
Notes for Sirius Mods Release 7.8

September, 2010



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>

November 13, 2010

© 2010 Sirius Software, Inc.

Proprietary Notices

The following products:

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

are proprietary products of Sirius Software, Inc.:

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

Model 204® and **Connect *™** are proprietary products of Computer Corporation of America, a wholly-owned subsidiary of Rocket Software, Inc., which owns the trademarks:

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

SoftSpy™ is a proprietary product of Information Technology Systems:

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

Contents

Proprietary Notices	ii
Contents	iii
Chapter 1: Introduction	1
Chapter 2: Janus SOAP ULI	3
New arguments for Record class ToXmlDoc method	3
Field references in CurrentRecord methods	3
New exception class: BadJournal	4
New constructor	5
ReasonCode property	5
New SelectionCriterion methods: IsNull and IsNotNull	6
Chapter 3: Janus SOAP XmlDoc API	9
XPathError exceptions	9
The XPathError exception class	10
NewFromRecord shared function in XmlDoc class	11
LoadFromRecord subroutine in XmlDoc and XmlNode classes	14
AddToRecord subroutine in XmlDoc class	17
Copying from multiple source records	19
Removing update tracking fields from an XmlDoc	22
Structure of XmlDoc for AddToRecord	23
AddToRecordError exception class	25
MoveNamespace argument for AddTopElement	28
DeleteTopElement subroutine in XmlDoc class	29
Chapter 4: Fast/Reload	33
Warning messages for some DV, STORE-x, and repeatability changes	33
EXONE -> non-EXONE with STORE-x NONE	34
REPT -> EXONE	35
REPT -> ONE STORE-x NONE	35
ONE -> ONE with changed DV	36
ONE -> EXONE	36
ONE with DV -> REPT	37
ONE SD LIT/ALL -> SD NONE	37
Non-EXONE SN LIT/ALL -> SN NONE	38

Chapter 5: Compatibility/Bug fixes	39
Backwards incompatibilities	39
Janus SOAPXmlDoc API	39
AddToRecord constraint on 'number' attribute	39
DefaultURI argument of AddSubtree	39
Deserialization prohibit default namespace declaration with Namespace=None	40
Fixes in Sirius Mods 7.8 but not in 7.7	41
LoadSystemMethodInfo returning Unicode methods	41
Fast/Reload LAI with various UAI SORT cases	41
Version corequisites	42

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 7.8, which is still under development. The previous version of the *Sirius Mods*, 7.8, was available in March, 2010.

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

2.1 New arguments for Record class ToXmlDoc method

The ToXmlDoc method in the Record class has the following new arguments:

CodepageTable=bool This Boolean argument, which defaults to `False`, specifies whether to use the base codepage translation table when creating the XmlDoc. See the description of the `CodepageTable=False|True` argument in “LoadFromRecord subroutine in XmlDoc and XmlNode classes” on page 14.

Note that this argument was actually introduced in version 7.6 of the *Sirius Mods*.

AllowNull= bool The value of this Boolean argument, which defaults to `False`, is copied to the AllowNull property of the XmlDoc created by ToXmlDoc. The XmlDoc's AllowNull property, in turn, determines whether field values that contain the X'00' character are stored in the XmlDoc with base64 encoding. Such values are base64 encoded if AllowNull is `False`.

See the description of the `AllowNull=` argument in “NewFromRecord shared function in XmlDoc class” on page 11.

Note that this argument was actually introduced in version 7.7 of the *Sirius Mods*.

2.2 Field references in CurrentRecord methods

For methods declared with a CurrentRecord attribute, it was the case under *Sirius Mods* 7.7 that field references were an exception to the following rule:

Statements within the method definition, even a CurrentRecord method call, may reference the record without having to be wrapped inside a record For loop.

Under *Sirius Mods 7.8*, field references are no longer an exception to this rule. You may reference a record field from within a method declared with `CurrentRecord` without being inside a record `For` loop.

For example, for the field `COLOR`, the `For Record currentRecord` and `End For` statements containing the `print COLOR` statement in the method definition below may be discarded under *Sirius Mods 7.8*:

```
local subroutine (Record in file myproc):printField -
    currentRecord in file myproc
    for record currentRecord
        print COLOR
    end for
end subroutine
```

2.3 New exception class: **BadJournal**

The `BadJournal` exception class reports errors in `CCAJRNL` or `CCAJLOG` datasets or streams, including naming errors. The `New` method of the `Journal` system class is the system method that automatically throws a `BadJournal` exception.

The following example shows a `Try` and `Catch` of a `Journal` class, `New` method, exception. An invalid journal name is specified to generate the `BadJournal` exception:

```
Begin

%sl      is object stringlist
%rc      is float
%journal is object journal
%bdjrn1  is object BadJournal

try printtext {~} is: {%journal = new('OLD~RNL')}
    catch BadJournal to %bdjrn1
    Print 'Failure!!! Reason code is: ' %bdjrn1:reasonCode
end try

%rc = %sl:appendJournalData( -
    Options='MAXIO=1000 WIDTH=138 ST AA USER', -
    Threads='*', Journal=%journal)

Print %rc
Print %sl:count
%sl:print

End
```

The Stringlist AppendJournalData method does not cancel if its Journal parameter is null. The request result shows the reason code (ReasonCode property value) stored in the exception object:

```
%journal = new('OLD~RNL') is: Failure!!! Reason code is: 1
0
0
```

The methods of the class are described in the following subsections.

2.3.1 New constructor

This constructor generates an instance of a BadJournal exception. As shown below, the optional argument of the New method is a setting of the ReasonCode property.

```
[%bdJrn1 =] [% (BadJournal):] New(ReasonCode=num)
```

New constructor syntax

2.3.2 ReasonCode property

This readOnly property returns a numeric reason code that indicates the cause of the BadJournal exception.

```
%rc = %bdJrn1:ReasonCode
```

ReasonCode syntax

Possible reason codes are:

- 1** Either the dataset or stream name is invalid, or the journal is invalid.
- 2** The dataset or stream is empty.
- 3** The journal was created with a different *Model 204* version than the current Online.
- 4** A merged journal is invalid.

2.4 New SelectionCriterion methods: IsNull and IsNotNull

These shared methods take no parameters and create a new SelectionCriterion object. The methods provide control for Null objects in the collection you are searching. They also let you determine whether a collection contains items that are objects, because they cancel the request if the collection being searched contains non-object (intrinsic type) items.

An IsNull criterion selects a collection item if the item is a Null object; an IsNotNull criterion selects an item objects if it is not Null.

