
Janus SOAP Reference Manual

Model 204 Interoperability Products



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>

February 14, 2004

© 2004 Sirius Software, Inc.

Deprecated Dollar-Function API

Proprietary Notices

The following products:

- *Janus SOAP*
- *Janus Sockets*
- *Janus TCP/IP Base*
- *Janus Web Server*
- *Sirius Functions*
- *Sirius Mods*

are proprietary products of Sirius Software, Inc.:

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

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

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

Model 204™ is a registered trademark of Computer Corporation of America.

Microsoft Internet Explorer™ and **Microsoft XML Core Services (MSXML)** are proprietary products of Microsoft Corporation.

Contents

Proprietary Notices	iii
Contents	v
Summary of Changes	ix
Sirius Mods Version 6.5	ix
Sirius Mods Version 6.4	ix
Sirius Mods Version 6.3	x
Sirius Mods Version 6.2	xi
Chapter 1: Janus SOAP Overview and Organization of This Manual	1
eXtensible Markup Language (XML)	1
Simple Object Access Protocol (SOAP)	3
Example SOAP request	3
Example SOAP response	4
XML Path Language (XPath)	4
Reading list	4
Organization of this manual	6
Chapter 2: XML	7
Example	8
XML syntax	9
Syntax of document, element, Attribute, Comment, PI	11
Char and Reference	12
Components of XMLDecl	13
Names and namespaces - objectives	13
Names and namespaces - syntax	14
Namespace URI for XPath prefixes	15
Well-formed documents and validation	15
Normalization during deserialization	16
Normalized line-end	17
Normalized attribute value	18
Chapter 3: Janus SOAP Concepts and Data Structures	19
XMLDoc structure, and node types	20
XMLDoc and nodelist data structures and states	22
XMLDoc states	23
Nodelist states	23
XPath, nodelists, \$XML_xxx example	24

Context node, [head of] argument result nodelist	25
Updating	27
Inserting nodes/copying subtrees	27
Deleting nodes	28
Transport: sending and receiving XML	29
Chapter 4: \$xml_xxx Functions	31
LONGSTRING arguments and result	31
\$xml_DOC2STR	32
\$xml_VAL	32
\$xml_NAME, \$xml_NS[INFO], other \$xml_xxx functions	32
\$xml_CHECK: Check validity of XMLDoc or nodelist ID	33
\$xml_COUNT: Get number of nodes in XPath result	34
\$xml_COUNT_NL: Get number of nodes in nodelist	35
\$xml_COP: Copy XMLDoc subtree	35
\$xml_DEL: Delete XMLDoc subtree	37
\$xml_DOC: Create XMLDoc	39
\$xml_DOC_GLOBAL and \$xml_DOC_SESSION: Session/global XMLDocs	39
\$xml_DOC_GLOBAL_DEL and \$xml_DOC_SESSION	42
\$xml_DOCID: Get ID of XMLDoc associated with nodelist	43
\$xml_DOC2STR: Serialize XMLDoc as UTF-8 string	44
\$xml_EXIST: Test for non-empty XPath result	47
\$xml_FREEL: FREE nodelist	48
\$xml_INFO: Get XMLDoc option setting	49
\$xml_INITDOC: EMPTY XMLDoc and FREE its nodelists	50
\$xml_INSATT: Insert attribute	50
Inserting xmlns attribute names	52
\$xml_INSCOM: Insert COMMENT node	53
\$xml_INSELT: Insert ELEMENT node	55
\$xml_INSPI: Insert PI node	59
\$xml_INSTEXT: Insert TEXT node or ELEMENT with TEXT child	61
\$xml_LEN: Get length of string-value	64
\$xml_NAME: Obtain the name of a node	66
\$xml_NL: Create nodelist	67
\$xml_NODE_SING: Select singleton nodelist from XMLDoc	68
\$xml_NODES: Select list of nodes within XMLDoc	69
\$xml_NS: Set namespace association for XPath prefix	71
\$xml_NSINFO: Get namespace association for XPath prefix	72
\$xml_PLG: Place nodes from XPath result onto a nodelist	73
\$xml_PRINT: Print XMLDoc or sub-tree	75
\$xml_RMV_NL: Remove one nodelist from another	78
\$xml_SER: Serialize sub-tree as string	78
\$xml_SET: Change XMLDoc option setting	80
NAMESPACE: set URI for prefix in XPath expressions	82
\$xml_STR2DOC: Deserialize string into XMLDoc	83
\$xml_TYPE: Obtain the type of a node	88

\$XML_VAL: Get string-value of node	89
\$WEB_XML_RECV: Receive XML document using Janus Web	90
\$WEB_XML_SEND: Send XML document using Janus Web	94
Chapter 5: Restrictions	97
NAMESPACE default and limitations	97
XML syntax support	97
XPath syntax support	98
XPath axis support	99
XPath node test support	99
XPath predicate support	99
[<n>] predicate	99
Location path within predicate	100
Existence test	100
Comparison test	100
XPath function support	100
[De]Serialization \$function support	100
Chapter 6: Messages	103
Appendix A: XPath	105
XPath operation	105
Axes	106
Node tests	106
Predicates	107
XPath syntax	108
XPath functions	110
XPath syntax cross-reference	111
Attributes are not children	112
Order of nodes in nodelist	112
XPath syntax used in version 6.4 Janus SOAP	113
Appendix B: References	115
Internet standards	115
Index	117

Summary of Changes

This section describes significant changes to the documentation. Usually, these changes correspond to enhancements made to the underlying product, although they might be major documentation enhancements.

Sirius Mods Version 6.5

The dollar-function API has been replaced with an easier to use object-oriented API. The new API is described in the current edition of this publication.

This edition is only intended to document the now-deprecated older API to assist in migration to the new API.

Sirius Mods Version 6.4

The following changes correspond to changes in *Janus SOAP* since version 6.3:

- New \$XML_CHECK function. See “[\\$XML_CHECK: Check validity of XMLDoc or nodelist ID](#)” on page 33.
- New \$XML_COP function. See “[\\$XML_COP: Copy XMLDoc subtree](#)” on page 35.
- New \$XML_DEL function. See “[\\$XML_DEL: Delete XMLDoc subtree](#)” on page 37. Note that if any deleted nodes had been referenced in a nodelist, they may not subsequently be referenced using a relative XPath expression.
- New \$XML_NODE_SING function. See “[\\$XML_NODE_SING: Select singleton nodelist from XMLDoc](#)” on page 68.
- Support for \$XML_PLC function. This \$function is now supported; see “[\\$XML_PLC: Place nodes from XPath result onto a nodelist](#)” on page 73.
- Support for \$XML_RMV_NL function. This \$function is now supported; see “[\\$XML_RMV_NL: Remove one nodelist from another](#)” on page 78.
- New \$XML_SER function. See “[\\$XML_SER: Serialize sub-tree as string](#)” on page 78 (this \$function was also introduced by maintenance to version 6.3 of *Janus SOAP*).
- Support for “[<n>]” predicate; see “[XPath predicate support](#)” on page 99.

- Support for location path within predicate, using “existence” or “comparison” test, as shown in the following two example XPath expressions:

```
/active/cust[invoice]/contact[fax]  
/inv/cust[@dt<"20030101"]/contact[state="MA"]
```

See “XPath predicate support” on page 99.

- New NOEMPTYELT option: see “\$XML_PRINT: Print XMLDoc or sub-tree” on page 75, “\$XML_SER: Serialize sub-tree as string” on page 78, and “\$XML_DOC2STR: Serialize XMLDoc as UTF-8 string” on page 44; also see “\$WEB_XML_SEND” in the *Janus Web Server Reference Manual*. (This option was also introduced by maintenance to version 6.3 of *Janus SOAP*.)
- A relative XPath expression may now be used with an XMLDoc ID (previously a nodelist ID was required).
- The “contx_n” argument is now always checked, if you provide one explicitly: it must either be 1, if it corresponds to an XMLDoc ID argument, or less than or equal to the size of its corresponding nodelist argument, and greater than zero.
- The initial version of *Janus SOAP* had a limit on the number of nodes in a nodelist; this has been removed.

Note

- We are planning for some pervasive changes for the Janus SOAP \$functions API in version 6.5; one of these will be to remove the use of a **contx_n** argument in \$functions that have an XPath expression argument.
- As a start in this direction, we have implemented \$XML_PLC and \$XML_DEL with an XPath expression argument, but no **contx_n** argument.
- Part of this change will also probably change the names of \$functions. We haven't finished the new nameset, but to “leave room” for new names, we've changed some names of \$functions formerly proposed, e.g., \$XML_PLACE is, for now, \$XML_PLC, and \$XML_REMOVE_NL is, for now, \$XML_RMV_NL.

Sirius Mods Version 6.3

- Session XMLDocs (“\$XML_DOC_GLOBAL and \$XML_DOC_SESSION: Session/global XMLDocs” on page 39 and “\$XML_DOC_GLOBAL_DEL and \$XML_DOC_SESSION” on page 42).

Sirius Mods Version 6.2

Base version of this manual.

Janus SOAP Overview and Organization of This Manual

Janus SOAP provides User Language programmers with facilities for processing eXtensible Markup Language (XML) documents. The purpose of *Janus SOAP* is to enable rich and automated Web services based on a shared and open Web infrastructure.

Most of the design of *Janus SOAP* is based on various standards, such as XML and XPath. Many sections in this manual refer to these and other standards, for example, “[Simple Object Access Protocol \(SOAP\)](#)” on page 3. However, it is important to recognize:

Janus SOAP enables you to process **any XML document, whether or not you are using SOAP messages and envelopes.**

The rest of this chapter mentions some of the standards employed by *Janus SOAP*, provides a reading list for these standards, and explains the organization of the other chapters in this manual.

1.1 eXtensible Markup Language (XML)

XML is a standard (endorsed by the World Wide Web Consortium, or W3C) which can be used for structuring almost any kind of data. Although the word “markup” reveals that the roots of XML are from document processing, and indeed the outermost object in XML is called a “document”, XML is ideally suited to structuring almost any kind of data which is exchanged between or within applications, particularly (although by no means exclusively) if they are communicating on a network.

