

---

# Notes for Sirius Mods Release 6.6

**August, 2004**

---



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

Telephone: (617) 876-6677  
FAX: (617) 234-1200  
E-mail: [support@sirius-software.com](mailto:support@sirius-software.com)  
World Wide Web: <http://sirius-software.com>

August 5, 2010

© 2010 Sirius Software, Inc.

---

## *Proprietary Notices*

The following products:

- *Janus SOAP*
- *SirFact*

are proprietary products of Sirius Software, Inc.:

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

**Model 204®** is a proprietary product of Computer Corporation of America, a wholly-owned subsidiary of Rocket Software, Inc., which owns the trademark:

**Rocket Software Corporate Office**  
**M204 Division**  
**275 Grove Street**  
**Suite 3-410**  
**Newton, Massachusetts 02466-2272**  
**USA**

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

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

---

## **Contents**

<b>Proprietary Notices</b> . . . . .	<b>ii</b>
<b>Contents</b> . . . . .	<b>iii</b>
<b>Chapter 1: Introduction</b> . . . . .	<b>1</b>
<b>Chapter 2: Maintenance and Support</b> . . . . .	<b>3</b>
Model 204 support . . . . .	3
<b>Chapter 3: All or Multiple Products</b> . . . . .	<b>5</b>
ASSERT statement availability . . . . .	5
\$list item length restriction . . . . .	5
Mixed-case User Language availability . . . . .	5
SIRMETH rules . . . . .	5
<b>Chapter 4: Janus SOAP ULI</b> . . . . .	<b>7</b>
User class definition changes . . . . .	7
HTML/Text statement To clause . . . . .	9
SIRCOMP user parameter . . . . .	9
System and subsystem classes . . . . .	9
Session objects . . . . .	10
Garbage collection . . . . .	10
Auto keyword on object declarations . . . . .	11
Recordset object manipulation methods . . . . .	11
Creating an empty record set . . . . .	12
Adding records to a record set . . . . .	12
Removing records from a Recordset . . . . .	13
ANDing Recordsets . . . . .	13
Making a copy of a record set . . . . .	14
Example . . . . .	15
<b>Chapter 5: Janus SOAP Xml* Classes</b> . . . . .	<b>17</b>
AddNamespace restrictions somewhat relaxed . . . . .	17
DeleteSubtree allows prefixed nodes . . . . .	17
Exists method . . . . .	18
LoadFromStringlist method . . . . .	19
LoadSystemMethodInfo method . . . . .	20
Print compaction options . . . . .	20

Serialization (non-Print) newline options . . . . .	22
Value is a settable property . . . . .	24
<b>Chapter 6:    SirFact . . . . .</b>	<b>25</b>
<b>Chapter 7:    Janus Sockets . . . . .</b>	<b>27</b>
HTTP Helper enhancements . . . . .	27
Handling server redirection . . . . .	27
File-based form uploads . . . . .	28
AddField method changes . . . . .	28
ContentToStringlist method . . . . .	29
LineEnd property . . . . .	31
AutoSendXMLContent property . . . . .	32
Automatic request header for Post calls . . . . .	32
Additional error checking . . . . .	32
Close is the only option for "connection" headers . . . . .	33
CLSOCK port enhancements . . . . .	33
Socket object method enhancements . . . . .	33
<b>Chapter 8:    Janus Web Server . . . . .</b>	<b>35</b>
HTTPVERSION parameter for JANUS DEFINE . . . . .	35
WEBOPT system parameter . . . . .	35
<b>Chapter 9:    Compatibility/Bug fixes . . . . .</b>	<b>37</b>
Backwards incompatibilities . . . . .	37
Janus SOAP XML processing . . . . .	37
Janus SOAP ULI . . . . .	37
Janus Sockets . . . . .	38
Fixes in <b>Sirius Mods</b> 6.6 but not in 6.5 . . . . .	39
Janus SOAP . . . . .	39
SirFact . . . . .	39
Version co-requisites . . . . .	39

---

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

This document lists the enhancements and other changes contained in *Sirius Mods* version 6.6, which was released in August, 2004. The previous version of the *Sirius Mods*, 6.5, was available in May, 2004.



---

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

As of version 6.6, *Sirius Mods* no longer supports *Model 204* version V4R2.1 and V5R2. The only supported versions are 4.2.0, 5.1, and 5.3.



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 ASSERT statement availability**

The ASSERT statement is now available to all customers that have any Sirius “API” products such as *Janus Web Server*, *Janus SOAP*, *Janus Sockets*, or *Fast/Unload User Language Interface*. For more information about the ASSERT statement see the ***Janus SOAP Reference Manual***.

### **3.2 \$list item length restriction**

For most purposes, \$list and Stringlist items can now be up to 2\*\*31-1 bytes long.

### **3.3 Mixed-case User Language availability**

Mixed-case User Language is now available to all customers that have any Sirius “API” products such as *Janus Web Server*, *Janus SOAP*, *Janus Sockets*, or *Fast/Unload User Language Interface*. For more information about mixed-case user Language see the ***Janus SOAP Reference Manual***.

### **3.4 SIRMETH rules**

The SIRMETH command provides system managers with the ability to allow specific subsystems or even all users, whether or not they are system managers, to do certain things that previously only system managers and pre-compiled subsystem procedures could do. These things are:

- Set system globals and strings.
- Set subsystem globals and strings.
- Set a subsystem context.