The syntax of the methods follows:

```
%selc = IsNull
%selc = IsNotNull
```

The examples below test a variety of searches against arraylist %a1 of objects of class T:

```
class T
  public
    variable x is float
  end public
end class

%a1 is arraylist of object t
%t is object t
%t1 is object t
%t2 is object t

%t1 = null
%t2 = new
%a1 = list(%t1, %t2)
```

1. The Arraylist class FindNextItem method, which throws an exception if its selection criterion matches no item, fails in the Try clause below when it tests the Null object item. The method's exception is not thrown because the test failure prevents the method from completing its search:

```
try %t = %a1:findNextItem(EQ(x,1))
  printtext found t
  printtext {~} = {%t:x}
catch itemNotFound
  printText None!
end try
```

The result is:

```
CANCELLING REQUEST: MSIR.0750: Class ARRAYLIST, function  
  FindNextItem: reference to null object in line xx
```

To complete this request without cancellation, you can use an IsNotNull criterion to bypass Null items:

```
try %t = %a1:findNextItem(AND(isNotNull, eq(x,1)))  
  printtext found t  
  printtext {~} = {%t:x}  
catch itemNotFound  
  printText None!  
end try
```

The search finds no matching items, so the Catch clause above catches the method's ItemNotFound exception, and the result is:

```
None!
```

2. Instead of bypassing Null items, you might instead want the search to include them:

```
try %t = %a1:findNextItem(OR(isNull, eq(x,1)))  
  printtext found t  
  printtext {~} = {%t:x}  
catch itemNotFound  
  printText None!  
end try
```

The Null item is found, but the Try clause PrintText invocation of %t:x fails, and the result is:

```
CANCELLING REQUEST: MSIR.0561: Text output:  
  reference to null object in line xx
```

If you want to search exclusively for the next Null item in a collection, you can simply use this:

```
%t = %a1:findNextItem(isNull)
```

3. To successfully locate the non-Null item in %a1, you could use either of the following method calls in the Try clause:

```
%t = %a1:findNextItem(isNotNull)  
%t = %a1:findNextItem(AND(isNotNull, eq(x,0)))
```

Thanks to the change in the EQ criterion in the second call above, the result of trying either of these searches is:

```
found t  
%t:x=0
```


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

3.1 XPathError exceptions

All XmlDocument and XmlNode class methods that accept XPath arguments now can throw an XPathError exception, which is described in [“The XPathError exception class” on page 10](#).

An example of using the XPathError exception is:

```
%err Object XPathError
Try
    %d:Print('a b c')
Catch XPathError To %err
    PrintText {~} = {%err:Reason}
    PrintText {~} = {%err:Description}
    PrintText {~} = {%err:CharacterPosition}
End Try
```

Since the expression in the above invocation of the Print method (a b c) is not a valid XPath expression, the above fragment will result in the following:

```
%err:Reason = SyntaxError
%err:Description = Expect "/" for new step or "[" for
predicate
%err:CharacterPosition = 3
```

The methods in the XmlDocument and XmlNode classes that can throw an XPathError exception are:

- Audit
- DefaultURI
- DeleteSubtree
- Exists
- Length
- LocalName
- Prefix
- PrefixURI
- Print
- QName

- SelectCount
- SelectNodes
- SelectSingleNode
- Serial
- ToXPathStringlist
- Trace
- Type
- UnionSelected
- URI
- Value
- ValueDefault
- XPathNodeID

3.1.1 The XPathError exception class

The members of the XPathError exception class are described below. Except for the constructor, `New`, all class members are read-only properties:

Reason An enumeration of type `XmlPathErrorReason`. The possible values are:

SyntaxError A violation of the syntax of an XPath expression.

EmptyResult There are no nodes matched by the XPath expression, and the method requires at least one matching node.

CharacterPosition

The position within the XPath expression at or before which the error was detected.

Description

A message that explains the error.

New The constructor for the class, `New` lets you set values for each member of the class.

```
%ex = New ( Reason = reasonEnum          -  
           [, CharacterPosition = num] -  
           [, Description = string] )
```

The Reason argument is required; all other arguments are optional, name required, and have default values of the null string or 0 as appropriate.

3.2 NewFromRecord shared function in XmlDoc class

This shared function creates a new XmlDoc object which contains the fields and fieldgroups from the current record. This record extraction is the same operation that is performed by the LoadFromRecord subroutine (“LoadFromRecord subroutine in XmlDoc and XmlNode classes” on page 14) and by the ToXmlDoc function in the Record class.

Whether to use ToXmlDoc, NewFromRecord, or LoadFromRecord depends on what is most convenient for your application. If you are already using a Record object which references the desired record, using ToXmlDoc may be more convenient; if not, then either NewFromRecord or LoadFromRecord (both of which require that the method be contained in a “record loop”) may be more convenient. You must use LoadFromRecord if you want to add the record's content as a subtree to a non-empty XmlDoc; in other cases the NewFromRecord “factory method” may be your choice.

Since NewFromRecord and ToXmlDoc create new XmlDoc objects, they have the `AllowNull=` argument for setting the created XmlDoc's AllowNull property; LoadFromRecord does not have the `AllowNull=` argument.

As stated, both NewFromRecord and LoadFromRecord must be contained in a “record loop”, for example, an FRN block, and they may not be invoked within a fieldgroup context.

Except for these considerations, ToXmlDoc, NewFromRecord, and LoadFromRecord all perform the same operation and have the same arguments. In our discussion of the “extract from record to XmlDoc” operation, we will generally use LoadFromRecord, except where one of these considerations is relevant to the discussion.

The arguments to NewFromRecord are:

```
%newXmlDoc = %(XmlDoc):NewFromRecord( -
    [AttributeValues=bool]         -
    [, AttributeNames=bool]       -
    [, NamesToLower=bool]         -
    [, AllowUnreversible=bool]    -
    [, CodepageTable=bool]        -
    [, AllowNull=bool] )
```

Some of the arguments are described here:

%newXmlDoc NewFromRecord returns a new XmlDoc object.

%(XmlDoc) The class name in parentheses denotes a shared method and is one way to invoke NewFromRecord; you can also use an object expression whose type is XmlDoc (even if the value of the expression is null).

AllowNull=*bool*

The value of this Boolean argument, which defaults to `False`, is copied to the `AllowNull` property of the `XmlDoc` created by `NewFromRecord`. The `XmlDoc`'s `AllowNull` property, in turn, determines whether field values that contain the `X'00'` character are stored in the `XmlDoc` with base64 encoding. Such values are base64 encoded if `AllowNull` is `False`.

A `False` value of the `AllowNull` property of an `XmlDoc` is its default. This prevents null characters from being stored in the `XmlDoc`, so it will be conformant to the XML Recommendation, which does not allow null characters in an XML document.