The syntax of XML provides for hierarchical structuring of data (again, the outer object is called a document) into the principle object type called an **element**. Elements and the other components of an XML document are described in “[XML](#)” on page 7.

A key benefit that makes XML so extremely powerful is that there is no fixed vocabulary for XML documents. Every XML document can have its own set of names (within the rules for which characters can occur in a name). Additionally, there is no structure dictated for an XML document, except that there is a single top-level element and other elements must be completely contained within their parent elements. These facets allow XML to represent an extremely wide range of types of data very effectively.

An XML document can be considered an abstract object; when XML is used for interchange between applications, it is “serialized“, or transmitted, completely in character form. One advantage of this is that it is human-readable, and can be conveniently viewed using a generic XML editor; both of these can be huge benefits for debugging. Additionally, standard network protocols can be used to exchange documents between a wide variety of applications on a wide variety of platforms; the world-wide web has demonstrated that using characters as the basis for information interchange is extremely powerful and flexible.

Beyond these core properties which make XML very attractive for structuring data, it has become the basis for a large family of standards. Often these standards are referred to as the XML “family”, in part because they are managed by the XML Working Group of the W3C. Some of these important standards are XML Schema, XML Stylesheet Transformations, XML Query, and Web Services Description Language (WSDL); see <http://www.w3c.org> for more information on these and other standards related to XML.

To make these “higher-level” standards feasible, and to make it feasible for you to receive, create, access, modify, and transmit XML documents, an XML processor is needed, which can transform a character-format (serialized) document into an internal format, operate on an internal format, and produce a character-format from an internal format. These features are provided by *Janus SOAP*.

(There are also XML processors which don't keep a document in an internal format for the application. This may be useful in some circumstances, and accordingly *Janus SOAP*, in some version after 6.3, will provide “sub-documents” to a User Language application, for example, to handle the items within a SOAP envelope.)

Quoting from *XML in a Nutshell* (“Reading list” on page 4),

XML offers the tantalizing possibility of truly cross-platform, long term data formats. ... XML delivers portable data. In many ways, XML is the most portable ... format designed since the ASCII text file.

You can use XML strictly as an internal datastructure in your application, or in *Model 204* files, or with operating system files, or with other programs using some communication mechanism. The simple, character-based format of XML enhances such communication. You can communicate with the Web (HTTP), either as a server application, for example, using *Janus Web Server*, or making client XML requests, for example, using *Janus Sockets*. You can use native *Model 204* IODEV communication facilities, or *Model 204* MQ Series, or any facility that can send and receive streams of characters.

1.2 Simple Object Access Protocol (SOAP)

The Simple Object Access Protocol (SOAP) is a lightweight protocol that supports the exchange of structured information between Web-based applications. SOAP employs the eXtensible Markup Language (XML) to serialize the objects passed between applications. SOAP can be used in combination with a variety of existing firewall-friendly Internet protocols and formats including HTTP, SMTP, and MIME. SOAP supports a wide range of application paradigms from messaging systems to Remote Procedure Call (RPC).

SOAP is an excellent standard for information exchange between applications, and it is so good that it is the reason we have chosen the name *Janus SOAP*. It is important to recognize, however, that

Janus SOAP enables you to process **any XML document, whether or not you are using SOAP messages and envelopes.**

In fact, with version 6.4, although you can use the features of *Janus SOAP* to process formal SOAP messages, there are no features which are specially oriented toward that; the features are general for handling any kind of XML document. Future versions will add more and more functionality to incorporate the standard processing of SOAP messages, so that your application will only need to deal with the application-specific parts of the messages.

1.2.1 Example SOAP request

Here is an example SOAP message which is a request to a SOAP server:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-
  ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>EMC</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.2.2 Example SOAP response

Here is an example SOAP message which could be a response to the above message:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-
  ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.3 XML Path Language (XPath)

XPath is a language designed specifically to select nodes from an XML document. It is very powerful yet is based on familiar syntax that mimics an XML document's hierarchy. XPath is the general mechanism used in *Janus SOAP* for selecting one or more nodes to operate on. It is a key component of XSLT, XPointer, and XLink, and has a common foundation with XML Query.

An introduction to the use of XPath in *Janus SOAP* is provided in [“XPath, nodelists, \\$XML_xxx example” on page 24](#); a more complete description of XPath is contained in [“XPath” on page 105](#).

1.4 Reading list

As mentioned, SOAP is heavily oriented to the concepts and facilities defined by the XML standards. There are two key aspects of XML which *Janus SOAP* application developers should understand at an appropriate level of detail:

1. The syntax, structure, and nomenclature of an XML document.
2. The syntax, nomenclature, and meaning of an XPath expression.

To supplement the information we have provided in [“XML” on page 7](#) and [“XPath” on page 105](#), you may wish to see the definitive list of standards shown in [“Internet standards” on page 115](#). In addition to, and as a subset of, those standards, the following shorter list of references should be useful in understanding the above key aspects:

XML in a Nutshell: A Desktop Quick Reference

This book by Elliotte Rusty Harold and W. Scott Means (First Edition: January, 2001, published by O'Reilly & Associates) is one of many that cover XML, Namespaces, XSLT, XPath, XML processors, and more. These books are somewhat similar, but this one has the benefit of its smaller size. The examples are good, and there is a good summary of XML's history. This first edition is somewhat dated, and the XML grammar omits some important productions (such as AttValue - although this is described elsewhere in the book, it should be listed in the grammar). The XML standards are evolving rapidly; it is difficult to find a completely up to date book. Maybe the second edition contains some of the important missing information, such as XML Schema. For the programming that you will do with XML using *Janus SOAP* or other platforms, some of this book, and the others like it, may be irrelevant or even, because it's scope is so large, confusing, but the book is fairly accurate and probably easier to read than the more formalized W3C standards.

XML background

<http://www.w3.org/XML/1999/XML-in-10-points>

XML spec

<http://www.w3.org/TR/REC-xml>

We refer to this as the **W3C XML Recommendation** throughout this manual. Note that the World Wide Web Consortium ("W3C", the World Wide Web standards organization) uses the term "Recommendation" to indicate that a standard has been approved by the W3C.

XPath spec

<http://www.w3.org/TR/xpath>

We recommend you start with section 5, Data Model.

MSXML MSXML 4.0 Release: Microsoft XML Core Services component. This is the component that is used, among other things, by Internet Explorer to apply an XSL stylesheet to display an XML document. It is a very good way to learn about XPath. Experience shows that IE 6.0 is required to get the correct results. You can download MSXML 4.0 from:

<http://www.microsoft.com/downloads/release.asp?ReleaseID=37176>

IE Tools for validating XML

This lets you, when using Internet Explorer to display a page with XSLT, right-click on the page and see the transformed XML:

<http://msdn.microsoft.com/msdn-files/027/000/543/search.asp>

1.5 Organization of this manual

The next chapter in this manual, [“XML” on page 7](#), is from the *W3C XML Recommendation*; it goes into some detail and should be sufficient to describe the XML features which pertain to *Janus SOAP*.

Following that is [“Janus SOAP Concepts and Data Structures” on page 19](#), which explains the concepts and data structures used in *Janus SOAP*. It also illustrates some XPath usage.

The principal reference material in this manual is then presented in [“\\$XML_xxx Functions” on page 31](#), which describes each of the “operations” you can perform with XML documents.

[“Restrictions” on page 97](#) explains limitations imposed by *Janus SOAP* (for example, the maximum length of a name) and a list of features specified in the standards but not implemented in the latest version of *Janus SOAP*.

[“XPath” on page 105](#) provides a fairly complete explanation of XPath, explaining its semantics and illustrating both the full XPath syntax and the syntax accepted by the current version of *Janus SOAP*.

Finally, [“References” on page 115](#) lists the authoritative sources for the XML family of standards. This, together with [“Reading list” on page 4](#), provide additional resources to help you with XML-centered processing.

CHAPTER 2 *XML*

As explained in “[Janus SOAP Overview and Organization of This Manual](#)” on page 1, XML provides the basis for a large number of varied standards, and for *Janus SOAP*. This chapter introduces the **W3C XML Recommendation**, that is, the XML standard. It gives you basic information about XML for understanding *Janus SOAP*. It also explains some of the XML concepts using the *Janus SOAP* API (that is, the \$XML_XXX functions). This approach gives you concrete examples which you can try in User Language, and which might make the abstract concepts easier to understand.

The syntax of XML provides for hierarchical structuring of data (the outer object is called a document) into the principle object type called an **element**. An element has a name, which need not be unique within the document. An element can have any number of **attributes**, each of which has a name, which must be unique within that element - but not within the document, and a value. Within an element can be a series of values and (“sub-”) elements, which provides XML with its hierarchical nature.

(There is also a provision for assigning unique identifiers to elements; this provides even more structuring possibilities than simple hierarchy. These identifiers are implemented with the element type definition features provided with either Document Type Declarations or with XML Schema. Element type definitions are omitted from this manual; they are not supported in version 6.4 of *Janus SOAP*.)

An XML document has exactly one outer, or “top-level” element, which contains, as descendants, any other elements that may be in the document.

In addition to the data contained in elements and attributes, any number of comments may appear wherever an element may appear. There is also a component called a processing instruction, or **PI**, which is really a comment that has a name.

All names (element names, attribute names, entity references, and PI targets) are case-sensitive; for example, a less-than symbol (“<”) can be included in an attribute value with the characters “<.;” - not “<. ;” nor “&Lt. ;”.

The rest of this chapter explains the syntax of XML and various rules for XML documents. See “[Restrictions](#)” on page 97 for limitations (some of which will be removed in future versions) imposed by *Janus SOAP* on the **W3C XML Recommendation**.