The format of the SIRMETH command is

```
SIRMETH {ALLOW | DISALLOW} -  
        {SYSTEMSET | SUBSYSTEMSET | -  
        SUBSYSTEMCONTEXT csubsys | ALL} -  
        [SUBSYSTEM subsys [NONPRE] ]
```

For more information about the SIRMETH command, see the ***Janus SOAP Reference Manual***.

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

## 4.1 User class definition changes

- New and Discard may be disallowed outside the class

The Public block of a class definition may now contain a Disallow qualifier for the New and Discard system methods. This option prevents users of a class from invoking either the New function or the Discard subroutine outside of the class definition.

The Disallow qualifier must be specified in the Public (non-shared) block, as in the following example:

```
class stooge
  public
    variable funiness    is float
    function age         is float
    subroutine slap(%howHard is float)
    disallow new
  end public
end class
```

Disallow is ignored for methods inside the class: If Disallow is specified in the class definition for New or Discard, you may still issue a New or Discard within the class wherever you want.

- Private constructors are allowed

You can now define constructor methods in the Private block of a class definition. Such constructors may only be invoked from inside the class.

If you want to define a private constructor named New, you must make sure to specify Disallow New in the Public block of the class definition. No other private constructors require an additional Disallow New in the Public block.

- Optional parameters allowed on methods.

Optional parameters are now allowed on method definitions. To indicate that a parameter is optional, simply specify the keyword `Optional` after its declaration:

```
function showMe(%itemNo is float, -
                %language is string len 32 optional)
```

When a parameter is indicated as optional, it can be specified on the method invocation or not:

```
%thingy:showMe(%which, 'Hungarian')
...
%thingy:showMe(%which)
```

Optional parameters that are not passed on invocation appear as standard default values inside the methods. In the above example, if a language is not specified on the method invocation, the value of `%language` inside the method is null. If you prefer something other than the standard default value in the method, you can use the `Default` keyword to indicate this value:

```
function showMe(%itemNo is float, -
                %language is string len 32 default('French'))
```

Obviously, the `Default` keyword implies `Optional`.

If the presence or absence of a parameter has meaning beyond simple replacement by a default value, you can use a `Present` or `Not Present` operation to test whether the parameter was specified:

```
if %language is not present then
    %language = %usersDefaultLanguage
end if
```

## 4.2 HTML/Text statement To clause

The HTML or Text statement now accepts a To clause that can be followed by a Stringlist that is to receive the output from the HTML or Text statement. In the following, the first few lines of Hamlet's most famous soliloquy are added to Stringlist

`%quoteList:`

```

%quoteList  is object stringList
...
%quoteList = new
...
text to %quoteList
    To be, or not to be: that is the question:
    Whether 'tis nobler in the mind to suffer
    The slings and arrows of outrageous fortune,
    Or to take arms against a sea of troubles,
    And by opposing end them?
end text

```

For more information about the HTML/text statement To clause, see the *Janus SOAP Reference Manual*.

## 4.3 SIRCOMP user parameter

This new system parameter is a standard *Model 204* bitmask type parameter where the X'01' bit means that string literals in HTML/Text statements are to be stored in CCATEMP rather than STBL. This is a compilation time parameter, and it can save large amounts of STBL space used to hold string literals (at a potential CPU cost in accessing the literals in CCATEMP).

## 4.4 System and subsystem classes

Shared methods in the system and subsystem classes are provided to allow manipulation of system-wide and subsystem-wide entities. The following is a very brief synopsis of the new methods:

- `%(system):SetGlobal(name, value)`  
`%(subsystem):SetGlobal(name, value)`
- `%(system):DeleteGlobal(name)`  
`%(subsystem):DeleteGlobal(name)`
- `%success = %(system):SetString(name, value, testvalue)`  
`%success = %(subsystem):SetString(name, value, testValue)`
- `%(system):DeleteString(name)`

`%(subsystem):DeleteString(name)`

- `%(system):String(name)`  
`%(subsystem):String(name)`
- `%(stringList = %(system):ListOfGlobals(gname)`  
`%(stringList = %(subsystem):ListOfGlobals(subsysName, gname)`
- `%(stringList = %(system):ListOfStrings(sname)`  
`%(stringList = %(subsystem):ListOfStrings(subsysName, sname)`

For more information on these methods see the *Janus SOAP Reference Manual*.

## 4.5 Session objects

In addition to global objects, session objects are now available. Session objects are very much like global objects, except instead of being associated with a login, they are associated with a logical session that can span multiple logins or be passed between logged-in threads. These sessions are the same sessions managed with the `$session` functions documented in the *Sirius Functions Reference Manual*.

Session objects, like global objects, can be statically bound to `%variables`:

```
%ruby is object jewel session
```

Or they can be set and retrieved dynamically using Object class methods:

```
%(object):getSession(%bauble, 'DIAMOND')  
...  
%(object):setSession(%bauble, 'EMERALD')
```

For more information about these methods, see the *Janus SOAP Reference Manual*.

