Salt parameter is specified, New returns an unpredictable random number that is based on initial input that has an extra degree of randomness.
- If only a Seed parameter is specified, New returns the next number in a specific series of pseudo-random numbers. Subsequent identical calls produce the next items in the series; recreating or resetting the object with the same seed lets you reproduce the series.

```
%rand = new([Seed=seed,] [Salt=salt])
```

New syntax

4.1.1.2 UpdateSeed subroutine

This method sets a random number object to the number that would be created by the New method (“[New constructor](#)”) with the given Seed and Salt specifications.

```
%rand:UpdateSeed([Seed=seed,] [Salt=salt])
```

UpdateSeed syntax

4.1.1.3 Value function

This method returns the value, within a default or explicit range, of the random number method object. Using this method is the only way to write the random number referred to by a RandomNumberGenerator object variable.

```
%num = %rand:Value([min,] [max])
```

Value syntax

4.1.2 CharacterMap class

A CharacterMap class object contains a mapping of characters to characters. Each character in an In string (the “input table”) is associated with, or mapped to, an individual character from an Out string (the “output table”). The output table may be supplemented with instances of a pad character to ensure a one-to-one mapping with the input table characters.

CharacterMap includes a constructor, copy methods, and an update method for modifying the map. Most of these methods are shown in the following example, which features the Translate method for longstrings (“[Translate function](#)” on page 14). In the example, a new CharacterMap is the argument for the Translate method; then that map is copied and modified, and the modified map is used in a second Translate call:

```
begin
%map is object characterMap
%map2 is object characterMap
%ls is longstring

%map = new(in='x-', out='!c')
%ls = 'xu--exx'
printtext {~} = '{%ls:translate(%map)}'

%map2 = %map:copy
%map2:update(in='x',out='s')
printtext {~} = '{%ls:translate(%map2)}'

end
```

The result is:

```
%ls:translate(%map) = '!ucce!!'
%ls:translate(%map2) = 'success'
```

The individual methods are described briefly in the following subsections.

4.1.2.1 **Copy function**

This method returns a CharacterMap object that is a copy of the method object.

```
%newmap = %map:Copy
```

Copy syntax

4.1.2.2 **DeepCopy function**

This method returns a CharacterMap object that is a deep copy of the method object.

```
%newmap = %map:DeepCopy
```

DeepCopy syntax

4.1.2.3 **New constructor**

The New method creates a character map, which associates individual characters from an In string (the “input table”) with the characters in the corresponding position in the “output table”. The output table consists of the characters in the Out string (plus copies of the Pad character, if necessary).

```
%map = New(In=string, [Out=string], [Pad=char])
```

New syntax

4.1.2.4 **Update subroutine**

The Update method updates the method object's current character mapping. The method lets you modify existing character associations or add new ones. The method's parameters and mapping rules are the same as the New method.

```
%map:Update(In=string, [Out=string], [Pad=char])
```

Update syntax

4.2 Constant methods

Constant methods are methods that belong to system classes but are unique because they are evaluated at compilation time. These methods require constant inputs and may have no parameters, but they have no run-time overhead.

For example, the following Constant method sets %x to a string with hex value x'0678':

```
%x = '0678':x
```

The hex conversion is done at compilation time. The method in this example (named X) is a compile-time-only equivalent of the intrinsic String class HexToString method.

As other intrinsic methods, the term “method object” is used for the constant value to which a Constant method is applied, even though the value is not really an object.

Unlike other intrinsic methods, Constant methods may be invoked only against constants. A variable as method object is **not allowed**:

```
%x = %y:x
```

Also, a statement like `%x = 1234:x` is **not** allowed, because a Constant method requires the constant to have the correct datatype (in this case, String).

Once a Constant method is evaluated, the expression compilation is replaced by the constant result of the evaluation, and all intermediate work quads/variables are deleted. Thus, `'0d25':x` is basically a hex constant.

The X function is the only Constant method as of *Sirius Mods* version 7.3.

```
%out = string:X
```

X syntax

4.3 Intrinsic Float class

These new methods are added: Absolute, Div, and Modulus.

4.3.1 Absolute function

This function returns a number that is the absolute value of the method object.

```
%value = number:absolute
```

Absolute syntax

The following statement returns 6.4 to %y:

```
%y = -6.4:absolute
```

4.3.2 Div function

This function returns the integer part of the result of dividing the method object by the method argument.

The numbers to be divided are first rounded to the nearest integer, including zero. Any remainder from the division is ignored.

```
%quo = number:Div(num)
```

Div syntax

The following statement returns -95 to %z:

```
%z = -191.4:div(2.3)
```

4.3.3 Modulus function

This function returns the remainder from the division of the method object by the method argument. The numbers to be divided are first rounded to an integer, including zero, so the remainder is always an integer or zero.

Mod is a synonym for Modulus.

```
%mod = number:Mod[ulus](num)
```

Modulus syntax

The following statement returns -1 to %z:

```
%z = -191.4:mod(2.0)
```

4.4 Intrinsic String class

Multiple new methods are added, as described in the following subsections.

4.4.1 Center function

The Center function is introduced. For a given input string and suitable specified length, this function returns a string of the requested length that has the original string embedded in the center. If the requested length is larger than the length of the input string, pad characters are added to the input string to produce the returned string. If the requested length is shorter, the input string's front and end are cropped to produce the returned string.