2.1 Example

The next example illustrates all of the major components of an XML document. The formatting into separate, indented lines is provided for readability but is not significant for this and for most business data exchange applications. The letter labels on the left are not part of the document; they are for the explanation which follows:

```

X:  <?xml version='1.0'?>
A:  <!-- Purchase order follows -->
B:  <purchase_order>
C:    <memo>Dave's order was "late"</memo>
D:    <?program-version 4.1?>
E:    <pitm>
      <partID>1234</partID>
F:      <price per="12" amt="1.280"/>
      <qty>36</qty>
G:    </pitm>
H:    <pitm>
I:      <price amt=".29"></price>
      <partID>5678</partID>
      <qty>2</qty>
      </pitm>
    </purchase_order>

```

In the following explanation of each of the labeled lines above, references are made to productions in “[Syntax of document, element, Attribute, Comment, PI](#)” on page 11 using [\[cnn\]](#).

X: <?xml version='1.0'?>

The XML Declaration (XMLDecl, [C23]) is an optional part of the prolog ([B22]), which is the set of components preceding the top-level element. If XMLDecl is present it must be the first markup in the document (only whitespace may precede it). It must specify at least the version; as of the current **W3C XML Recommendation**, "1.0" is the only valid version. The clauses in XMLDecl are positional, that is, they must be given in the order shown in the syntax.

A: <!-- Purchase order follows -->

This is a comment at top-level; [A1], [B22], [D27] allow zero or more comments and PIs before and after the top-level element.

B: <purchase_order>

This is the element start tag or STag ([G40]) of the top-level element ([A1]).

C: <memo>Dave's order was "late"</memo>

With “leaf” elements, that is, if the only thing between the STag and Etag is CharData ([F39]), you can usually implement the information either as an element or as an attribute of the parent element. This example highlights one small distinction:

- AttValue ([M10]) has less flexibility; it can't contain both quotes and apostrophes, nor “<”, nor “&” (an entity ([CD68]) or character ([CD66]) reference must be used to “escape” these characters)
- content ([O43]) is not only free with quotes and apostrophes, but also allows CDsect [Q18]

D: <?program-version 4.1?>

This is a PI [V16]; presumably the name (actually, the target) “program-version” is used by the application reading this document.

E: <pitm> This is the STag of an element which is contained within another element and which contains child elements; this provides a good way to group elements together.

F: <price per="12" amt="1.280"/>

This is an example of the EmptyElemTag ([I44]), which can be useful if an element contains no data (just the name can be meaningful to the application) or, if it only contains data using attributes.

G: </pitm>

This is the ETag [H42] of an element; it must exactly match the STag for the element (again, XML is case sensitive).

H: <pitm>

Here is another STag of an element; it is the “sibling” of another with the same name. The ability to have sub-elements and the ability to repeat elements with the same name in a given parent element are the important data modeling distinctions between elements and attributes.

I: <price amt=".29"></price>

Note that not all instances of a given element type (the price item is an element type) need to have the same attributes, nor do they need to have the same sub-structure. Also, it is completely optional, when an element has no content, whether to use the EmptyElemTag or to use an STag immediately followed by an ETag (as is done above in (F:)).

2.2 XML syntax

This section contains a version of the XML syntax. It is taken from the **W3C XML Recommendation**, which is the authoritative reference:

<http://www.w3.org/TR/REC-xml>

The syntax below has been changed in these ways:

- The only structure in the XML syntax not supported by *Janus SOAP* in version 6.4 is the Document Type Declaration or DTD (“<!DOCTYPE...>”). Actually, a DTD can be tolerated if you use the DTD_IGNORE option of the deserialization \$functions (“\$XML_STR2DOC: Deserialize string into XMLDoc” on page 83 and \$WEB_XML_RECV), but in version 6.4 the information contained in the DTD is not used nor made available to the User Language program.
- Therefore, the productions have been altered to remove those parts of an XML document introduced in the DTD.
- As explained in “Char and Reference” on page 12, only characters representable in 8-bit EBCDIC are handled, so fewer characters are supported in the productions for Letter ([CE84]) and Char ([CA2]).
- The productions are re-ordered (to make it easier to read the grammar), and we have added letters before them, so that when we refer to [B22] in the text, you know that this is between [Ann] and [Cnn] in this grammar, and is production [22] for the same non-terminal (in this case, *prolog*) in the **W3C XML Recommendation**.

The conventions used are:

- 'xxx' (apostrophes) or "xxx" (quotes) enclose an item **xxx** that must appear exactly as shown.
- #**xnn** specifies the character (in ISO-10646) with code value **nn**; for example, #x09 #x0D #x0A #x20 specify the tab, carriage return, linefeed, and space characters, respectively
- [¬**xyz**] specifies any character except **x**, **y**, or **z**
- [**chars**] specifies any character within the set **chars**, where **chars** can be the concatenation of these sets:
 1. **x**, meaning the single character **x**
 2. **x-y**, meaning characters in the range from **x** to **y**, inclusiveThe resulting set of **chars** is the union of the concatenated specified sets.
- **set1 - set2** (where this “-” is not enclosed in [...]) is the set of strings described by **set1**, with the set of strings described by **set2** removed
- | is used to separate alternatives
- ? is used to follow an optional item
- * is used to follow an item which can occur any number of times (including not at all)
- + is used to follow an item which can occur one or more times
- (**xxx**) (parentheses) are used to group items that are followed by * or ?

- **[rule]** (“off to the right”) is used to mention an additional syntax rule
- ***/*comment*/*** is a comment

The syntax is shown in three sections: first, the major components; second, the productions that describe individual characters; third, the components of the “XML Declaration” (<?xml version=...?>).

2.2.1 Syntax of document, element, Attribute, Comment, PI

```
[A1] document ::= prolog element Misc*
[B22] prolog   ::= XMLDecl? Misc*
[C23] XMLDecl ::= '<?xml' VersionInfo
                EncodingDecl? SDDDecl? S? '?>'
[D27] Misc    ::= Comment | PI | S
[E3]  S       ::= (#x20 | #x9 /* Whitespace */
                | #xD | #xA)+
[F39] element ::= STag content ETag
                | EmptyElemTag
                                [Element Type Match]
[G40] STag    ::= '<' Name (S Attribute)*
                S? '>' [Unique Att]
[H42] ETag    ::= '</' Name S? '>'
[I44] EmptyElemTag ::= '<' Name (S Attribute)*
                S? '/>' [Unique Att]

[J4] NameChar ::= Letter | Digit | '.' | '-'
                | '_' | ':'
[K5] Name     ::= (Letter | '_' | ':') (NameChar)*

[L41] Attribute ::= Name Eq AttValue
[M10] AttValue  ::= '"' ([¬&"] | Reference)* '"'
                | "'" ([¬&'] | Reference)* "'"

[N25] Eq       ::= S? '=' S?

[043] content  ::= CharData? ( (element
                | Reference | CDsect | PI
                | Comment) CharData? )*

[P14] CharData ::= [¬&]* - ([¬&]* ']'>' [¬&]*
[Q18] CDsect   ::= CDstart CData CEnd
[R19] CDstart  ::= '<![CDATA['
[S20] CData    ::= (Char* - (Char* ']'>' Char*))
[T21] CEnd     ::= ']'>'

[U15] Comment  ::= '<!--' ( (Char - '-')
                | ('-' (Char - '-')) )* '-->'
```

```
[V16] PI      ::= '<?' PITarget (S (Char* -
                               (Char* '?>' Char*))? '?>'
[W17] PITarget ::= Name - (('X' | 'x') ('M' | 'm')
                          ('L' | 'l'))
```

2.2.2 Char and Reference

```
[CA2] Char      ::= #x9 | #xA | #xD
                  | [#x20-#xFF] /* ISO-10646 */
[CB67] Reference ::= EntityRef | CharRef
[CC66] CharRef   ::= '&#' [0-9]+ ';' [Legal Char]
                  | '&#x' [0-9a-fA-F]+ ';'
[CD68] EntityRef ::= '&' Name ';'
[CE84] Letter    ::= [A-Za-z] [Legal Char]
```

In version 6.4 of *Janus SOAP*, XMLDocs are maintained in EBCDIC; this is why productions [CA2] and [CE84] do not allow the full range of ISO-10646 characters as shown in the **W3C XML Recommendation**. (ISO-10646 is the standard for the universal character set, also known as Unicode.)

All characters which can be encoded in EBCDIC are allowed in the serial form of an XML document; for example, the Bell (#x07) character is allowed. This means that the following rules and restrictions are not enforced by *Janus SOAP*:

- the `Legal Char` rule on `CharRef` ([CC66])
- the same restrictions on `Char` ([CA2], contained in `Comment` [U15], `PI` [V16], and `CData` [S20])
- the same restrictions in the **W3C XML Recommendation** about all characters (and hence applying to `Chardata` [P14], which is in an element's content [O43])

These restrictions are also not enforced by MSXML 4.0 (as of the date of release of version 6.3 of *Janus SOAP*).

One purpose of an `EntityRef` is to allow characters which may be illegal to use in a particular context of an XML document (but notice that a `Reference` is only allowed in an element's content ([O43]) an `AttValue` ([M10]). For example, within an element's content, the string "[>]" is not allowed, so you may replace the greater-than symbol (">") with either its character code in a `CharRef`, or with the predefined entity `>`:

```
]]&gt;;
```

There is also a facility for defining your own entities in a DTD, but since DTDs are not supported in version 6.4 of *Janus SOAP*, the only entity references supported are the five predefined entities: `<`, `>`, `&`, `"`, and `'`.

2.2.3 Components of XMLDecl

```
[XA24] VersionInfo ::= S 'version' Eq
                        ("" VersionNum ""
                         | "" VersionNum "")
[XB26] VersionNum ::= ([a-zA-Z0-9_.:] | '-')+

[XC80] EncodingDecl ::= S 'encoding' Eq
                        ("" EncName ""
                         | "" EncName "" )
[XD81] EncName      ::= [A-Za-z] ([A-Za-z0-9._]
                        | '-')* /* Only Latin chars */

[XE32] SDDec1       ::= S 'standalone' Eq (
                        ("" ('yes' | 'no') "")
                        | ("" ('yes' | 'no') "")) )
```

2.2.4 Names and namespaces - objectives

A feature of XML is allowing a document to contain some elements and attributes which are defined by one organization, and other elements and attributes defined by another organization. In order to achieve this “merging”, the XML Namespaces Recommendation provides for a way to qualify these merged names so that they will not conflict.