## 4.6 Garbage collection

The *Janus SOAP ULI* now supports garbage collection — the cleaning up of orphaned object cycles. Garbage collection can be initiated with the Object class `GarbageCollect` method:

```
%(object):GarbageCollect
```

Garbage collection is too complex a topic to cover in these Release Notes; before using this method, see the *Janus SOAP Reference Manual* for more details about garbage collection.

## 4.7 Auto keyword on object declarations

System objects can now be declared with the `Auto` keyword, followed by either of these:

- The term `New`
- An enumeration value (if declaring an enumeration)

Using `Auto` indicates that the first reference to the object variable (if it is not an assignment to the variable) causes an instance of the object to be created, as if it had been explicitly instantiated. For example:

```
%colors    is collection arrayList of string len 32 auto new
%colors:add('Magenta')
```

is equivalent to

```
%colors    is collection arrayList of string len 32
%colors = new
%colors:add('Magenta')
```

The `Auto New` clause can be used in object declaration, even if the declaration is in a `Public`, `Private`, `Public Shared`, or `Private Shared` block in a `Class` declaration.

This facility is especially useful in helping to avoid having to write a constructor for an object just to instantiate collections or `ArrayLists` inside the object. It also avoids having to worry about null values for an enumeration variable that typically has a default value:

```
%expertUser    is enumeration boolean auto true
```

## 4.8 Recordset object manipulation methods

These methods are introduced in version 6.6 and described in the following sections:

```
New constructor
AddRecordset subroutine
AddRecord subroutine
RemoveRecordset subroutine
RemoveRecord subroutine
AndRecordset subroutine
Copy function
```

### 4.8.1 Creating an empty record set

To create a Recordset object that references an empty record set, you can now use the `New` constructor of the Recordset object. The Recordset object constructor has one optional parameter: a LockStrength enumeration. If no LockStrength value is specified, a default value of Share is used (as is consistent with the Record object).

For example:

```
%rs is object recordSet in foo
%rs = new
%rs = new( none )
%rs = new( exclusive )
%rs = new( share )
```

### 4.8.2 Adding records to a record set

To add records to the set referenced by a Recordset object, the `AddRecordset` and `AddRecord` Recordset object methods are now available. `AddRecordset` adds a Recordset object's records to the records in an existing Recordset object:

```
%target:AddRecordset( recordSetObject )
```

`AddRecord` adds the record in a Record object to the records in an existing Recordset object:

```
%target:AddRecord( recordObject )
```

For these methods, the lock strength of the records being added to the set must be greater than or equal to the lock strength of the set being added to. Consequently, adding records to a record set will not cause the Recordset object to try to acquire a record lock that it does not have. Therefore, these methods never cause a record locking conflict. Within a recordset, all records are always locked at the same strength (the lock strength with which the object was created).

In summary, these locking rules are in effect:

- To a set locked at exclusive level, you may add only records that are locked at exclusive level.
- To a set locked at share level, you may add records locked at exclusive level or share level.
- To an unlocked set, you may add records locked at any strength (exclusive, share, none).

Any violation of these rules results in request cancellation.

If the `recordSetObject` parameter of `AddRecordset`, or the `recordObject` parameter of `AddRecord`, is null, the request is cancelled. These parameters must have the same file context as the object for which the method is being called, or a compilation failure will result.

These two methods take the place of `PLACE RECORD/RECORDS`. Effectively, they are an OR operation.

### **4.8.3 Removing records from a Recordset**

To remove records from any set, the following Recordset object methods may now be used.

To remove a record set:

```
%target:RemoveRecordset( recordSetObject )
```

To remove a single record:

```
%target:RemoveRecord( recordObject )
```

If the `recordSetObject` parameter of `RemoveRecordset`, or the `recordObject` parameter of `RemoveRecord`, is null, the request is cancelled. These parameters must have the same file context as the object for which the method is being called, or a compilation failure results.

These two methods take the place of `REMOVE RECORD/RECORDS`.

Since the methods simply remove records from a set, the records being removed can be of any lock strength.

### **4.8.4 ANDing Recordsets**

To AND two Recordset objects, the `AndRecordset` method may now be used:

```
%target:AndRecordset( recordSetObject )
```

If the `recordSetObject` parameter is null, the request is cancelled. The parameter must have the same file context as the object for which the method is being called, or a compilation failure results.

Since ANDing can only remove records from a set, there are no restrictions on lock strength.

### **4.8.5 Making a copy of a record set**

To add or remove records from a set, but keep the original set around, you can now use the `Copy` function of the Recordset object:

```
%clone = %original:Copy
```

This method takes an optional parameter: a LockStrength enumeration. As with the `New` constructor, the default lock strength is `Share` if the LockStrength parameter is unspecified. You can only create a copy of lock strength less than or equal to the original, so `Copy` cannot produce a record locking conflict.