`Centre` is a synonym for the Center function.

```
%out = string:Center(length, [Pad=char,] [OffsetLeft=bool])
```

Center syntax

Note: This method is much like the `$Center` function except in cases where an uneven number of characters is to be padded or cropped. `$Center` puts the extra character on the right when padding and on the left when cropping. The Center method by default puts the extra character on the right in both cases, and it provides an optional parameter if you want to use the left side instead for the extra character.

4.4.2 Char function

The Char function returns one byte of the method object string, at the position requested by its argument.

```
outStr = string:Char(position)
```

Char syntax

If the position exceeds the length of method object string, the request is cancelled; otherwise, Char is the same as the Substring function with the given position argument and with the length argument set to 1.

4.4.3 OneOf function

This function checks if the method object string is matched by one of the strings in an input list of strings. The input list of strings is concatenated in a single delimited longstring argument to the method. A return code of 1 signals a successful match.

```
%rc = string:oneof(stringSet)
```

OneOf syntax

4.4.4 Replace and Remove functions

The Replace method replaces one or multiple occurrences of a substring of an input string (the method object) with a substitute string, and it returns the modified version of the input string. The Remove method removes one or multiple occurrences of a specified substring from an input string, and it returns the modified version of the input string.

```
%out = string:Replace(substring, replacement, [Count=num])
```

Replace syntax

```
%out = string:Remove(substring, [Count=num])
```

Remove syntax

4.4.5 Reverse function

The Reverse function is introduced. This function returns the characters in the method object string in right-to-left order.

```
outStr = string:Reverse
```

Reverse syntax

4.4.6 ToUpper and ToLower functions

The ToUpper and ToLower function are introduced. These functions return the alphabetic characters in the method object string as all-uppercase or all-lowercase characters, respectively. Non-alphabetic characters are returned as is, and the input string undergoes no other change.

```
outStr = string:ToUpper
```

```
outStr = string:ToLower
```

ToUpper and ToLower syntax

4.4.7 Translate function

The Translate method for longstrings is a method that is roughly equivalent to \$lstr_translate.

The Translate method takes a CharacterMap object as input. A CharacterMap object ([“CharacterMap class” on page 9](#)) maps In (“input table”) characters to Out (“output table”) characters.

In the following request, a's are translated to B's, and c's are translated to d's, as dictated by the character mapping in %map:

```
Begin
%map is object characterMap
%ls is longstring

%map = new(in='ac',out='Bd')
%ls = 'aaaccc'
printtext {~} = '{%ls:translate(%map)}'
End
```

The result is:

```
%ls:translate(%map) = 'BBBddd'
```

4.4.8 X function

This function returns the unencoded value of a hex-encoded string. It is a compile-time-only equivalent of the HexToString intrinsic method. Since in use the method acts like a hex constant, it is documented with the new Constant methods ([“Constant methods” on page 11](#)).

4.5 Screen classes

4.5.1 SCRNSTBL user parameter

The default value of the SCRNSTBL user parameter is changed from 4096 to 6144. SCRNSTBL specifies the maximum amount of STBL space available to an application that uses a Screen object.

This change is propagated by maintenance to versions 7.2 and 7.1 of the *Sirius Mods*.

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

5.1 \$Lstr_Word may return word longer than 255 bytes

The \$Lstr_Word function now may return a Longstring whose length exceeds 255 bytes. Previously, attempting to return such a result caused either request cancellation or truncation to 255 bytes of the returned string.

This change was also introduced as maintenance in versions 7.1 and 7.2 of the *Sirius Mods*

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 (7.2).

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).

- There are no backwards incompatibilities, that is, the results of successful processing with *Sirius Mods* version 7.2 are identical, when compared with the same inputs to *Sirius Mods* version 7.3.

6.1 Backwards incompatibilities

Backwards incompatibilities are described per product in the following sections.

6.1.1 Janus SOAP ULI Intrinsic methods

The following backwards compatibility issues have been introduced in *Janus SOAP ULI* Intrinsic methods:

- Methods that have a single-character-only parameter

The Right, Left, and Substring methods, which were available as of *Sirius Mods* 7.2, have a Pad parameter that is not allowed to be more than a single character. Longer values caused a request cancellation. In *Sirius Mods* 7.3, the compiler enforces the single-character-only rule for the Pad parameter for these methods, so such errors involving constants do not get to evaluation.

In theory, a *Sirius Mods* 7.2 user may have a too-long constant pad character for a Right, Left, or Substring method invocation in some code that does not cause a problem because it hasn't been executed. With this version 7.3 change, such a request would suddenly get a compilation error.

Note: Since the compiler can only detect bad constants, the single-character-only rule is still enforced at run time also.

6.2 Fixes in Sirius Mods 7.3 but not in 7.2

This section lists fixes to functionality existing in the *Sirius Mods* version 7.2 but which, due to the absence of customer problems, have not, as of the date of the release, been fixed in that version.

- There are no fixes in *Sirius Mods* 7.3 that are not available in previous versions.

6.3 Version corequisites

This section lists any restrictions on usage of various products (including *Sirius Mods* itself) that will be imposed by use of version 7.3 of *Sirius Mods*.

- There are no corequisites associated with *Sirius Mods* 7.3.