Also, the Namespaces Recommendation provides a way for an application to examine, in effect, the “defining organization” of a name in an XML document, so that various properties can be inferred, and names from the same “organization” can be grouped together.

Conceptually, the Namespaces Recommendation qualifies a name with a Universal Resource Identifier (**URI**). There are various rules for various types of URIs; one familiar type is the same as URLs on the World Wide Web, such as

<http://www.w3.org/2001/XMLSchema>

The important aspect of a URI, as far as the names in an XML document are concerned, is simply that it is a unique string for the names that are associated with it.

The characters which are valid in a URI (for example, slash, “/”) exceed the set of characters which are valid in an XML name. Therefore, the technique employed for XML Namespace qualification is to use a special kind of attribute - one that begins with “xmlns” - to associate a name **prefix** with a URI. Then attaching a prefix to a name effectively attaches the URI to a name.

The syntax for making this association is explained in the next section.

2.2.5 Names and namespaces - syntax

The *W3C XML Recommendation* syntax rule for names is shown in “Syntax of document, element, Attribute, Comment, PI” on page 11, as the Name ([K5]) production. In addition, the XML Namespaces Recommendation provides additional rules on element and Attribute names (but no additional rules on PI targets). The syntax from the Namespaces Recommendation is:

```
[NA4] NCName      ::= (Letter | '_' ) (NCNameChar)*
[NB5] NCNameChar ::= Letter | Digit | '.' | '-' | '_'
[NC6] QName      ::= (Prefix ':')? LocalPart
[ND7] Prefix     ::= NCName
[NE8] LocalPart  ::= NCName
```

The restrictions are as follows:

- The name can have at most one colon (“:”), which separates the name into a non-null **prefix** and a non-null **local name**.
- A name without a prefix is simply a local name.
- The **prefix**, if any, must be associated with a **namespace URI** using an attribute of the form:

```
xmlns:prefix="URI"
```

For example, all elements (and attributes of those elements) within the content of the definitions element below can use the prefix “xsd” to qualify their names to belong to the “http://www.w3.org/2001/XMLSchema” URI:

```
<definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  ... content of definitions element ...
</definitions>
```

- An element can also have a default namespace attribute of the form:

```
xmlns="URI"
```

Such a construct, syntactically the same as an Attribute, is called a **namespace declaration**. The scope of a namespace declaration is the element containing it (but not the attributes of that element) - that is, the descendant elements, and their attributes - until some descendent element provides another namespace declaration with the same prefix.

An element can also disable any default namespace with:

```
xmlns=""
```

- The namespace URI associated with a name is

1. the in-scope URI associated with the prefix of the name, if the name has a prefix
 2. the in-scope default namespace URI, if the name does not have a prefix and there is a default namespace URI in scope
 3. no namespace URI, otherwise
- Two names are identical if they have the same local name and either they both do not have a namespace URI or they both have the same namespace URI.

Although the **W3C XML Recommendation** does not require that attribute and element names follow the XML Namespaces Recommendation, the operation of XPath requires it. Therefore, since XPath is so important for *Janus SOAP*, one of its operating modes is to require Namespaces conformance in the XML document. See [“NAMESPACE: set URI for prefix in XPath expressions” on page 82](#).

2.2.6 Namespace URI for XPath prefixes

It is important to realize that the URI associated with a prefix *in the XML document* is controlled by the `xmlns` namespace declarations in the document. However, when a prefix is used in a name in an XPath argument to a `$XML_xxx` function, the URI for that prefix must be established so that the full XPath name (local part and URI namespace) can be obtained. This association of XPath prefixes to URIs can be established using [“\\$XML_NS: Set namespace association for XPath prefix” on page 71](#).

2.3 Well-formed documents and validation

Before an XML document can be processed, its structure must match the rules expressed in the productions in [“Syntax of document, element, Attribute, Comment, PI” on page 11](#), along with the extra rules alluded to in square brackets (for example, `[Unique Att]`, indicating that a single attribute name may not be given twice in the list of attributes for an element). When the syntax is correct, including these rules, the document is called **well-formed**.

Janus SOAP enforces the syntax rules of well-formed documents.

In addition to this checking, an XML processor may also check to see that the format of the document matches the structure and restrictions declared for it in either the Document Type Declaration or the document's Schema. If the document matches the type structure and restrictions, it is called **valid**. In the **W3C XML Recommendation**, this validation of a document is an optional feature of an XML processor.

In version 6.4, *Janus SOAP* does not validate the XML document. A future version will almost certainly incorporate this feature. Note that support of XML Schema is most

likely; Document Type Declarations have several shortcomings, including a limitation on the types of constraints that can be placed on the document, a specialized baroque syntax that doesn't conform to the element/attribute structure of XML, and incorporation of some features that have nothing to do with document validation.

2.4 Normalization during deserialization

When an XML processor, in particular *Janus SOAP*, parses an XML document from character form into an internal representation, it must make some transformations of the document. Some of these transformations have to do with substitution of entity references; for example, `>` in the `content` of an element or in the `AttValue` of an Attribute, will be handled exactly as if a greater-than symbol (“>”) occurred at that point in the document; similarly for character references, such as `[`, which is handled as if a left square bracket symbol (“[”) occurred at that point in the document.

The other significant source of normalizations concerns whitespace characters. In the XML syntax, the whitespace characters are (in hexadecimal, using ISO-10646 character codes):

tab	x09
linefeed	x0A
carriage return	x0D
space	x20

In general, the whitespace characters can be used in the `S` production, which must separate many of the tokens in a document (for example, it must follow the element name, if the `S`Tag contains an Attribute) and may optionally be used in many other places (for example, it may appear before or after the equal sign (“=”) between an Attribute name and its value.

The *W3C XML Recommendation* specifies two normalizing transformations of whitespace:

1. When a special combination of linend characters - carriage return and linefeed - occur **anywhere** in an XML document, they are replaced by a single linefeed character.
2. When any whitespace character appears in the value of an attribute, it is replaced by a single space character.

The following two sub-sections describe these transformations in more detail.

In addition, the deserialization \$functions offer options to control additional normalization of whitespace characters which occur in the `content` of an element; this is described in “[\\$XML_STR2DOC: Deserialize string into XMLDoc](#)” on page 83 (the other deserializer is \$WEB_RECV_XML, which operates in the same fashion with these options).

2.4.1 Normalized line-end

As specified in “2.11 End-of-Line Handling” of the *W3C XML Recommendation*, all instances of a carriage return followed by linefeed are converted to a single linefeed character.

This behavior only applies to deserialization; that is, there is no modification of whitespace characters in values passed as the *value* argument of “[\\$XML_INSELT: Insert ELEMENT node](#)” on page 55 or “[\\$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child](#)” on page 61. Therefore the values of the “FOO1” element and the “FOO2” PI created by the following two \$XML_xxx functions are different:

```
* Get EBCDIC carriage return and linefeed:
%CL = $X2C('ØD25')
* Element value is linefeed:
%X = $XML_STR2DOC(%D, '<top> <FOO1>' WITH %CL -
  WITH '</FOO1> </top>')
* PI value is carriage return and linefeed:
%X = $XML_INSPI(%D, 'FOO2', %CL, '/')
```

Also, the normalization applies to the characters in the input serialized string, not the values after entity substitution. Therefore the values of “FOO1” and “FOO2” created by the following two \$XML_STR2DOC invocations are different:

```
* Get EBCDIC carriage return and linefeed:
%CL = $X2C('ØD25')
* Element value is linefeed:
%X = $XML_STR2DOC(%D, '<FOO1>' WITH %CL -
  WITH '</FOO1>')
%X = $XML_INITDOC(%D)
* Element value is carriage return and linefeed
* (note, character references are ISO-10646):
%X = $XML_STR2DOC(%D, '<FOO2>&#xØD;&#xØA;' -
  WITH '</FOO2>')
```

Finally, whitespace in the TEXT node child of an element (but not in any other type of node) can be affected by the WSP_PRESERVE/WSP_NORM option of “[\\$XML_STR2DOC: Deserialize string into XMLDoc](#)” on page 83.

2.4.2 Normalized attribute value

In addition to replacing all CR-LF sequences by LF (“[Normalized line-end](#)” on page 17), attribute values have additional whitespace normalization. As specified in “3.3.3 Attribute-Value Normalization” of the *W3C XML Recommendation*, after the CR-LF normalization, every instance of a whitespace character (tab, line-feed, and carriage return) in an Attribute value is converted to a space character. Leading and trailing spaces are not stripped, nor are sequences of multiple spaces collapsed.

This behavior only applies to deserialization; that is, there is no modification of whitespace characters in attribute values passed as the *value* argument of “[\\$XML_INSATT: Insert attribute](#)” on page 50. Therefore the values of the “FOO” attribute created by the following two \$XML_xxx functions are different:

```
* Get carriage return:
%C = $X2C('ØD')
* Attribute value is space:
%X = $XML_STR2DOC(%D, '<top FOO="' WITH %C -
  WITH '"> <in/> </top>')
```

Also, the normalization applies to the characters in the input serialized string, not the values after entity substitution. Therefore the values of the “FOO” attribute created by the following two \$XML_STR2DOC invocations are different:

```
* Get carriage return:
%C = $X2C('ØD')
* Attribute value is space:
%X = $XML_STR2DOC(%D, '<top FOO="' WITH %C -
  WITH '"/>')
```

```
%X = $XML_INITDOC(%D)
* Attribute value is carriage return - CR is
* the same in EBCDIC and ISO-10646:
%X = $XML_STR2DOC(%D, '<top FOO="#xØD;"/>')
```

Finally, whitespace in an attribute (and in any type of node other than a TEXT node, which is the child of an element) is not affected by the WSP_PRESERVE/WSP_NORM option of “[\\$XML_STR2DOC: Deserialize string into XMLDoc](#)” on page 83.