The following fragment:

```
%s = 'Field with null/' With '00':X With
 '/'
Store Record
    F00 = %s
End Store
%r = $CurRec
FRN %r
    %doc = %doc:NewFromRecord
End For
PrintText {~} = {%doc:AllowNull}
%doc:Print
```

produces this output:

```
%doc:AllowNull = False
<Record version="1" file="QAWORK"
number="1">
    <field name="F00" encoding="base64">
        xomFk4RApomjiECVpJ0TYQBh
    </field>
</Record>
```

In the above output, notice that `FOO` is base64 encoded, because it contains a null character, and null characters are not

allowed in an XmlDoc whose AllowNull property is `False`. The following fragment:

```
%s = 'Field with null/' With '00':X With
 '/'
Store Record
  F00 = %s
End Store
%r = $CurRec
FRN %r
  %doc =
%doc:NewFromRecord(AllowNull=True)
End For
PrintText {~} = {%doc:AllowNull}
%doc:Print
```

produces the following output:

```
%doc:AllowNull = True
<Record version="1" file="QAWORK"
number="1">
  <field name="F00">
    Field with null/00;/
  </field>
</Record>
```

In the above output, FOO is not base64 encoded; the XmlDoc contains a null character, which is displayed by the Print method using a character reference (`�`). This may be useful for visually inspecting the contents of the XmlDoc, again noting that such a document is not, strictly speaking, conformant to the XML Recommendation.

See the description of the `AttributeValue=` argument in [“LoadFromRecord subroutine in XmlDoc and XmlNode classes” on page 14](#) for a list of all conditions which force base64 encoding of the 'field' element.

... other arguments ... See [“LoadFromRecord subroutine in XmlDoc and XmlNode classes” on page 14](#) for a discussion of all the other arguments of `NewFromRecord`.

See [“LoadFromRecord subroutine in XmlDoc and XmlNode classes” on page 14](#) for a discussion of extracting the contents of the current record into an XmlDoc.

Note that this method was actually introduced in version 7.6 of the *Sirius Mods*, although the `AllowNull=` argument was not provided until version 7.7 of the *Sirius Mods*.

3.3 LoadFromRecord subroutine in XmlDoc and XmlNode classes

This subroutine adds to an XmlDoc object a subtree that contains the fields and fieldgroups from the current record. This record extraction is the same operation that is performed by the NewFromRecord function (“NewFromRecord shared function in XmlDoc class” on page 11) and by the ToXmlDoc function in the Record class.

Whether to use ToXmlDoc, NewFromRecord, or LoadFromRecord depends on what is most convenient for your application. If you are already using a Record object that references the desired record, using ToXmlDoc may be more convenient; if not, then either NewFromRecord or LoadFromRecord (both of which require that the method be contained in a “record loop”) may be more convenient. You must use LoadFromRecord if you want to add the record's content as a subtree to a non-empty XmlDoc; in other cases, the NewFromRecord “factory method” may be your choice.

Since NewFromRecord and ToXmlDoc create new XmlDoc objects, they have the AllowNull= argument for setting the created XmlDoc's AllowNull property; LoadFromRecord does not have the AllowNull= argument.

As stated, both NewFromRecord and LoadFromRecord must be contained in a “record loop”, for example, an FRN block, and they may not be invoked within a fieldgroup context.

Except for these considerations, ToXmlDoc, NewFromRecord, and LoadFromRecord all perform the same operation and have the same arguments. In our discussion of the “extract from record to XmlDoc” operation, we will generally use LoadFromRecord, except where one of these considerations is relevant to the discussion.

The arguments to LoadFromRecord are:

```
%xmlNr:LoadFromRecord( -  
    [AttributeValues=bool] -  
    [, AttributeNames=bool] -  
    [, NamesToLower=bool] -  
    [, AllowUnreversible=bool] -  
    [, CodepageTable=bool] )
```

Where:

%xmlNr LoadFromRecord is in both the XmlDoc and XmlNode classes.

When the method object is an XmlDoc, or refers to the Root node of an XmlDoc:

- The XmlDoc must not contain any nodes except the Root node.

- A 'Record' element is added as the top level element of the XmlDocument. Children are added to the 'Record' element, representing the outer fields and fieldgroups of the record.

When the method object is an XmlNode not referring to the Root node: of an XmlDocument:

- The node must be an Element node.
- Children are added to that element, representing the outer fields and fieldgroups of the record.
- See [“AddToRecord subroutine in XmlDocument class” on page 17](#) for some examples of extracting multiple records into a single XmlDocument.

See [“Structure of XmlDocument for AddToRecord” on page 23](#) for a description of the XmlDocument created by LoadFromRecord.

AttributeValues=bool

This name required argument is a Boolean value that indicates whether a field value will be stored as “XML text” or as an XML attribute (belonging to its field, which is an XmlDocument element).

For example, `<APSUBUND>COMM</APSUBUND>` is text format, and `<APSUBUND value="COMM"/>` is attribute value format.

The default value is `False`, which produces text format. In this format, the value of a field will be converted to base64 in the XmlDocument if the field contains a byte that is:

- Equal to X'00', if the AllowNull property of the XmlDocument is `False`.
- Between X'00' and X'3F'.
- Not translatable from EBCDIC to Unicode, using either the standard Unicode translation table or the base codepage translation table, as determined by the `CodepageTable=` argument.
- Not invertible when translating from EBCDIC to Unicode and back to EBCDIC using the standard Unicode translation table, if the `CodepageTable=` argument is `False`.

This argument must be `False` if the `XmlDoc` is to be used as the method object of the `AddToRecord` subroutine (see [“AddToRecord subroutine in XmlDoc class” on page 17](#)).

`AttributeNames=bool`

This name required argument is a Boolean value that indicates whether each field name is to be stored in the `XmlDoc` as an element name or as the value of a 'name' attribute.

For example, `<APSUBUND>COMM</APSUBUND>` is element-name format, and the following is name-as-attribute format:

```
<field name="APSUBUND">
  COMM
</field>
```

The default value as of *Sirius Mods* version 7.6 (and maintenance back to version 7.3) is `True`, which produces name-as-attribute format. Formerly, the default value was `False`.

The name-as-attribute format from the `True` option is better suited to operations on the `XmlDoc`, particularly a record copying operation. The element-name format from the `False` option produces more compact output when the `XmlDoc` is serialized.

This argument must be `True` if the `XmlDoc` is to be used as the method object of the `AddToRecord` subroutine (see [“AddToRecord subroutine in XmlDoc class” on page 17](#)).

`NamesToLower=bool`

This name required argument is a Boolean value that indicates whether field names are stored in all lowercase characters. The default value is `False`, which does not translate uppercase name characters to lowercase.

If the `XmlDoc` is to be used for record copying, this argument should probably be `False`.

`AllowUnreversible=bool`

This name required argument is a Boolean value that indicates whether a request is cancelled if a field name would be changed irreversibly by lowercasing or by replacing with a period the characters that would be invalid in an XML document.

The default value is `False`, which allows request cancellation to alert you about irreversible field names.

If the XmlDoc is to be used for record copying, this argument should probably be `False`.

CodepageTable=bool

This name required argument is a Boolean value; if `True`, the translations defined by the **base** Unicode codepage are used when translating from EBCDIC to Unicode for storing in the XmlDoc.