### 4.8.6 Example

Here is a short example that includes each of the new Recordset methods:

```

begin
  %baseStooges is object recordset in file glwproc
  %copyStooges is object recordset in file glwproc
  %curleyStooge is object recordset in file glwproc
  %shempStooge is object record in file glwproc

  * Create a base record set of the two 'invariant' stooges
  fd to %baseStooges name = 'MOE' or name = 'LARRY'
  end find
  text
  =====
  {%baseStooges:count} stooges were always present.
  Their names are:
end text
fr in %baseStooges
  text
  {name}
  end text
end for

* Using a recordSet object, add Curley to create stooges V1
%copyStooges = %baseStooges:copy
fd to %curleyStooge with name = 'CURLEY'
end find
%copyStooges:addRecordset( %curleyStooge )
text
=====
The 3 stooges version 1.0:
end text
fr in %copyStooges
  text
  {name}
  end text
end for

* Curley leaves ... remove him for the set
%copyStooges:removeRecordset( %curleyStooge )

* Using a record object, add Shemp to create stooges V2
in glwproc for 1 record with name = 'SHEMP'
  %shempStooge = %shempStooge:currentRecord
end for
%copyStooges:addRecord( %shempStooge )

text
=====
The 3 stooges version 2.0:
end text

```

```
fr in %copyStooges
  text
    {name}
  end text
end for

* Now Shemp is gone
%copyStooges:removeRecord( %shempStooge )

end
```

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

## 5.1 AddNamespace restrictions somewhat relaxed

The AddNamespace subroutine now adds a namespace declaration to an element even if the element already contains attributes. This was introduced as a maintenance change in version 6.5.

Also (not introduced as maintenance), AddNamespace allows the addition of a namespace to an element that has children, as long as the element has no Element children.

Finally, the concept of "implicit" declarations (explained in the version 6.5 documentation) is being retracted. The AddNamespace documentation will change to reflect the operation of AddSubtree (etc.) in version 6.5: all declarations are copied.

## 5.2 DeleteSubtree allows prefixed nodes

The DeleteSubtree method no longer prevents the deletion of nodes that have prefixes or URIs in the subtree being deleted.

Note that in two cases (only), DeleteSubtree should not be considered a complete "undo" operation of Add/Insert methods:

- If the URI argument of AddAttribute creates a namespace declaration, DeleteSubtree of the added attribute node does not delete the declaration:

```
%d = New
%n = %d:AddElement('x')
Print 'Before AddAttribute/Delete:' -
    And %d:Serial(, 'EBCDIC')
%n = %n:AddAttribute('p:a', 'v', 'ftp:a')
%n>DeleteSubtree
Print 'After AddAttribute/Delete:' -
    And %d:Serial(, 'EBCDIC')
```

This results in:

```
Before AddAttribute/Delete: <x/>
After  AddAttribute/Delete: <x xmlns:p="ftp:a"/>
```

- If the argument of AddSubtree (or InsertSubtreeBefore) references an attribute node that has a prefix not in scope at the target of the copy operation, DeleteSubtree applied to the added attribute node does not delete the declaration:

```
%d = New
%n = %d:AddElement('x')
%n2 = %n:AddElement('x1')
%n = %n:AddElement('x2')
%n2 = %n2:AddAttribute('p:a', 'v', 'ftp:a')
Print 'Before AddSubtree/Delete:'
Print %d:Serial(, 'EBCDIC')
%n2 = %n:AddSubtree(%n2)
%n2>DeleteSubtree
Print 'After  AddSubtree/Delete:'
Print %d:Serial(, 'EBCDIC')
```

This results in:

```
Before AddSubtree/Delete:
<x><x1 xmlns:p="ftp:a" p:a="v"/><x2/></x>
After  AddSubtree/Delete:
<x><x1 xmlns:p="ftp:a" p:a="v"/><x2 xmlns:p="ftp:a"/></x>
```

### 5.3 Exists method

The Exists function determines whether a given XPath expression returns a non-empty result:

```
%bool = nr:Exists(XPath)
```

This method returns True if the XPath result is non-empty, and False otherwise. For example, consider the following request:

```
Begin
%d Object XmlDocument
%d = New
Print 'Root exist?' And %d:Exists('.'):ToString
Print 'Elem of empty doc exist?' And %d:Exists('*'):ToString
End
```

The output is the following:

```
Root exist? True
Elem of empty doc exist? False
```

If you only want to check for the existence of some nodes in the document, this is the most efficient way to do so. The `SelectCount` method has the relative disadvantage that it may continue to examine the `XmlDoc` tree after the first matching node is found.

The `SelectSingleNode` method has the same relative disadvantage, but it is a little more subtle about the cases in which this occurs, because:

- The algorithm for evaluating a particular XPath expression may visit nodes in an order other than the document order.
- Since `SelectSingleNode` guarantees it will return the first node in `XmlDoc` order, it may continue even after the first match is found.

## 5.4 LoadFromStringlist method

The `LoadFromStringlist` function deserializes a text representation of an XML document that is contained in a `Stringlist` object:

```
%pos = doc:LoadFromStringlist(strLst, [options])
```

The `strLst` argument is a `Stringlist` object; otherwise this method behaves like the `LoadXml` method. The input consists of the concatenation of the `Stringlist` items; that is, there is no insertion of line end characters at the end of each item.

You can use this method, for example, to create XML documents as plain text for a test (or for some other application), as shown in the fragment below, which also makes use of the new `Text To` statement (see “[HTML/Text statement To clause](#)” on page 9):

```
%d Object XmlDoc
%d = New
%sl Object Stringlist
%sl = New
Text To %sl
  <test>
    <test2>
      supercalifragilisticexpailodocious
    </test2>
  </test>
End Text
%d:LoadFromStringlist(%sl)
```

## 5.5 LoadSystemMethodInfo method

The LoadSystemMethodInfo subroutine loads an XmlDocument with information about methods in Sirius classes:

```
Call doc:LoadSystemMethodInfo(pattern)
```

This method loads an XmlDocument with information about methods in classes selected by the “pattern” argument. Pattern is a string whose case is ignored and which can have any of the following forms:

### **methodPattern**

This selects, from any system class, all methods that match the specified pattern.