Janus SOAP is based on the use of XML documents. The preceding chapters of this manual, and various references on XML, explain that an XML document is an object that can contain any type of data; do not think of an XML document as intended for human reading. Even though the purpose of an XML document is not for human reading, one of the benefits is that an XML document can be simply and meaningfully expressed or represented entirely with readable characters. This character form of an XML document is called the **serial** form. When operating on an XML document with *Janus SOAP*, the serial form is converted to an internal data structure, called an **XMLDoc**.

You can perform the following operations on XML documents:

Receive

The process of receiving a document actually consists of two steps:

1. Receiving the document text using some “transport” mechanism, such as *Janus Web Server* (HTTP, as server) *Janus Sockets* (usually, HTTP, as client), *Model 204 MQ Series*, access from a file, etc..
 2. Converting the XML document into its internal representation (an XMLDoc) so that other operations can be performed on it.
- The first and second steps can be combined, if the XML document is received by *Janus Web Server*; that is performed by `$WEB_XML_RECV`. For other forms of transport, the steps are performed separately, with the first step receiving the text form of the document into a `LONGSTRING`, and the second step performed by `$XML_STR2DOC`.

Update

You can modify an XMLDoc with various `$XML_xxx` functions. Note that if you start with an empty XMLDoc, these functions (specifically, the `$XML_INSxxx` functions) allow you to generate an XMLDoc “directly”, without first representing it in the serial text form. You can also update an XMLDoc into which you have received a document.

Access

Since an XML document is a hierarchical structure, in your application you will want to select some part of the hierarchy to, for example, obtain its value. There are various `$XML_xxx` functions to do this. Updating `$XML_xxx` functions also require that you specify where in the hierarchy an update is performed.

Selecting nodes from an XMLDoc is performed using XPath. XPath is used when merely accessing a single node in the document, for example, getting an element's string value with `$XML_VAL`. You can also work with lists of selected nodes, called `nodelists`;

\$XML_NODES produces such a list, there are other \$XML_xxx functions to work with them, and individual items in a nodelist can be used to directly operate on that node.

Send

The process of sending a document actually consists of two steps:

1. Converting the XMLDoc into its serial text representation.
2. Sending the document text using some “transport” mechanism, such as *Janus Web Server* (HTTP, as server) *Janus Sockets* (usually, HTTP, as client), *Model 204 MQ Series*, access from a file, etc..

The first and second steps can be combined, if the XML document is sent by *Janus Web Server*; that is performed by \$WEB_XML_SEND. For other forms of transport, the steps are performed separately, with the first step performed by \$XML_DOC2STR, and the second step sending the result of \$XML_DOC2STR using the appropriate transport.

Other operations

There are other operations, such as creating and initializing an XMLDoc or nodelist, setting some property of an XMLDoc - for example, the URI associated with a prefix to be used in an XPath expression.

\$XML_PRINT can be useful to display a document, or some part of it, usually for debugging purposes.

This chapter and the next explain how to perform the above operations on XML documents. Note that all of the \$XML_xxx functions are documented in the next chapter, “\$XML_xxx Functions” on page 31. The other two \$functions which are part of *Janus SOAP* are \$WEB_XML_SEND and \$WEB_XML_RECV; these are documented in the *Janus Web Server Reference Manual*.

3.1 XMLDoc structure, and node types

The \$XML_xxx functions operate on a representation of an XML document which is called an **XMLDoc**; an XMLDoc is a tree structure of nodes. The types of nodes which can be in an XMLDoc are the following:

ATTRIBUTE

This type of node is used to represent an attribute of an XML element (except for namespace declarations; see NSATTRIBUTE).

When we discuss a node type and either an ATTRIBUTE or a NSATTRIBUTE can be used, we will refer simply to an attribute node (lower case letters).

Note that some attributes can be affected by an ELEMENT's ancestors:

- `xmlns:xxx` (that is, NAMESPACE attributes) - In accordance with the XPath Recommendation, all namespace declarations in scope for an element appear as namespace nodes of that element, whether or not they are directly specified on the element's start tag. (This is true when the NAMESPACE setting is ON; otherwise, there are no NAMESPACE nodes in the XMLDoc. See [“\\$XML_SET: Change XMLDoc option setting” on page 80.](#))
- `xml:lang` and `xml:space` - Conversely, these attributes only appear as attribute nodes of the elements in which they are directly specified. However, the **effects** of these attributes propagate to an element's descendants, until overridden.

COMMENT This type of node is used to represent a comment (“<!--...-->”) in an XML document.

DOCUMENT This type of node is the root of the XMLDoc tree. It has zero or one ELEMENT child nodes and any number of COMMENT and PI child nodes. It and ELEMENT nodes are the only nodes which can have child nodes.

ELEMENT This type of node is used to represent an element in an XML document. It and the DOCUMENT node are the only nodes which can have child nodes.

NSATTRIBUTE This type of node is used to represent a namespace declaration for an XML element.

When we discuss a node type and either an ATTRIBUTE or a NSATTRIBUTE can be used, we will refer simply to an attribute node (lower case letters).

PI This type of node is used to represent a processing instruction (“<?target ...?>”) in an XML document.

Note: Although the “<?xml version=...?>” declaration has the same appearance as a processing instruction, it is not a PI. Also note that the values of an XMLDoc's “<?xml ...” declaration can be obtained with [“\\$XML_INFO: Get XMLDoc option setting” on page 49](#) and set with [“\\$XML_SET: Change XMLDoc option setting” on page 80.](#)

TEXT This type of node is used to represent character content within an XML element. Note that a TEXT node will never contain the null string, and that two TEXT nodes will never be adjacent.

The XMLDoc node types listed above correspond almost exactly with the structures contained in an XML document, with the DOCUMENT node corresponding to a node

which contains the document as a whole. The children of the DOCUMENT node are the “top-level” element and any top-level processing instructions and/or comments which precede or follow it. Do not confuse the DOCUMENT node, which is the root of the XMLDoc tree, with the top-level element of the XML document.

3.2 XMLDoc and nodelist data structures and states

The XMLDoc is an internal data structure which represents an XML document instance; an XMLDoc is created by an occurrence of the \$XML_DOC function, which returns the identifier of the XMLDoc (the **XMLDoc ID**). An XMLDoc can also be accessed with a **nodelist**, which contains a list of nodes selected from an XMLDoc. The nodelist is a data structure allocated by an occurrence of the \$XML_NL function, which returns the identifier of the nodelist (the **nodelist ID**). A single XMLDoc can have zero, one or more nodelists associated with it.

The DOCUMENT node of an XMLDoc is always present. You can insert additional nodes, either by processing a character stream containing an XML document instance (e.g., with \$WEB_XML_RECV), or by using \$XML_INSxxx functions to insert nodes.

All operations which operate on the “contents” of an XMLDoc are performed using XPath expressions (actually, as explained in “XPath syntax” on page 108, we use PathExpr, in the XPath syntax) to select one or more nodes of an XMLDoc. There are two forms of XPath expressions:

Absolute XPath expression

An absolute XPath expression selects nodes from an XMLDoc, starting at the root, or DOCUMENT, node. The syntax of an absolute XPath expression begins with a forward slash (“/”). Whenever you use an absolute XPath expression, since the starting point is always fixed (the root), you can provide the ID, either of the XMLDoc to search, or of one of its nodelists.

Relative XPath expression

A relative XPath expression selects nodes from an XMLDoc, starting from a **context node** which is determined when the expression is used. The syntax of a relative XPath expression begins with a character other than a slash. Whenever you use a relative XPath expression, you must specify the starting point. This can be done in either of the following ways:

1. Provide an XMLDoc ID; in this case, the context node is the DOCUMENT node.
2. Provide a nodelist ID and an optional number indicating which one of the nodes on the nodelist to use as the context node. That number defaults to 1, i.e., the first node of the nodelist.

Note:

- We are planning for some pervasive changes for the Janus SOAP \$functions API in version 6.5; one of these will be to remove the use of the optional argument (designated as **contx_n** in the \$XML_xxx functions descriptions) in \$functions that have an XPath expression argument.
- As a start in this direction, we have implemented \$XML_PLC and \$XML_DEL with an XPath expression argument, but no **contx_n** argument. These \$functions introduce the “style” of the *Janus SOAP* API which will be in place when only a “single node” reference is used with XPath expressions, and so for them, the nodelist is standing in place of a single node datatype, and any node other than the first in the nodelist is ignored.

In addition to operating on the contents of an XMLDoc, there are several operations (e.g., “[\\$XML_SET: Change XMLDoc option setting](#)” on page 80) which operate on the XMLDoc as a whole; these operations only allow an XMLDoc ID argument. If you need to obtain the ID of the XMLDoc associated with a nodelist, use “[\\$XML_DOCID: Get ID of XMLDoc associated with nodelist](#)” on page 43.

The next two subsections describe the XMLDoc and nodelist states; the following section (“[XPath, nodelists, \\$XML_xxx example](#)” on page 24) continues the explanation of XPath and nodelists. Further information about XPath expressions and nodelists is also contained in “[XPath](#)” on page 105.

3.2.1 XMLDoc states

An XMLDoc can have one of the following three states:

EMPTY An XMLDoc in this state does not have any nodes other than the DOCUMENT node. This is the state of an XMLDoc after execution of \$XML_DOC or \$XML_INITDOC.

WELL-FORMED

An XMLDoc in this state contains at least the top-level ELEMENT node.

Non-EMPTY not WELL-FORMED

An XMLDoc in this state contains at least one node but no ELEMENT nodes.

Note that only an XMLDoc in the WELL-FORMED state may be “serialized” (that is, converted into a complete text representation of an XMLDoc), and that you can only use an EMPTY XMLDoc as the target of “deserializing” the text representation of an XML document.

3.2.2 Nodelist states

A nodelist can have one of the following three states:

- FREE** A nodelist in this state is not associated with any XMLDoc (as a consequence, it cannot have any nodes). This state is explained in “[\\$XML_FREENL: FREE nodelist](#)” on page 48.
- EMPTY** A nodelist in this state does not have any nodes, but it is associated with an XMLDoc. This can be due to an XPath result without any nodes (“[\\$XML_NODES: Select list of nodes within XMLDoc](#)” on page 69 and “[\\$XML_PLC: Place nodes from XPath result onto a nodelist](#)” on page 73) or due to the removal of all nodes from a nodelist (“[\\$XML_RMV_NL: Remove one nodelist from another](#)” on page 78).
- Non-EMPTY** A nodelist in this state has one or more nodes; as a consequence, of course, it is associated with an XMLDoc.