This argument is for the unusual case where you anticipate that the XML document is to be used later by the `AddToRecord` subroutine (see [“AddToRecord subroutine in XmlDoc class”](#)), and the standard Unicode translation tables in place when `AddToRecord` is invoked may differ from those in place when the record was copied to the XmlDoc.

The default value is `False`, which uses the standard Unicode translation tables, including any modifications specified in UNICOD Trans or UNICOD Map commands. The advantage of using `CodepageTable=False` is that it will allow you to readily modify the XmlDoc directly (that is, with the `AddElement` and `AddAttribute` methods). Those operations will use the standard Unicode translation tables; there is no way to perform them using the base codepage translation tables.

All fields are copied to the XmlDoc by `LoadFromRecord`.

See [“AddToRecord subroutine in XmlDoc class”](#) for an example of extracting a record to an XmlDoc for record copying, and see the `ToXmlDoc` method in the `Record` class for other examples using the method arguments.

This method was actually introduced in version 7.6 of the *Sirius Mods*, but the `XmlNode` class `LoadFromRecord` implementation prior to version 7.8 required that the XmlDoc be empty.

3.4 AddToRecord subroutine in XmlDoc class

This subroutine adds to the current record the fields and/or fieldgroups that are contained in the method XmlDoc object according to the structure created by the `LoadFromRecord` subroutine (see [“LoadFromRecord subroutine in XmlDoc and XmlNode classes” on page 14](#)). `AddToRecord` must be contained within a “record loop” (for example, an FRN block), and must not be contained in a fieldgroup context.

The arguments to `AddToRecord` are:

```
%doc:AddToRecord( -  
  [DisableFieldConstraints=bool] -  
  [, CopyIDs=bool] -  
  [, IgnoreUndefinedFields=bool] )
```

Where

%doc The method object is an `XmlDoc` whose structure conforms to that created by the `LoadFromRecord` subroutine; see [“Structure of `XmlDoc` for `AddToRecord`” on page 23](#).

`DisableFieldConstraints=bool`

This name required argument is a Boolean value; if `True`, then various field constraints (such as `LENGTH-EQ`, which was introduced in *Model 204 V7R2*) are not checked when the fields are added to the record.

The default is `False`.

`CopyIDs=bool`

This name required argument is a Boolean value; if `True`, then the fieldgroup IDs are copied, if possible, from the `XmlDoc` (stored as the 'groupID' attribute of 'fieldgroup' elements) to the record, and the 'maxGroupID' attribute value of the 'Record' element is also used as the maximum fieldgroup ID in the record, if possible. On any given 'fieldgroup' element, a 'groupID' attribute may be 0 or missing or not greater than the current maximum fieldgroup ID; in any of these cases, a new fieldgroup ID is generated for the fieldgroup occurrence.

If `False`, new fieldgroup IDs are generated.

The default is `False`.

`IgnoreUndefinedFields=bool`

This name required argument is a Boolean value; if `True`, then:

- If a field in the `XmlDoc` is not defined in the current file, the field is ignored.
- If a fieldgroup in the `XmlDoc` is not defined in the current file, the fieldgroup, and all descendants of the fieldgroup element, are ignored.

If `False`, any field or fieldgroup in the `XmlDoc` that is not defined in the current file throws an `AddToRecordError` exception.

The default is `False`.

The AddToRecord subroutine can throw an AddToRecordError exception; see [“AddToRecordError exception class” on page 25](#).

Fields in the XmlDoc that are defined as CAT or CTO are ignored by AddToRecord.

Any fieldgroup update tracking fields in a fieldgroup are automatically set when that fieldgroup is created by AddToRecord, but no other updated tracking fields are automatically set by the updates performed by AddToRecord.

Any fields in the XmlDoc that are defined as update tracking fields are copied to the record. See [“Removing update tracking fields from an XmlDoc” on page 22](#) for an example of removing update tracking fields from AddToRecord's XmlDoc; generally this will not be needed, but it may be useful in certain situations.

The basic approach to copying a record from one file to another is:

```
In SRCFILE FRN %source
  %doc = %doc:NewFromRecord
End For
In TARGFILE Store Record
End Store
%targ = $CurRec
In TARGFILE FRN %targ
  %doc:AddToRecord
End For
```

Some additional examples of record copying are shown in [“Copying from multiple source records”](#).

Note that this method was actually introduced in version 7.6 of the *Sirius Mods*, although until version 7.8 of the *Sirius Mods* an exception was thrown if the version of *Model 204* in use was older than V7R2.

3.4.1 Copying from multiple source records

Since the LoadFromRecord subroutine ([“LoadFromRecord subroutine in XmlDoc and XmlNode classes” on page 14](#)) can extract from a record into a subtree of an Element node in an XmlDoc, you can use it, together with AddToRecord, to combine multiple source records into one target record.

As a simple example, you can put the fields from two records “side by side” into a target record:

```
FRN %x
  %doc = %doc:NewFromRecord
End For
%top = %doc:SelectSingleNode('*')
FRN %y
  %top:LoadFromRecord
End For
Store Record
End Store
%rn = $CurRec
FRN %rn
  %doc:AddToRecord
End For
```

The XmlDoc that is input to AddToRecord would have a structure similar to the following:

```
<Record ...>
  <field ...>
    ... first field from record %x
  </field>
  ...
  <field ...>
    ... first field from record %y
  </field>
  ...
</Record>
```

Of course, you could also accomplish this particular objective by using two XmlDocs:

```
FRN %x
  %doc1 = %doc1:NewFromRecord
End For
FRN %y
  %doc2 = %doc2:NewFromRecord
End For
Store Record
End Store
%targ = $CurRec
FRN %targ
  %doc1:AddToRecord
  %doc2:AddToRecord
End For
```

You can also use LoadFromRecord to modify the fieldgroup structure within a record

(using V7R2 of *Model 204*). For example, you could take the “outer” fields of one record and move them to be fieldgroup members in the target record:

```
In SRCFILE FRN %x
  %doc = %doc:NewFromRecord
End For
%fg = %doc:SelectSingleNode('*/*fieldgroup[@name="GRP"]')
In SRCFILE FRN %y
  %fg:LoadFromRecord
End For
In TARGFILE Store Record
End Store
%rn = $CurRec
In TARGFILE FRN %rn
  %doc:AddToRecord
  PAI
End For
```

The use of a separate source and target file above (which in general is a typical usage of the record copying methods) is more or less required here, because the fields in SRCFILE are defined as outer fields, but in TARGFILE they are defined as members of fieldgroup GRP (or they would need to be “FIELDGROUP *” fields). So, for example, definitions for SRCFILE might include:

```
DEFINE FIELD F00
DEFINE FIELDGROUP GRP
```

and definitions for TARGFILE might include:

```
DEFINE FIELDGROUP GRP
DEFINE FIELD F00 WITH FIELDGROUP GRP
```

And the XmlDoc that is input to the above AddToRecord subroutine may look like:

```
<Record ...>
  <fieldgroup name="GRP" ...>
    <field name="F00">
      value of foo
    </field>
  </fieldgroup>
</Record>
```

And the corresponding output of the above PAI would be:

```
\GRP = 1
  F00 = value of foo
/GRP = 1
```

3.4.2 Removing update tracking fields from an XmlDoc

In the simple situation, the record copying operation provided by LoadFromRecord and AddToRecord produces a copy of all of the record's fields, including update tracking fields (such as UPDATE-TIME fields). However, in some circumstances, you may not want some of the update tracking fields to be propagated by AddToRecord. This can readily be accomplished by using DeleteSubtree to remove the tracking fields you want to suppress (of course, DeleteSubtree could be used to remove other fields and/or fieldgroups, as well).