### **classPattern':'methodPattern**

This selects, from any system class matching *classPattern*, all methods that match *methodPattern*.

### **'System':classPattern':'methodPattern**

This is the same as `classPattern':'methodPattern`.

Note that the **New** method will only be displayed if either is true:

- The “methodPattern” is the string “New”.
- The **New** method has any arguments.

The LoadSystemMethodInfo method is primarily intended for User Language products delivered by Sirius Software, and the schema of the document may change from release to release. Also, it will not display classes in the module LOCAL, if they are selected by a pattern (that is, contains a wildcard character), or if there is no *classPattern* component.

## 5.6 Print compaction options

The following mutually exclusive Print method options are provided for more compact formats:

- **Compact:** An element's entire start tag is printed on a single line, which includes attributes and namespace declarations. If it has no children or has a single Text child, and it does **not** have attributes nor namespace declarations, the Text child is

serialized on the same line as the start and end tags. For example:

```
<top>
  <in1 a="xyz" b="foo">
    content1
  </in1>
  <in2>content2</in2>
</top>
```

Compact is the new default for the Print method; as mentioned in [“Janus SOAP XML processing” on page 37](#), this presents a compatibility issue.

- **Expanded:** A new line is started for each attribute, namespace declaration, and child. For example:

```
<top>
  <in1
    a="xyz"
    b="foo"
  >
    content1
  </in1>
  <in2>
    content2
  </in2>
</top>
```

Expanded is the format previously used.

- **AttributeCompact:** Attributes and namespace declarations are printed on the same line as the start tag. For example:

```
<top>
  <in1 a="xyz" b="foo">
    content1
  </in1>
  <in2>
    content2
  </in2>
</top>
```

- **ElementCompact:** An entire element is printed on one line, if it has no attributes nor

namespace declarations and has no children other than possibly a Text child. For example:

```
<top>
  <in1
    a="xyz"
    b="foo"
  >
  content1
</in1>
<in2>content2</in2>
</top>
```

- **BothCompact:** Combining the effect of AttributeCompact and ElementCompact, this is the most compacted option. It displays on one line an element that has no children or that has a single Text child. For example:

```
<top>
  <in1 a="xyz" b="foo">content1</in1>
  <in2>content2</in2>
</top>
```

These options make the Print method a little bit more like the format of the non-Print serialization methods, because the latter do not introduce any line breaks (by default). The non-Print methods also have some options to make them a little bit more like the Print method with the BothCompact option (see [“Serialization \(non-Print\) newline options”](#)).

As mentioned in [“SirFact” on page 25](#), the Compact form is used in the *SirFact* display of XmlDocument subtrees.

**Note:** The Compact and Expanded option keywords were also introduced as part of maintenance in version 6.5, and the default was also changed to Compact at that time.

## 5.7 **Serialization (non-Print) newline options**

A set of options is available for the following methods (in these classes):

- Serial (XmlDoc and XmlNode)
- Xml (XmlDoc)
- WebSend (XmlDoc)
- AddXml (HttpRequest)

The options provide a serialized stream that is easier to read (for example, with WebSend, using a browser's View/Source operation to look at XHTML produced with *Janus SOAP*). These options insert a “newline” sequence after serializing any of the following:

- an element start-tag, if it has any non-text node children
- an empty element tag or element end-tag
- a PI
- a comment
- a text node, if it has any siblings

This is analogous to the new BothCompact option of the Print method (“[Print compaction options](#)” on page 20). In addition, you can specify an indentation value for each level of the subtree.

The options available to any of these methods are:

**CR** Insert a carriage-return character as the newline sequence in the above cases.

**LF** Insert a newline character as the newline sequence in the above cases.

**CRLF** Insert a carriage-return character followed by a newline character as the newline sequence in the above cases.

In addition, a *Janus Web Server* connection or an HTTPRequest object has the ability to define the newline sequence for the connection or object, respectively. So, the WebSend and AddXml methods also allow the following option:

**Newline** Insert the newline sequence defined for the connection or object as the newline sequence in the above cases.

All the above options are mutually exclusive. The other option available for these methods is:

**Indent <n>** Where *n* is the indentation for each lower level in the serialized subtree. This option can be specified in conjunction with either CR, LF, CRLF, or Newline. It can also be specified without any of these for WebSend and AddXml, and in that case, it implies the Newline option.

So, Indent 2 (with a newline sequence) would produce:

```
<top>
  <leaf1 xx="yy">value</leaf1>
  <sub>
    <leaf2>value</leaf2>
  </sub>
</top>
```

## 5.8 Value is a settable property

The Value property in classes XmlDocument and XmlNode is now settable. It is not allowed if the head of the argument XPath result is either the Root or is an Element with more than one text child.