The following pseudo-state is also referenced in this manual:

- Non-FREE** This is an EMPTY or non-EMPTY nodelist, that is, it is a nodelist which is associated with an XMLDoc.

The only way that a nodelist can move from the FREE to the non-FREE state is to use it as the “output” or “target” nodelist argument for one of the following \$functions:

- `$XML_INSELT`
- `$XML_NODES`
- `$XML_PLC`

3.3 XPath, nodelists, \$XML_xxx example

This section illustrates a small XML document received as an HTTP request, and part of a User Language request showing how some \$XML_xxx functions are used. First, here is the XML document:

```
<purchase_order>
  <date>25 July, 2001</date>
  <pitm>
    <partnum>1234</partnum>
    <qty>3</qty>
  </pitm>
  <pitm>
    <partnum>5678</partnum>
    <qty>2</qty>
  </pitm>
</purchase_order>
```

Here is some User Language which could be used to receive and process this request.

```

* Create XMLDoc, get HTTP request as contents:
%DOC = $XML_DOC
%T = $WEB_XML_RECV(%DOC)

* Create work nodelist, get all pitm elements:
%NL = $XML_NL
%COUNT = $XML_NODES(%NL, %DOC, -
    '/purchase_order/pitm')

* Process each pitm:
FOR %J FROM 1 TO %COUNT
    %PARTNUM = $XML_VAL(%NL, 'partnum', , %J)
    %QTY      = $XML_VAL(%NL, 'qty', , %J)
    ...
END FOR

```

\$XML_VAL and \$XML_NODES, like many \$XML_xxx functions, have two optional arguments which allow you to process any of the nodes in an XMLDoc, rather than the default, which is to process the first node of its input nodelist.

1. One of these optional arguments shown above is an XPath expression (for \$XML_NODES, `/purchase_order/pitm`; for \$XML_VAL, `partnum` and `qty`). An XPath expression selects a list of nodes, starting either from the XMLDoc root (when an **absolute** Xpath expression is used) or from a particular context node in an XMLDoc (when a **relative** Xpath expression is used). Syntactically, an XPath expression that begins with a slash ("`/`") is absolute.

\$XML_NODES operates on the entire result of its XPath expression argument: it sets a target nodelist argument to this list of nodes so you can use the nodelist as input to other \$XML_xxx functions. Most other \$XML_xxx functions, however, operate on the first of the nodes resulting from the argument's XPath expression.

2. The other optional argument is shown for \$XML_VAL above; this is a position (`%J` in the example) in the input nodelist (`%NL` in the example). This determines which one of the nodes in the input nodelist is used as the context for a relative XPath expression. This \$XML_VAL example does not have an XPath expression, so the default is used, which is to simply select the context node itself, i.e., the `%J`th node in the input nodelist `%NL`.

The next sub-section describes the three arguments which determine an XPath expression result.

3.3.1 Context node, [head of] argument result nodelist

Many \$XML_xxx functions perform their operations on a list of selected nodes (or, usually, on the first node in a list) that is determined from three input arguments, shown here in the order in which the arguments are used to select the nodes (but not the order the arguments are specified to the \$XML_xxx functions):

- Id** This is the identifier, either of an XMLDoc, or of a nodelist on it. If the third of this set of arguments (*XPath_expr*) is relative and this is the ID of a nodelist, it must be non-empty.
- Contx_n** Position of context node in context nodelist or XMLDoc. Must be:
- 1 if *id* is an XMLDoc ID;
 - ≥ 1 and \leq number of nodes in nodelist if *XPath_expr* is relative and *id* is a nodelist ID.
- Optional: default is 1.
- XPath_expr** This is an XPath expression, which selects a list of nodes as a nodelist. This is optional and defaults to the (relative) XPath expression “.”, which selects the context node itself. If this expression is relative (that is, does not start with a slash (“/”), the context node for the XPath expression is:
- the DOCUMENT node, if the first of this set of arguments (*id*) is an XMLDoc ID;
 - the node referenced by the nodelist item at position *contx_n*, if *id* is a nodelist ID.

\$XML_NODES sets a target nodelist argument to the result of the XPath expression. Most other \$XML_XXX functions which have these three arguments (such as \$XML_VAL) do not actually create the nodelist resulting from the XPath expression; rather they operate on the first node of that nodelist. We call that first node the **head of the argument XPath result**.

For a relative XPath expression, the position is first used to determine, from the input nodelist, the context node for the XPath expression.

Finally, the XPath expression is used to select a list of nodes to operate on (or the head of the list is operated on).

Do not confuse the context node position argument with XPath's position() function (which is the same as an “occurrence number” in square brackets, e.g., [2]), even though both can be used to process a node from a set of nodes based on position. For example, either of the two following fragments will obtain the string-value of the partnum of the second pitm child of the first node in nodelist %X.

First, using XPath to go directly to the second pitm:

```
%NAME = $XML_VAL(%X, 'pitm[2]/partnum')
```

Alternatively, using XPath to select the list of pitm children and then using the context node position to obtain the second node in that list:

```
%T = $XML_NODES(%N2, %X, 'pitm')  
%NAME = $XML_VAL(%N2, 'partnum', 2)
```

The first example is better if this is the only operation to be performed on this nodelist, because, in the second example:

1. you have to manipulate an additional nodelist (%N2 in the above example)
2. there is additional storage used to point to the other nodes
3. there is overhead to locate all nodes identified by the XPath expression

The major items in the above list (2 and 3) are avoided by using [“\\$XML_NODE_SING: Select singleton nodelist from XMLDoc” on page 68](#) rather than \$XML_NODES,

Note that nodelists are always stored in “document order”, but the XPath position() function depends on the order implied by the XPath “axis”. If the axis in the *XPath_expr* is a “reverse” axis, the simple correspondence implicit in this example can break down. See [“Order of nodes in nodelist” on page 112](#).

3.4 Updating

The contents of an XMLDoc can be established by one of the serialization \$functions, either [“\\$XML_STR2DOC: Deserialize string into XMLDoc” on page 83](#) or \$WEB_XML_RECV, which is described in the *Janus Web Server Reference Manual*. Whether you use that to set the “initial” contents of an XMLDoc or start with an EMPTY XMLDoc, you can then insert nodes into it, using one of the \$XML_INSxxx functions (such as [“\\$XML_INSELT: Insert ELEMENT node” on page 55](#)) or using [“\\$XML_COP: Copy XMLDoc subtree” on page 35](#).

Also, once an XMLDoc has one or more nodes in addition to the DOCUMENT node, you can delete nodes from it using [“\\$XML_DEL: Delete XMLDoc subtree” on page 37](#).

3.4.1 Inserting nodes/copying subtrees

The insert and copy operations are designed to make it easy to “append” nodes to an XMLDoc in a “depth-first, left-to-right” order in the simple case. To do that, the default behavior is to insert a node as the **last** child of the head of the argument XPath result, but you can over-ride this default by specifying which child number (first, second, etc.) you want to the node to be inserted as (with the special value of -1 meaning, insert as the last child). (\$XML_INSATT does not provide a child number, because the order of attribute nodes is not under the control of the application, and just to be legalistic, attribute nodes are not children of their elements; they are, simply, **attributes** of their elements.)

In addition:

- The `$XML_INSELT` function sets a target nodelist argument to contain just the inserted `ELEMENT` node, ready for inserting its children or attributes. It also allows insertion of a single `TEXT` child of the `ELEMENT`; this approach is useful for inserting attributes in “leaf elements”.
- The `$XML_INSTEXT` function has an option to insert an `ELEMENT` node together with a child `TEXT` node; this latter combined operation is a common case for “leaf elements”.

Here is an example of use of the updating `$XML_xxx` functions:

```
%DOC = $XML_DOC
%NL = $XML_NL
%T = $XML_INSELT(%NL, %DOC, 'story', , '/')
%T = $XML_INSCOM(%NL, 'My first XML document')
%T = $XML_INSTEXT(%NL, 'greeting', 'Hello, world')
%Y = $XML_NL
%T = $XML_INSELT(%Y, %NL, 'paragraph')
%T = $XML_INSTEXT(%Y, 'line', 'Ask not what')
%T = $XML_INSTEXT(%Y, 'line', 'Hear no evil')
```

This creates the following XML document:

```
<story>
  <!--My first XML document-->
  <greeting>Hello, world!</greeting>
  <paragraph>
    <line>Ask not what</line>
    <line>Hear no evil</line>
  </paragraph>
</story>
```

3.4.2 Deleting nodes

See “[\\$XML_DEL: Delete XMLDoc subtree](#)” on page 37 for the \$function which is used to delete a sub-tree from an XMLDoc.

Note that if any deleted nodes had been referenced in a nodelist, they may not subsequently be referenced using a relative XPath expression (nor implicitly so, as the first item of a nodelist argument such as to “[\\$XML_COP: Copy XMLDoc subtree](#)” on page 35). If you need to “clean up” a nodelist item which refers to a deleted XMLDoc node, you can use “[\\$XML_RMV_NL: Remove one nodelist from another](#)” on page 78, as shown in the following example:

```

%N = $XML_NL
%R = $XML_NL
* Get nodelist for chapters, and delete second
* chapter from the XMLDoc:
%X = $XML_NODES(%N, %D, '/book/chapter')
%X = $XML_NODE_SING(%R, %N, , 2)
%X = $XML_DEL(%R)
* Cleanup the chapter nodelist, show author of
* remaining chapters, & display the document:
FOR %I FROM 1 TO $XML_RMV_NL(%N, %R)
    PRINT 'Author:' AND $XML_VAL(%N, '@author', %I)
END FOR
%X = $XML_PRINT(%D, '/')

```

Note that in the above example, the only reason %N is cleaned up (with \$XML_RMV_NL) is to get rid of the second item from the nodelist, before referencing the nodelist items using relative XPath (the `author` attribute in \$XML_VAL).