Consider an example in which AddToRecord is being used to add fields to a record that already has fields stored in it:

```
IN ?&SOURCE DEFINE FIELD REC.UPD WITH UPDATE-TIME
IN ?&SOURCE DEFINE FIELD FOO
IN ?&TARGET DEFINE FIELD REC.UPD WITH UPDATE-TIME
IN ?&TARGET DEFINE FIELD FOO
...
Do the following on Monday:
  In ?&SOURCE Store Record
    FOO = 'Record stored on Monday'
  End Store
  %rn = $CurRec
  In ?&SOURCE FRN %rn
    %doc = %doc:NewFromRecord
  End For
  %serial = %doc:Serial
  In LOG Store Record
    RECORD.XML = %serial
  End Store
...
Do the following on Tuesday:
  In LOG For 1 Record
    %doc:LoadXml(RECORD.XML)
  End For
  In ?&TARGET Store Record
    FOO = 'Record stored on Tuesday'
  End Store
  %rn = $CurRec
  In ?&TARGET FRN %rn
    %doc:AddToRecord
  End For
```

In this scenario, GRP.UPD would be set to Monday, overlaying the more recent (and probably desired) value which was set on Tuesday. To prevent this overlay, you can insert the following statement after %doc:LoadXml(RECORD.XML):

```
%doc>DeleteSubtree('*/fieldgroup/field[@name="GRP.UPD"]')
```

3.4.3 Structure of XmlDoc for AddToRecord

The method object of the AddToRecord subroutine is an XmlDoc whose structure conforms to that created by the LoadFromRecord subroutine. You can also create an XmlDoc to be used for AddToRecord; the rules for the XmlDoc are described here. The outline of the XmlDoc is:

```
<Record [version="1"] [codepage="--codepage name--"]
  [maxGroupID="--fieldgroup counter--"]
  [file="---file name---"] [number="---record number---"]>
  <field name="---field name---">
    ---field value---
  </field>
  <field name="---field name---" encoding="base64">
    ---base64 encoded field value---
  </field>
  <fieldgroup name="---fieldgroup name---"
    [groupID="---fieldgroup ID---"]>
    <field ...
    <fieldgroup ...
    ...
  </fieldgroup>
  ...
</Record>
```

The requirements for the XmlDoc are:

Comments and PIs

Comment and PI nodes are allowed anywhere in the XmlDoc.

Elements in non-null namespaces

In addition to the elements mentioned below for the various elements contained in the XmlDoc, the 'Record' and 'fieldgroup' elements may have any additional child elements which are in any non-null namespace. These additional elements are ignored. For example, the following is a valid AddToRecord XmlDoc representing an empty record:

```
<Record>
  <foo:bar xmlns:foo="u:xyz"/>
</Record>
```

However, the only null-namespace child elements permitted of elements in the null namespace are those described for each element type below. For example, the following is an invalid AddToRecord XmlDoc:

```
<Record>
  <foo/>
</Record>
```

Attributes in non-null namespaces

In addition to the attributes mentioned below for the various elements contained in the XmlDoc, those elements may contain any additional attributes which are in any non-null namespace. These additional attributes are ignored. For example, the following is a valid AddToRecord XmlDoc representing an empty record:

```
<Record foo:bar="x" xmlns:foo="u:xyz"/>
```

Of course, elements in non-null namespaces may have any attributes. The only attributes permitted in elements in the null namespace are those described below. For example, the following is an invalid AddToRecord XmlDoc:

```
<Record foo="x"/>
```

Record element

The top level element of the XmlDoc must be named 'Record'; all of its attributes are optional and are described below. The element children of the 'Record' element are optional; they may be:

- field
- fieldgroup

The 'Record' element may not have a Text child node.

version attribute of Record element

If present, this attribute must have the numeric value '1'.

codepage

If present, this attribute contains the name of the codepage whose base translation tables are used for converting the Unicode field names and field values (if not base 64 encoded) in the XmlDoc to EBCDIC.

maxGroupID attribute of Record element

If present, this attribute contains the value of the maximum fieldgroup ID to be set (if greater than or equal to all of the fieldgroup IDs stored in the record) in the added record, if the `CopyIDs=True` argument is provided. This must be a non-negative integer value.

file attribute of Record element

This attribute is ignored.

number attribute of Record element

If present, this must either be "-1", signifying that the input record number is not known, or must be a non-negative integer value. As noted in ["AddToRecord constraint on 'number' attribute" on page 39](#), this represents a compatibility issue.

field element

The 'field' element may be a child of either the 'Record' or 'fieldgroup' element; it contains a field occurrence. It must have a 'name' attribute, and may have an 'encoding' attribute. It may have one Text child, which is either the value of the occurrence or, if the 'encoding' attribute is present (with the value 'base64'), is the base 64 encoded value of the occurrence.

name attribute of field element

This attribute is required; it is the name of the field.

encoding attribute of field element

This attribute is optional; if present, it must have the value 'base64', indicating that the Text child of the 'field' element is the base 64 encoding of the field value.

fieldgroup element

The 'fieldgroup' element may be a child of either the 'Record' or 'fieldgroup' element; it is the top of a subtree representing a fieldgroup occurrence. The element children of the 'fieldgroup' element are optional; they may be:

- field
- fieldgroup

The fieldgroup element may not have a Text child node. It must have a 'name' attribute, and may have a 'groupID' attribute.

name attribute of fieldgroup element

This attribute is required; it is the name of the fieldgroup.

groupID attribute of fieldgroup element

If present, this attribute contains the value of the fieldgroup ID to be set (if possible) for the fieldgroup occurrence in the added record, if the `CopyIDs=True` argument is provided.

3.4.4 **AddToRecordError exception class**

The members of the AddToRecordError exception class are described below. Except for the constructor, `New`, all class members are read-only properties:

Reason An enumeration of type AddToRecordErrorReason. The possible values are:

InvalidNode A node in the XmlDoc does not conform to the structure as created by the LoadFromRecord subroutine.

UntranslatableFieldName

A field name in the XmlDoc is not translatable to EBCDIC.