For example:

```
%worker:Value('billingRate') = %worker:Value('billingRate') - 10
```

---

**CHAPTER 6** *SirFact*

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

**Compact XML display** When displaying the contents of an XmlDocument, the Compact form of the XmlDocument Print method is used.



---

**CHAPTER 7** *Janus Sockets*

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

## **7.1 HTTP Helper enhancements**

### **7.1.1 Handling server redirection**

An HTTP server may return an indication that a page has moved and that the client process should fetch it from a new location. A server indicates this case by returning a response code in the 300s and by returning a response header named “location” that indicates the new location of the page or resource. The version 6.5 HTTP Helper does nothing special with redirects: the 300+ response code is returned to the application as if it were any other response.

In Version 6.6 of the Sirius Mods, your HTTP Helper application may use the new `MaxRedirects` property of the HTTP Request object to request automatic handling of redirects that a Get or Post method call receives from the server:

```
%HttpRequest:MaxRedirects
```

Since redirects may be chained, that is, the location to which a redirect leads can also return a redirect, `MaxRedirects` specifies the maximum number of times a redirect is followed when it is returned from a web server. Its default is zero (do not automatically handle redirects). If non-zero, it indicates the maximum number of times a redirect is followed on *one* Get or Post call. If the maximum number of redirections have been followed, and the last response is still a redirection response code, the last redirection response code is returned to the HTTP Helper application.

`MaxRedirects` may be set to an integer from 0 through 100. An attempt to set it to anything else results in request cancellation.

When `MaxRedirects` is set to a value greater than zero, the following HTTP redirection response codes are automatically handled:

301	Moved Permanently
302	Found
303	See Other
307	Temporary Redirect

For more details concerning HTTP Response codes, see <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>.

In addition, to facilitate working with redirects, the new `URL` method of the `HTTPResponse` object returns the actual URL from which a response was obtained. In the case of redirects, this will **not** be the URL requested on the `HTTPRequest` object. The `URL` method can inform an application that a redirection happened and what the real location of the requested item was. For example:

```
%HTTPResponse = %HTTPRequest:Get('XMLCLIENT')

if (%HTTPResponse:URL ne %HTTPRequest:URL) then
    print 'Hey Moe, we got redirected to ' and -
        %HTTPResponse:URL
end if
```

### 7.1.2 File-based form uploads

The HTTP Helper now supports Multipart Form encoding to provide browser-like handling of file uploads. A version 6.6 `HTTPRequest` object can send file data to a web site from either a longstring or a Stringlist. The Post of file data using this feature emulates an HTML form that uses `enctype="multipart/form-data"` and a `type="file"` input field.

To enable this feature:

1. Set to True the new boolean property, `MultiPartFormEncoding`, of the `HTTPRequest` object,
2. Add form data with `AddField`.
3. Use the `Post` function to send the request.

`MultiPartFormEncoding` must be set to True **before** any `AddField` statements are issued, or the request is cancelled.

### 7.1.3 AddField method changes

In version 6.6, the `HTTPRequest` method `AddField` is enhanced to support file uploads. The method has two new optional parameters, and an existing parameter has an additional type option.

Here is the new syntax for `AddField`:

```
%req:addField(fieldname, fieldvalue, [filename], [xlatflag])
```

Where:

- `fieldname`, a string that identifies the form field name, is unchanged from version 6.5.

- `fieldvalue` can now be either a `Stringlist` or a `LongString` (previously, only `longstrings` were allowed). If it is a `Stringlist`, a line-end sequence is appended to each `Stringlist` item as it is added to the `Post` data. The new `LineEnd` property (“[LineEnd](#)” on page 31) sets which of three line-end sequence options is appended.
- `filename`, a string, is a new optional parameter that identifies the file to be uploaded. The default is no file specification, that is, a non-file data `Post`.
- `xlatflag` is a new optional parameter that controls whether the `fieldvalue` content gets EBCDIC-to-ASCII translation. If `xlatflag` is not specified, the default is `True` (use EBCDIC-to-ASCII translation). If set to `False`, no translation occurs.

The following example prompts for a procedure name and uploads the procedure using the new features of the HTTP Helper.

```
begin

    %request is object HTTPRequest
    %response is object HTTPResponse
    %upfile is object Stringlist

    %procname is string len 255
    %rc is fixed

    %procname = $read('Enter proc name please')
    %rc = $procopn( %procname )
    if ( %rc ) then
        print 'could not open the proc, dude.'
        stop
    end if
    %upfile = new
    %upfile:AppendOpenProcedure
    %rc = $proccls

    %upfile:print
    %request = new
    %request:URL = 'whatever'
    %request:MultiPartFormEncoding = true
    %request:addField('filename',%upfile,%procname)
    %response = %request:post( 'CSOCK' )
    print %response:statusline
    print %response:content

end
```

#### **7.1.4 ContentToStringlist method**

This `HTTPResponse` object method places into a `Stringlist` object the content returned from an `HTTPRequest` `Get` or `Post` method. `ContentToStringlist` does the following:

1. Instantiates a Stringlist object.
2. Breaks the HTTPRequest returned content into lines by scanning for line-ends.
3. Translates each line from ASCII to EBCDIC, and places it in its own Stringlist element in the Stringlist it created.
4. Returns the Stringlist.

No single content line may exceed 6124 bytes in length (excluding the line end), or the request is cancelled. The scanned-for line-ends are the three typical characters (CRLF, CR, LF).