3.5 Transport: sending and receiving XML

The provision for sending and receiving XML is very simple:

- Sending XML involves converting the information in an XMLDoc to a character stream; this operation is called **serialization**. There is a \$function which is designed to send an XML document as an HTTP response; this is \$WEB_XML_SEND and is described in the *Janus Web Server Reference Manual*. For other transport mechanisms, such as *Janus Sockets* or *Model 204 MQ Series*, “[\\$XML_DOC2STR: Serialize XMLDoc as UTF-8 string](#)” on page 44 is used to place the serialized form into a LONGSTRING, which can then be sent.
- Receiving XML involves converting the information in the character, marked-up form of an XML document into an XMLDoc; this operation is called **deserialization**. There is a \$function which is designed to receive an XML document which has arrived as an HTTP request; this is \$WEB_XML_RECV and is described in the *Janus Web Server Reference Manual*. For other transport mechanisms, such as *Janus Sockets* or *Model 204 MQ Series*, the character format XML document can be placed into a LONGSTRING; “[\\$XML_STR2DOC: Deserialize string into XMLDoc](#)” on page 83 is used to then place the information into an XMLDoc.

A key part of *Janus SOAP* is a set of \$functions for manipulating an XML document. These functions begin with the 5 characters “\$XML_” and are referred to as the “\$XML_xxx functions”. In addition, there are other \$functions you will use with *Janus SOAP* to work with XML, for example to obtain an XML document from a network request, or to provide an XML document as a network response. Such \$functions are usable only with *Janus SOAP*, but they are provided with some other product, which also contains their full documentation.

Two *Janus Web Server* functions important for working with XML **are** documented in this chapter, however: \$WEB_XML_RECV and \$WEB_XML_SEND.

In addition to the reference material in this chapter, the index contains a major heading labelled **\$Function prototypes**. Under this heading are minor headings containing the form of the \$functions, for your convenient reference.

4.1 LONGSTRING arguments and result

The LONGSTRING datatype was introduced in version 6.2 of the *Sirius Mods*. It provides an atomic type which can contain a string longer than 255 bytes. For more information about LONGSTRINGs, see the *Sirius Functions Reference Manual*.

The \$XML_xxx functions are LONGSTRING capable. This means that all \$XML_xxx string arguments and results are of type LONGSTRING, which is to say:

- Input values may exceed 255 bytes in length.
- Various \$XML_xxx functions will return a string longer than 255 bytes, if the corresponding value in the XML document exceeds 255 bytes.

The following sub-sections provide some guidelines to determine when you **must** use a LONGSTRING %variable to hold the result of a \$XML_xxx function. However, since the server table requirements and the processing overhead for a LONGSTRING are not much more than for a STRING LEN 255 %variable, it is a good idea with the \$XML_xxx functions to use a LONGSTRING wherever you might be using a STRING LEN 255 %variable, unless it is very easy to determine that the length will not exceed 255 bytes, and you have extensive processing with the result %variables after the \$XML_xxx invocations and you are concerned about the very minor overhead of LONGSTRINGs.

4.1.1 **\$XML_DOC2STR**

Almost always, you should use a LONGSTRING %variable to hold the result of \$XML_DOC2STR - the total concatenated length of all markup and character content in a document will most likely exceed 255 bytes. Thus, the first invocation of \$XML_DOC2STR below will never fail (for length reasons) but the second will usually cause a request cancellation:

```
%SS STRING LEN 255
%LS LONGSTRING
%LS = $XML_DOC2STR(%DOC)
%SS = $XML_DOC2STR(%DOC)
```

4.1.2 **\$XML_VAL**

Generally, it is a good idea to use a LONGSTRING %variable to hold the result of \$XML_VAL, unless you know that you cannot obtain a string-value longer than 255 bytes. For example, the first two invocations of \$XML_VAL below will succeed but the third will cause a request cancellation:

```
%SS STRING LEN 255
%LS LONGSTRING
%X = $XML_STR2DOC(%DOC, '<top> <big>' WITH -
    $LSTR_LEFT('a', 300) WITH '</big> <little>' -
    WITH 'Less than 256 chars</little> </top>')
%LS = $XML_VAL(%DOC, '/top/big')
%SS = $XML_VAL(%DOC, '/top/little')
%SS = $XML_VAL(%DOC, '/top/big')
```

As noted above, the best approach here is probably to use LONGSTRING %variables where you might use STRING LEN 255 %variables, unless it is easy to determine that the length will not exceed 255 bytes. and you are convinced that the very minor overhead of LONGSTRINGs will become significant in your application.

4.1.3 **\$XML_NAME, \$XML_NS[INFO], other \$XML_xxx functions**

Besides \$XML_DOC2STR and \$XML_VAL, other \$XML_xxx functions either cannot return a value longer than 255 bytes or, with typical XML documents, are unlikely to do so. If you have name or namespace information that exceeds 255 bytes, it may be necessary to use a LONGSTRING %variable. For example, the first two invocations of \$XML_NAME below will succeed but the third will cause a request cancellation:

```

%SS STRING LEN 255
%LS LONGSTRING
%X = $XML_STR2DOC(%DOC, '<top> <' WITH -
    $LSTR_LEFT('big', 300, '_') WITH -
    '/> </top>')
%LS = $XML_NAME(%DOC, '/*')
%SS = $XML_NAME(%DOC, '/*')
%SS = $XML_NAME(%DOC, '/top/*')

```

As noted above, the easiest approach here is probably to use LONGSTRING %variables where you might use STRING LEN 255 %variables, unless it is easy to determine that the length will not exceed 255 bytes. and you are convinced that the very minor overhead of LONGSTRINGs will become significant in your application.

4.2 **\$XML_CHECK: Check validity of XMLDoc or nodelist ID**

\$XML_CHECK checks the validity of an XMLDoc ID or nodelist ID.

```

%type = $XML_CHECK(id)

id Identifier to be checked.

```

\$XML_CHECK arguments

\$XML_CHECK takes one argument and returns 1 if its argument is a valid XMLDoc identifier, 2 if it is a valid nodelist identifier, and 0 otherwise.

```

$xml_check does not have any request cancel
errors

```

\$XML_CHECK errors

Note that various \$XML_xxx functions impose additional restrictions on their “id” arguments, such as requiring that a nodelist be non-FREE. That restriction can be checked using “[\\$XML_DOCID: Get ID of XMLDoc associated with nodelist](#)” on page 43, and others can be checked with other \$XML_xxx functions, notably “[\\$XML_COUNT: Get number of nodes in XPath result](#)” on page 34 and “[\\$XML_COUNT_NL: Get number of nodes in nodelist](#)” on page 35.

\$XML_CHECK is new in version 6.4 of *Janus SOAP*.

4.3 **\$XML_COUNT: Get number of nodes in XPath result**

\$XML_COUNT gets the number of nodes in the result of its argument XPath expression.

```
%count = $XML_COUNT(id, XPath_expr, contx_n)
```

<code>id</code>	Identifier of XMLDoc or non-FREE nodelist. For relative <i>XPath_expr</i> : <ul style="list-style-type: none">• If <i>id</i> is an XMLDoc ID, the DOCUMENT node is the context.• If <i>id</i> is a nodelist ID, it must be non-EMPTY, and the first item is the context.
<code>XPath_expr</code>	An XPath expression which results in a nodelist, the count of which is returned.
<code>contx_n</code>	Position of context node in context nodelist or XMLDoc. Must be: <ul style="list-style-type: none">• 1 if <i>id</i> is an XMLDoc ID;• ≥ 1 and \leq number of nodes in nodelist if <i>XPath_expr</i> is relative and <i>id</i> is a nodelist ID. Optional: default is 1.

\$XML_COUNT arguments

\$XML_COUNT takes three arguments and returns the number of nodes in the argument XPath result.

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*XPath_expr*) invalid or missing
- Argument three (*contx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- *XPath_expr* relative, and nodelist *id* item number *contx_n* references a node which has been deleted from the XMLDoc
- Insufficient free space in CCATEMP

\$XML_COUNT request cancel errors

Note that the *XPath_expr* argument is required, which is not the case with most \$XML_xxx functions - usually the default is '.', that is, the context node. However, using the context node for *XPath_expr* would cause this \$function to always return 1, unless the nodelist is FREE or EMPTY, which would cancel the request.

4.4 **\$XML_COUNT_NL: Get number of nodes in nodelist**

\$XML_COUNT_NL gets the number of nodes in a nodelist.

```
%count = $XML_COUNT_NL(n1)

n1  Identifier of non-FREE nodelist.
```

\$XML_COUNT_NL arguments

\$XML_COUNT_NL takes one argument and returns the number of nodes in nodelist *n1*.

- Argument one (*n1*) not a non-FREE nodelist

\$XML_COUNT_NL request cancel errors

4.5 **\$XML_COP: Copy XMLDoc subtree**

\$XML_COP copies a subtree of an XMLDoc, inserting it as a child of a target parent node.

```
%junk = $XML_COP(target_parent, -  
source_subtree, child_num)
```

target_parent Identifier of XMLDoc, if the subtree is to be copied as a child of the DOCUMENT node, or of a nodelist, whose first item is an ELEMENT, which is to be the parent of the copy of the subtree.

source_subtree Identifier of nodelist, whose first item is the top of the subtree to copy.

child_num Number of existing (non-attribute, of course) child to insert before, or -1; for example, with the following values of *child_num*, the node will be inserted:

- 1: after last (non-attr) child (or perform normal insertion if *source_subtree* node is an attribute)
- 1: before first (non-attr) child
- 2: before second (non-attr) child
- etc...

This argument is optional: default is -1. If the *source_subtree* node is an attribute, it must be -1; otherwise it must be less than two plus the number of (non-attr) children of the head of the argument XPath result.

\$XML_COP arguments

\$XML_COP takes three arguments and returns zero.