UntranslatableFieldgroupName

A fieldgroup name in the XmlDoc is not translatable to EBCDIC.

UntranslatableValue

A field value in the XmlDoc is not translatable to EBCDIC.

InvalidBase64

A string used for the base64 encoding of a field in the XmlDoc is not a valid base64 string.

FieldNameTooLong

A field name in the XmlDoc is longer than 255 characters.

FieldgroupNameTooLong

A fieldgroup name in the XmlDoc is longer than 255 characters.

ValueTooLong

The value of a field in the XmlDoc that is not defined as a LOB field in the current file is longer than 255 characters or is longer than the defined LEN attribute, if the field is a fixed OCCURS field.

UnknownFieldName

A field name in the XmlDoc is not defined in the current file.

UnknownFieldgroupName

A fieldgroup name in the XmlDoc is not defined in the current file.

ExpectedField

A field name in the XmlDoc is defined as a fieldgroup in the current file.

ExpectedFieldgroup

A fieldgroup name in the XmlDoc is defined as a field in the current file.

ErrorAddingField

An error occurred adding a field, such as a violation of a field constraint.

ErrorAddingFieldgroup

An error occurred adding a fieldgroup, such as a file full condition.

ErrorObtainingRecord

AddToRecord was unable to lock the record in exclusive mode.

InvalidFieldgroupID

A fieldgroup ID in the XmlDoc is not numeric.

InvalidCodepage

The codepage name specified on the 'codepage' attribute of the 'Record' element is not a known codepage name.

ErrorAddingMaxFieldgroupID

The attempt to set the fieldgroup ID counter in the record failed; this is a very unusual condition.

InsufficientStorageForLOB

STBL, VTBL, or User Buffer storage was unavailable. The `Description` property indicates which of these is applicable.

InvalidVersion

Invalid value of the 'version' attribute of the 'Record' element; the only allowed value is 1.

InvalidInputRecordNumber

Invalid value of the 'number' attribute of the 'Record' element; it must either be "-1" or a non-negative integer.

Description

A message that explains the error.

UntranslatableHexValue

If the Reason indicates that a string in the XmlDoc is not translatable to EBCDIC, this contains the hexadecimal representation of the Unicode codepoint which is not translatable.

FieldOrFieldgroupName

If the error involves a field or fieldgroup (for example, if `Reason = ErrorAddingField`), this is the field or fieldgroup name.

NodeName

If the error involves a named node in the XmlDoc (for example, some cases when `Reason = InvalidNode`), this is the name of the node.

NodeType

If the error involves a node in the XmlDoc (for example, all cases when `Reason = InvalidNode`), this an XmlNodeType enumeration value of the type of the node.

Value

If the error involves a value in the XmlDoc (for example, when `Reason = InvalidBase64`), this is the value that is in error (actually, up to 255 bytes of the value).

InputRecordNumber

The value of the 'number' attribute of the 'Record' element in the XmlDoc.

New

The constructor for the class, `New` lets you set values for each member of the class.

```
%ex = New ( Reason = reasonEnum           -  
           [, Description = string]       -  
           [, UntranslatableHexValue = hexString] -  
           [, FieldOrFieldgroupName = string] -  
           [, NodeName = string]         -  
           [, NodeType = xmlNodeTypeEnum]  -  
           [, Value = string]            -  
           [, InputRecordNumber = number])
```

The Reason argument is required; all other arguments are optional, name required. The default value of InputRecordNumber is -1; all other default values are the null string or the Null object, as appropriate.

Note that this class was actually introduced in version 7.6 of the *Sirius Mods*.

3.5 MoveNamespace argument for AddTopElement

The following optional, name required argument has been added to the AddTopElement function in the XmlDoc class:

```
MoveNamespace=boolean
```

If MoveNamespace=True and a URI argument (with a non-null string) is provided, then if the "old" top element contains a namespace declaration which is the same as the declaration inserted due to the URI argument, the declaration is deleted from the old top element.

Note: You should use MoveNamespace=True only when some consumer of the XML document cares about not having the redundant declaration. This would be an unusual situation, because the information content of the document does not change. Using MoveNamespace=True might trigger a scan of the entire XmlDoc, which AddTopElement does not normally do.

3.6 DeleteTopElement subroutine in XmlDocument class

The syntax for DeleteTopElement is:

```
doc.DeleteTopElement
```

This subroutine removes the top Element from an XmlDocument, and make its children be the “middle siblings” of the left and right siblings of the deleted Element.

- The deleted element may not have more than one Element child.
- The deleted element may not have a Text child.

In common usage, the Element child of the deleted Element will become the new top Element of the XmlDocument. DeleteTopElement is a convenient, better-performing alternative to using AddSubtree to copy all but the top Element (or in multiple iterations, removing successive top Elements) from an XmlDocument.

For example, you can use it to remove the SOAP wrappers from an XmlDocument; if the contents of %xdoc are:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <data>
      ...
    </data>
  </soap:Body>
</soap:Envelope>
```

Then the following User Language fragment:

```
%xdoc.DeleteTopElement
%xdoc.DeleteTopElement
```

Results in the following contents of %xdoc:

```
<data>
  ...
</data>
```

The remaining considerations for using DeleteTopElement concern the occurrence of namespace declarations in the XmlDocument; there are a few details to explain how they are manipulated, and to explain how the performance of DeleteTopElement is affected.

As shown in the “SOAP wrapper removal” example above, DeleteTopElement will remove from the XmlDocument any namespace declarations (“xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"”) which **are not** referenced

after removing the top Element. Note that a variant on the above example can occur using a default namespace declaration:

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <data xmlns="">
      ...
    </data>
  </Body>
</Envelope>
```

Then the following User Language fragment:

```
%xdoc:DeleteTopElement
%xdoc:DeleteTopElement
```

Results in the following contents of %xdoc:

```
<data xmlns="">
  ...
</data>
```

As seen in this example, the null namespace declaration is not removed, even though it is not, strictly speaking, needed.

In addition, DeleteTopElement will move, to the new top Element, any namespace declarations which **do** continue to be referenced after the top Element deletion:

```
%n = %d:AddElement('foo', , 'u:uri')
%n:AddElement('bar', , 'u:uri')
%d:Print
%d:DeleteTopElement
Print 'After deletion:'
%d:Print
```

The result of the above User Language fragment is:

```
<foo xmlns="u:uri">
  <bar/>
</foo>
After deletion:
<bar xmlns="u:uri"/>
```

Regarding the performance of DeleteTopElement, it should always be significantly better than an approach using AddSubtree, and in many cases, it is constant, regardless of the size of the XmlDocument. For a detailed explanation of when the cost of DeleteTopElement might vary with the size of the XmlDocument, there are three reasons that the descendants of the new top Element need to be recursively visited:

1. Fixing references to moved namespace declaration

As mentioned above, one or more namespace declarations can be moved from the deleted element to the new top element. Having done that, any references to the moved namespace must be fixed up. A count is kept to the total references to moved namespace declarations; when this goes to zero there is no more need to do this. So, for example:

```
<p:oldTop xmlns:p="p:uri">
  <p:newTop>
    <child/>
  </p:newTop>
</p:oldTop>
```

The p:newTop element needs to have its namespace reference fixed, but that is the only reference to it, so there is no need to visit child.