The ContentToStringlist syntax is:

```
%lines = %httpresp:ContentToStringlist
```

where `%lines` is a Stringlist object.

This method is useful when the content returned from the HTTP server is a series of ASCII text lines separated by line-end characters (for example, an HTML document). Accessing such content with the HTTPResponse Content method would leave all the lines in one longstring, which the application would have to separate into lines using string-handling User Language code.

ContentToStringlist is also useful for getting readable output for debugging purposes: For example:

```
%myGradeBookResponse:contentToStringlist:print
```

The statement above produces much more readable output than this:

```
print %myGradeBookResponse:content
```

The following ContentToStringlist example prompts for a URL, fetches the page at that URL, places it in a Stringlist using ContentToStringlist, and displays the lines.

```
begin

%HTTPRequest is object HTTPRequest
%HTTPResponse is object HTTPResponse
%myPage is object Stringlist
%i is float

%HTTPRequest = new
%HTTPRequest:URL = $read('Hey Moe, give me a URL!')
%HTTPResponse = %HTTPRequest:Get('XMLCLIENT', 0 )
if ( %httpResponse is null ) then
    print 'Could not connect'
    stop
end if

if ( %HTTPResponse:Code = 200 ) then
    %myPage = %HTTPResponse:ContentToStringlist
    print %myPage:Count and 'lines obtained.'
    for %i from 1 to %myPage:count
        print 'Line' and %i and ': ' and %myPage:item(%i) at 15
    end for
else
    print 'Sorry dude, you got an error: ' and -
        %HTTPResponse:statusline
end if
end
```

### 7.1.5 LineEnd property

The new `LineEnd` property of the `HTTPRequest` object lets you select which of three line-end sequences to use when adding a file to form Post data by passing a `Stringlist` object to the `AddField` method of the HTTP request object.

By default, each item in the `Stringlist` copied into the Post data has a line-end sequence appended to it. The line-end sequence CRLF (carriage-return/line-feed) is the default (hex value X'0D0A'). However, a client application may instead use carriage-return only (X'0D'), or line-feed (X'0A').

To specify an alternative line-end sequence for `AddField`, you set the `LineEnd` property of the `HTTPRequest` object. The `LineEnd` property sets and returns an Enumeration (the `LineEnd` enumeration), which has one of the following values:

- CRLF** Use carriage-return/line-feed. This is the default for a newly created `HTTPRequest` object.
- CR** Use only carriage-return.

**LF** Use only line-feed.

For example:

```
%HttpRequest:LineEnd = CRLF
%HttpRequest:LineEnd = CR
%HttpRequest:LineEnd = LF
%HttpRequest:AddField( ....
```

The LineEnd property must be set **before** the AddField call.

The LineEnd setting remains in effect unless the property is reset with another assignment.

### 7.1.6 **AutoSendXMLContent property**

This HttpRequest object method allows or suppresses the automatic generation of an HTTP content-type header for a Post produced by the HttpRequest AddXml method.

As of *Sirius Mods* version 6.6, if you do not explicitly set a content-type HTTP request header via AddHeader, AddXml generates a content type header of “text/xml” upon a Post. To suppress this automatic generation, you can set the `AutoSendXMLContentType` property (HttpRequest class) to False. This property sets and returns a Boolean enumeration that indicates whether AddXml automatic header generation is on or off.

### 7.1.7 **Automatic request header for Post calls**

As of *Sirius Mods* version 6.6, the HTTP Helper automatically sends the following request header on Post calls if no explicit content-type header is set, if form fields are set, and if multi-part form encoding is **not** enabled:

```
content-type: application/x-www-form-urlencoded
```

In version 6.5, to set this header you must do so explicitly using the AddHeader method of the HttpRequest object.

### 7.1.8 **Additional error checking**

These Post-oriented actions are incompatible with an HTTP Get, and they result in request cancellation if detected by Get function error checking:

- An AddXML call added Post data to the request.
- An AddLongString call added Post data to the request.
- The MultiPartFormEncoding property set to `True` enabled multipart form encoding.

### 7.1.9 Close is the only option for "connection" headers

In *Sirius Mods* 6.5, the HTTPRequest object of the HTTP Helper automatically generates the HTTP Request header "Connection: close" under HTTP 1.1. In *Sirius Mods* 6.6, if you try to set a "connection" header using the AddHeader method of the HTTPRequest object, any value except `close` will result in request cancellation.

**Note:** If you explicitly set "connection" to `close`, the automatic generation of "Connection: close" is suppressed to ensure that an extra connection header is not sent.

## 7.2 CLSOCK port enhancements

- MASTER parameter allowed

The MASTER parameter of the JANUS DEFINE command is now allowed for CLSOCK ports. A port defined as MASTER can be accessed on a *Janus Sockets* connection request without specifying its port name. The socket port name parameter on \$SOCK\_CONN, on the Socket object New constructor, and on the HTTP Helper Get and Post methods now defaults to the MASTER port, if any.

- Default for REMOTE parameter of JANUS DEFINE

REMOTE \* \* is now the default for CLSOCK ports. It is expected that this will be more convenient and desirable than the former requirement to explicitly designate a specific host or IP address. Explicit specification remains an option.