2. Fixing pointers to a deleted namespace declaration

When a namespace declaration is deleted, it may be pointed to by other namespace declarations (there is a graph of them in the XmlDoc); these pointers need to be fixed up to point to the first non-deleted declaration following the deleted one. This is not needed by the descendants of any element which has a namespace declaration (in the original XmlDoc). So, for example:

```
<p:oldTop xmlns:p="p:uri">
  <newTop>
    <q:child xmlns:q="q:uri">
      <grandchild/>
    </q:child>
  </newTop>
</p:oldTop>
```

Both newTop and q:child need to have their namespace pointers fixed up, but since q:child has a namespace declaration, grandchild doesn't need to be fixed up, so it doesn't need to be visited.

3. Turning off "ancestor may have non-null default namespace declaration"

In order to make certain namespace operations faster, there is a "state" maintained at XmlDoc nodes, indicating that a non-null default namespace declaration may occur on an ancestor. This state is fixed up if DeleteTopElement deletes a non-null namespace declaration (as happens in the second "SOAP wrapper removal" example above, the one using a default namespace). The need to set this state is eliminated in descendants of an element which has a default namespace

declaration (either null or not). So, for example:

```
<oldTop xmlns="p:uri">
  <newTop>
    <q:child xmlns="q:uri">
      <grandchild/>
    </q:child>
  </newTop>
</p:oldTop>
```

The state needs to be fixed at newTop, but since q:child has a default namespace declaration, the state at grandchild doesn't need to be fixed up, so it doesn't need to be visited.

CHAPTER 4 *Fast/Reload*

The following are new or changed features in *Fast/Reload*:

4.1 Warning messages for some DV, STORE-x, and repeatability changes

V7R2 of *Model 204* introduced several new field attributes. Reorganizing a file and changing some of these attributes may create differences in field processing which may not be obvious at first glance.

To highlight this, *Fast/Reload* will issue a warning message (MSIR.1037) for each attribute change which may have some of these subtle effects.

For the most part, applications using the reorganized file will continue to work as before.

One source of possible change in application behavior involves use of the following User Language constructs:

- the `Is Present` test;
- the count returned by a `Count Occurrences Of` statement;
- the number of iterations of a `For Each Occurrence Of` loop.

All three of the above constructs have equivalent exposure to the changes in field attributes, even though the examples below only use `Is Present`, for brevity.

The other source of possible change in behavior is:

- the value returned when referencing a physically absent field occurrence which was AT-MOST-ONE prior to the reorganization.

This is shown in the examples using the `Print` statement, although of course the change could be in any kind of reference to the field value.

When changing field attributes as part of reorganizing a file, you should understand the impact of the change; to help in this, the conditions which cause warning messages are listed in the following subsections, along with a very brief example of a possible change in behavior.

Notes:

- Other changes in these field attributes do not cause the kinds of problems discussed here. For example, you can change the DEFAULT-VALUE, STORE-DEFAULT, and STORE-NONE attributes of an EXACTLY-ONE field, without any consequences about User Language application behavior.
- Although the examples below all involve fields defined within fieldgroups, the potential problems, and the warning messages issued by *Fast/Reload*, apply equally to fields defined outside fieldgroups.

In these examples, assume you are reorganizing, with UAI/LAI, a file with the following “old” definitions:

```
DEFINE FIELD KEY WITH ORDERED
DEFINE FIELDGROUP GRP
DEFINE FIELD EXO.OLD (FG GRP)
DEFINE FIELD REPT.OLD.MISS (REPT FG GRP)
DEFINE FIELD REPT.OLD.XX (REPT FG GRP)
DEFINE FIELD REPT.OLD.NUL (REPT FG GRP)
DEFINE FIELD AMO.OLD.DF_YY (ONE DV 'YY' FG GRP)
DEFINE FIELD AMO.OLD.SD_LIT (ONE DV 'AA' SD LIT FG GRP)
```

And assume that a record has been stored as follows:

```
Store Record
  KEY = 'Some unique value'
Then Continue
  Add Fieldgroup GRP
    REPT.OLD.XX = 'xx'
    REPT.OLD.NUL = ''
    AMO.OLD.SD_LIT = 'AA'
  End Add
End Store
```

Each example contains a field definition, used in the LAI step, to highlight one condition, and contains a User Language fragment which is assumed to be contained within the following:

```
FR Where KEY = 'Some unique value'
  For Fieldgroup GRP
    ... example fragment
  End For
End For
```

The conditions are shown in the following subsections.

4.1.1 EXONE -> non-EXONE with STORE-x NONE

Case: Changing a field which was EXACTLY-ONE in the UAI to AT-MOST-ONE or REPEATABLE, with either STORE-DEFAULT NONE or STORE-NONE NONE.

Changed definition in LAI:

```
DEFINE FIELD EXO.OLD (REPT SN NONE FG GRP)
```

User Language fragment:

```
If EXO.OLD Is Present Then Print 'Present'  
Else Print 'Not Present'; End If
```

Result prior to reorg:

```
Present
```

Result after reorg:

```
Not Present
```

4.1.2 REPT -> EXONE

Case: Changing a field which was REPEATABLE in the UAI to EXACTLY-ONE.

There is one exception to this, which does not produce a warning - if the field was not a member of a fieldgroup in the UAI, and is a member of a fieldgroup in the LAI. Note that the “out of synch” check of this “collecting loose fields” LAI feature ensures there is no exposure to a change in User Language behavior when using such a field.

Changed definition in LAI:

```
DEFINE FIELD REPT.OLD.MISS (EXONE FG GRP)
```

User Language fragment:

```
If REPT.OLD.MISS Is Present Then Print 'Present'  
Else Print 'Not Present'; End If
```

Result prior to reorg:

```
Not Present
```

Result after reorg:

```
Present
```

4.1.3 REPT -> ONE STORE-x NONE

Case: Changing a field which was REPEATABLE in the UAI to AT-MOST-ONE with either STORE-DEFAULT NONE or STORE-NONE NONE.

Changed definition in LAI:

```
DEFINE FIELD REPT.OLD.XX (ONE DV 'xx' SD NONE FG GRP)
```

User Language fragment:

```
If REPT.OLD.XX Is Present Then Print 'Present'  
Else Print 'Not Present'; End If
```

Result prior to reorg:

```
Present
```

Result after reorg:

```
Not Present
```

4.1.4 ONE -> ONE with changed DV

Case: Changing a field which was AT-MOST-ONE in the UAI to have a different DEFAULT-VALUE in the LAI.