**Note:** If the REMOTE default setting is in effect for a CLSOCK port, REMOTE \* \* is not included in JANUS DISPLAY output.

## 7.3 Socket object method enhancements

The GetMySocketNumber, GetSocketObject, and ServerSocket methods are changed as follows:

- The GetMySocketNumber method is renamed GetSocketNumber.
- The GetSocketNumber method no longer allows references to the method socket after this method returns.
- GetSocketObject and ServerSocket no longer allow access to the sockets by number after they return.
- If the object returned by a GetSocketObject invocation is the method object for a subsequent GetSocketNumber invocation, the NOCLOSE setting of the original numbered socket is applied to the numbered socket returned by GetSocketNumber.

- The CLOSE and NOCLOSE keywords are no longer valid options for the Set method.

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

## **8.1 HTTPVERSION parameter for JANUS DEFINE**

The new JANUS DEFINE parameter HTTPVERSION allows changing of the HTTP version returned to the browser by *Janus Web Server*. This parameter should be used with caution (if at all).

## **8.2 WEBOPT system parameter**

The new WEBOPT parameter is a standard *Model 204* bitmask-type parameter. The X'01' setting means that RACF calls are to be performed in a special RACF subtask. If using *Janus Web Server* with RACF doing login validation, it is possible to get very high RACF call loads. Since these calls often involve synchronous I/Os that stop the *Model 204* main task, they are generally not very good for *Model 204* performance. Under the WEBOPT X'01' setting, these calls are done in a subtask, reducing their impact on Online performance dramatically.



This chapter lists any compatibility issues with prior versions of the *Sirius Mods*. It also lists any bugs that have been fixed in this version of the *Sirius Mods* but have not, as of the date of this release, been fixed in the immediately preceding version (6.5).

In general, backward incompatibility means that an operation that was previously performed without any indication of error now operates, given the same inputs and conditions, in a different manner. Possibly not listed as backwards incompatibilities are 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).

## **9.1 Backwards incompatibilities**

### **9.1.1 Janus SOAP XML processing**

In version 6.6 of *Janus SOAP*, the default option for the `XmlDoc` and `XmlNode Print` method has changed. It is now `Compact` (see “[Print compaction options](#)” on page 20 for an explanation). Note that the `Compact` and `Expanded` option keywords were also introduced as part of maintenance in version 6.5, and the default was also changed to `Compact` at that time.

### **9.1.2 Janus SOAP ULI**

#### **No duplicate labels in methods**

In version 6.5, duplicate labels within a method definition were allowed. As of version 6.6, all labels within a method definition must be unique, so the second `Sublab:` label below causes an error:

```
Begin
Class example
  Public
  Subroutine sub1
  End Public
  Subroutine sub1
    Jump To Sublab
  Sublab:
    Print 'Sub1'
    Jump To Sublab
  Sublab:
    Print 'Sub2'
  End Subroutine
End Class
%test Object example
%test = New
%test:sub1
Jump To Outlab
Outlab:
Print 'Out1'
Jump To Outlab
Outlab:
Print 'Out2'
End
```

Note: In the example above, the `Outlab:` labels that follow the invocation of the subroutine generate a message which, by default, is a warning message but which allows the request to execute.

### 9.1.3 Janus Sockets

- In version 6.5 of the HTTPHelper object, if no content-type header was set with the `AddHeader` method, an `AddXml` method `Post` sent no content-type header. In version 6.6, if you do not explicitly set a content-type HTTP request header via `AddHeader`, a content-type header of “text/xml” is automatically generated when you do a `Post`.

You can suppress this version 6.6 automatic header generation by setting the `AutoSendXMLContentType` property of the `HTTPRequest` object (new in version 6.6) to `False`.

- The `Socket` method `GetMySocketNumber` is renamed `GetSocketNumber`.
- The `GetSocketNumber`, `GetSocketObject`, and `ServerSocket` methods in the `Socket` class are changed so that a socket is either accessible by a number or an object, but not both. That is, after a `GetSocketNumber` call, the method object `socket` is no longer accessible as an object, and after a `GetSocketObject` or `ServerSocket` call, the socket can no longer be accessed by its number.

- The number of bytes used for each possible user socket (SOCKMAX) is changed from 5 to 6.

## **9.2 Fixes in *Sirius Mods* 6.6 but not in 6.5**

This section lists fixes to functionality existing in the *Sirius Mods* version 6.5 but which, due to the absence of customer problems, have not, as of the date of the release, been fixed in that version.

### **9.2.1 Janus SOAP**

- Previously, if an XmlDocument subtree that contained an explicit namespace declaration was deleted, an internal error was introduced into the XmlDocument, and subsequent usage could cause incorrect results. Such a declaration was necessarily unreferenced, since if it was referenced, deletion of a node with a URI would have been attempted, and the deletion operation would not have been allowed.

### **9.2.2 SirFact**

- Previously, when a SIRFACT NODUMP rule was matched, a message was issued that a file was not found; the name of the file consisted of 8 hexadecimal zeroes.

## **9.3 Version co-requisites**

This section lists any restrictions on usage of various products (including *Sirius Mods* itself) which will be imposed by use of version 6.6 of *Sirius Mods*.

- There are no corequisites associated with *Sirius Mods* 6.6.