Changed definition in LAI:

```
DEFINE FIELD AMO.OLD.DF_YY (ONE DV 'ZZ' FG GRP)
```

User Language fragment:

```
Print AMO.OLD.DF_YY
```

Result prior to reorg:

```
YY
```

Result after reorg:

```
ZZ
```

4.1.5 ONE -> EXONE

Case: Changing a field which was AT-MOST-ONE in the UAI to EXACTLY-ONE in the LAI.

Changed definition in LAI:

```
DEFINE FIELD AMO.OLD.DF_YY (EXONE FG GRP)
```

User Language fragment:

```
If AMO.OLD.DF_YY Is Present Then Print 'Present'  
Else Print 'Not Present'; End If
```

Result prior to reorg:

```
Not Present
```

Result after reorg:

```
Present
```

4.1.6 ONE with DV -> REPT

Case: Changing a field which was AT-MOST-ONE with DEFAULT-VALUE in the UAI to REPEATABLE in the LAI.

Changed definition in LAI:

```
DEFINE FIELD AMO.OLD.DF_YY (REPT FG GRP)
```

User Language fragment:

```
Print '>' With AMO.OLD.DF_YY With '<'
```

Result prior to reorg:

```
>YY<
```

Result after reorg:

```
><
```

4.1.7 ONE SD LIT/ALL -> SD NONE

Case: Changing a field which was AT-MOST-ONE with STORE-DEFAULT LIT or ALL in the UAI to STORE-DEFAULT NONE in the LAI.

Changed definition in LAI:

```
DEFINE FIELD AMO.OLD.SD_LIT (ONE DV 'AA' SD NONE FG GRP)
```

User Language fragment:

```
If AMO.OLD.SD_LIT Is Present Then Print 'Present'  
Else Print 'Not Present'; End If
```

Result prior to reorg:

Present

Result after reorg:

Not present

4.1.8 Non-EXONE SN LIT/ALL -> SN NONE

Case: Changing a field which was AT-MOST-ONE or REPEATABLE, with STORE-NULL LIT or ALL, in the UAI to STORE-NULL NONE in the LAI.

There is one exception to this, which does not produce a warning - if the field was not a member of a fieldgroup in the UAI, and is a member of a fieldgroup in the LAI. Note that the “out of synch” check of this “collecting loose fields” LAI feature ensures there is no exposure to a change in User Language behavior when using such a field.

Changed definition in LAI:

```
DEFINE FIELD REPT.OLD.NUL (REPT SN NONE FG GRP)
```

User Language fragment:

```
If REPT.OLD.NUL Is Present Then Print 'Present'  
Else Print 'Not Present'; End If
```

Result prior to reorg:

Present

Result after reorg:

Not Present

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 previous generally available version (7.8).

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).

5.1 Backwards incompatibilities

Backwards incompatibilities are described per product in the following sections.

5.1.1 Janus SOAPXmlDoc API

The following backwards compatibility issues have been introduced in the *Janus SOAP XmlDoc API*.

5.1.1.1 AddToRecord constraint on 'number' attribute

As described in “[Structure of XmlDoc for AddToRecord](#)” on page 23, if the 'number' attribute of the 'Record' element in the input XmlDoc of the AddToRecord subroutine is present, it must be an integer greater than or equal to -1.

Previously, this attribute was ignored.

5.1.1.2 DefaultURI argument of AddSubtree

In some cases, an Element in a default namespace, which was added to the XmlDoc by a deserialization method, will not get the correct namespace URI when it is copied using the DefaultURI argument of the AddNamespace subroutine.

Note that this problem was also fixed (with the resulting incompatibility) in the version 7.7 *Sirius Mods* by maintenance supplied by ZAP77A4 on 17 August, 2010.

For example:

```
Text To %s1
<a:a xmlns:a="http://aaa" xmlns="http://ddd">
  <b:b xmlns:b="http://bbb">
    <c>123</c>
  </b:b>
</a:a>
End Text
%in:LoadXml(%s1)
%n Object XmlNode
%n = %in:SelectSingleNode('/*/*')
%out:AddSubtree(%n, DefaultURI='u:who')
```

Prior to fixing this problem, the above results in:

```
<b:b xmlns:b="http://bbb">
  <c xmlns="http://ddd">
    123
  </c>
</b:b>
```

The correct result, as produced by version 7.8 of the *Sirius Mods* or version 7.7 with with ZAP77A4 applied, is as follows (note the namespace for element 'c'):

```
<b:b xmlns:b="http://bbb">
  <c xmlns="u:who">
    123
  </c>
</b:b>
```

5.1.1.3 **Deserialization prohibit default namespace declaration with Namespace=None**

With the Namespace=None XmlDocument property, namespace declarations which bind a prefix are not allowed, for example:

```
<foo xmlns:p="http://p.com/>
```

The above has never been allowed, due to the prohibition against colons in XML names when Namespace=None.

However, previous versions of the *Sirius Mods* allowed deserialization of a default namespace declaration, for example:

```
<foo xmlns="http://p.com/>
```

The above was erroneously treated as if 'xmlns' were an attribute.

Deserialization of default namespace declarations is no longer allowed.

This fix was also introduced in version 7.7 of the *Sirius Mods*, via ZAP77B2.

5.2 Fixes in Sirius Mods 7.8 but not in 7.7

This section lists fixes to functionality existing in the *Sirius Mods* version 7.8 but which, due to the absence of customer problems, have not, as of the date of the release, been fixed in that version.

5.2.1 LoadSystemMethodInfo returning Unicode methods

The LoadSystemMethodInfo now returns information for Unicode intrinsic methods.

5.2.2 Fast/Reload LAI with various UAI SORT cases

The following two bugs have been fixed:

- Possible use of UAI SORT key as hashed file sort key.

For example, the above should not be allowed:

```
* Fast/Unload step:
UAI SORT FOO LENGTH 3 TRUNC
...
* Fast/Reload step:
CREATE FILE (NOFORMAT) QAWORK
PARAMETER FILEORG 8
END
OPEN QAWORK
*UPDATE
IN QAWORK INITIALIZE
HASH FOO
FILELOAD -1,-1,0,30000000,10000,10000,10000,32
LAI
```

The above (incorrectly) completes without any indication of error.

- A spurious error ("TAPEI format error" or unknown field) is issued if the first UAI SORT item is a FUEL %variable and the subsequent LAI is loaded into a non-sort/non-hash file. Note that the *Fast/Unload* UAI output file must also contain the correct value to fix this bug; with version 4.4 of *Fast/Unload*, ZAP4414 is required, with version 4.5, ZAP4518 is required.

5.3 Version corequisites

This section lists any restrictions on usage of various products (including *Sirius Mods* itself) that will be imposed by use of version 7.8 of *Sirius Mods*.

- There are no corequisites associated with *Sirius Mods* 7.8.