The following example will copy the top-level ELEMENT from one XMLDoc as the “body” of a SOAP request document:

```
%OUT = $XML_DOC  
%X = $XML_STR2DOC(%OUT, '<SOAP-ENV:Envelope/>')  
%N1 = $XML_NL  
%X = $XML_INSELT(%N1, %OUT, 'SOAP-ENV:Body', '/*')  
%REQUEST = $XML_DOC  
... insert or deserialize into %REQUEST  
%N2 = $XML_NL  
%X = $XML_NODES(%N2, %REQUEST, '/*')  
%X = $XML_COP(%N1, %N2)
```

- Argument one (*target_parent*) neither an XMLDoc ID nor the ID of a nodelist whose first item is an ELEMENT or DOCUMENT node
- Argument two (*source_subtree*) not a non-EMPTY nodelist
- Head of *source_subtree* a DOCUMENT node
- *Target_parent* the same as, or a descendant of, *source_subtree*
- Source and target XMLDocs have different values of NAMESPACE setting
- Head of nodelist *Target_parent* or *source_subtree* references a node which has been deleted in the XMLDoc
- Argument three (*child_num*) < 1 and not -1, or greater than one plus the number of (non-attr) children of target parent node, or not -1 if source subtree node is an attribute
- Any of the errors which would occur based on insertion of the subtree node at the given target parent, for example, insertion of a duplicate attribute; see the \$XML_INSxxx function corresponding to the type of the subtree node

\$XML_COP request cancel errors

Note that the target parent and subtree nodes can either be in the same or in different XMLDocs.

\$XML_COP is new in version 6.4 of *Janus SOAP*.

4.6 \$XML_DEL: Delete XMLDoc subtree

\$XML_DEL deletes a subtree of an XMLDoc.

```
%junk = $XML_DEL(id, XPath_expr)
```

id Identifier of XMLDoc or non-FREE nodelist.
For relative *XPath_expr*:

- If *id* is an XMLDoc ID, the DOCUMENT node is the context.
- If *id* is a nodelist ID, it must be non-EMPTY, and the first item is the context.

XPath_expr An XPath expression which results in a nodelist, the head of which is the top of the subtree to delete. Optional: default is '.', that is, the context node.

\$XML_DEL arguments

\$XML_DEL takes two arguments and returns zero.

The following example will delete the first “employee” child of the top-level ELEMENT of the XMLDoc:

```
%X = $XML_DEL(%DOC, '/*/employee')
```

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*XPath_expr*) invalid
- Head of result of *XPath_expr* is the DOCUMENT node
- Head of result of *XPath_expr* is a node whose left and right siblings are TEXT nodes
- *XPath_expr* relative, and nodelist *id*'s first item references a node which has been deleted from the XMLDoc
- Insufficient free space in CCATEMP

\$XML_DEL request cancel errors

Note that if any deleted nodes had been referenced in a nodelist, they may not subsequently be referenced using a relative XPath expression.

\$XML_DEL is new in version 6.4 of *Janus SOAP*.

4.7 **\$XML_DOC: Create XMLDoc**

\$XML_DOC creates an XMLDoc.

```
%doc = $XML_DOC
```

\$XML_DOC has no arguments

\$XML_DOC does not take any arguments. It returns the ID of an EMPTY XMLDoc (that is, it contains only a DOCUMENT node). An XMLDoc ID is always a positive number.

Each invocation of any given occurrence of this \$function will return the same XMLDoc ID. When an occurrence is re-invoked, the XMLDoc is EMPTY'd (see "[\\$XML_INITDOC: EMPTY XMLDoc and FREE its nodelists](#)" on page 50).

```
$XML_DOC does not have any request cancel errors
```

\$XML_DOC errors

4.8 **\$XML_DOC_GLOBAL and \$XML_DOC_SESSION: Session/global XMLDocs**

\$XML_DOC_GLOBAL creates or retrieves a global XMLDoc. \$XML_DOC_SESSION creates or retrieves a session XMLDoc.

```
%doc = $XML_DOC_GLOBAL(name, option)
%doc = $XML_DOC_SESSION(name, option)
```

name Name of the global or session XMLDoc; this may not be the null string

option One of the following keywords, possibly with the modifier PREP or INIT:

- OLD
This retrieves a global or session XMLDoc that was created with the same name; the XMLDoc must exist
- NEW
This creates a global or session XMLDoc with the indicated name; no XMLDoc with that name may exist
- ANY
This creates a global or session XMLDoc with the indicated name; if it doesn't already exist; if it does, it retrieves the global XMLDoc with that name

This argument is optional and defaults to ANY. You can modify the keyword with the prefix or suffix PREP or INIT, which indicates that the XMLDoc should be initialized (set to the EMPTY state) (of course, this is superfluous with NEW or ANY if the global XMLDoc doesn't exist). The following example will create a global EMPTY XMLDoc named INPUT_DOC, whether or not it already exists:

```
%DOC = $XML_DOC_GLOBAL('INPUT_DOC', 'INITANY')
```

\$XML_DOC_GLOBAL and \$XML_DOC_SESSION arguments

\$XML_DOC_GLOBAL and \$XML_DOC_SESSION take two arguments and return the ID of an XMLDoc, which is always a positive number.

Each invocation of a given occurrence of these \$functions will return the same XMLDoc ID. So multiple individual invocations of these functions will not allow you to access multiple global or session XMLDocs in a single request at the same time.

For a different request to access that same global or session XMLDoc, it must issue a \$XML_DOC_GLOBAL or \$XML_DOC_SESSION, respectively, with the same name. Separate \$XML_DOC_GLOBAL and \$XML_DOC_SESSION invocations can be issued against the same name. When this is done, each instance of \$XML_DOC_GLOBAL or \$XML_DOC_SESSION will return a separate XMLDoc identifier but they will actually refer to the same underlying XMLDoc. For example, after

```
%ONE = $XML_DOC_GLOBAL('WHATEVER', 'NEW')
%TWO = $XML_DOC_GLOBAL('WHATEVER')
%X = $XML_INSCOM(%ONE, 'WASTE NOT', '/')
%X = $XML_INSCOM(%TWO, 'WANT NOT', '/')
```

there'd be a single global XMLDoc with name WHATEVER that contains two comments, the first WASTE NOT and the second WANT NOT.

The content (that is, the nodes) of an XMLDoc is only stored once, in the underlying XMLDoc. Therefore the second call to \$XML_STR2DOC in the following cancels the request because it attempts to deserialize into a non-EMPTY XMLDoc:

```
%ONE = $XML_DOC_GLOBAL('WHATEVER', 'NEW')
%TWO = $XML_DOC_GLOBAL('WHATEVER')
%RC = $XML_STR2DOC(%ONE, %S)
%RC = $XML_STR2DOC(%TWO, %T)
```

The nodelists associated with different XMLDoc identifiers are kept separate, whether or not the XMLDoc identifiers refer to the same underlying XMLDoc. In the following example, the \$XML_INITDOC does not FREE nodelist %N1:

```
%DOC1 = $XML_DOC_GLOBAL('WHATEVER', 'NEW')
%DOC2 = $XML_DOC_GLOBAL('WHATEVER')
%N1 = $XML_NL
...
%X = $XML_NODES(%N1, %DOC1, ...)
...
%X = $XML_INITDOC(%DOC2)
```

- Argument two (*option*) invalid
 - *Option* NEW and global or session XMLDoc already exists
 - *Option* OLD and global or session XMLDoc doesn't exist
 - Session not open for \$XML_DOC_SESSION

\$XML_DOC_GLOBAL and \$XML_DOC_SESSION request cancel errors

\$XML_DOC_GLOBAL and \$XML_DOC_SESSION have independent namespaces. That is, the same name used for \$XML_DOC_GLOBAL and \$XML_DOC_SESSION reference different XMLDocs. A \$XML_DOC_SESSION call when there is no session open causes a request cancellation. For more information about sessions see the ***Sirius Functions Reference Manual***.

To delete a global or session XMLDoc, see [“\\$XML_DOC_GLOBAL_DEL and \\$XML_DOC_SESSION” on page 42](#).

\$XML_DOC_SESSION is new in version 6.3 of the *Sirius Mods*.

4.9 **\$XML_DOC_GLOBAL_DEL and \$XML_DOC_SESSION_DEL**

`$XML_DOC_GLOBAL_DEL` deletes a global XMLDoc. `$XML_DOC_SESSION_DEL` deletes a session XMLDoc.

```
%count = $XML_DOC_GLOBAL_DEL(name_pattern)
%count = $XML_DOC_SESSION_DEL(name_pattern)

name_pattern  A pattern which specifies the matching names
               of global or session XMLDocs to delete.
               This may not be the null string.
```

\$XML_DOC_GLOBAL_DEL and \$XML_DOC_SESSION_DEL arguments

`$XML_DOC_GLOBAL_DEL` and `$XML_DOC_SESSION_DEL` take one argument and return the number of global or session XMLDocs which matched argument one (***name_pattern***) and which have been deleted.

The name specified for `$XML_DOC_GLOBAL_DEL` or `$XML_DOC_SESSION_DEL` can be an explicit name or it can contain the following wildcard characters:

- * Matches any number of characters including none
- ? Matches any single character
- " Indicates that the next character must be treated literally even if it is a wildcard character.

For example,

```
%RC = $XML_DOC_GLOBAL_DEL('TYRANNOSAURUS')
```

would only delete a global XMLDoc named "TYRANNOSAURUS".

```
%RC = $XML_DOC_GLOBAL_DEL('STEG*')
```

would delete global XMLDocs named "STEG", "STEGOSAURUS" and "STEG.DATA" if they existed.

```
%RC = $XML_DOC_GLOBAL_DEL('ST??')
```

would delete global XMLDocs named "STAN", "STEP" and "STUN.DATA" if they existed.

```
%RC = $XML_DOC_GLOBAL_DEL('***')
```

would delete globals \$lists named “*”, “*LOOK” and “*ZAP.DATA” if they existed.

```
%RC = $XML_DOC_GLOBAL_DEL('*')
```

would delete all global \$lists.

When a \$XML_DOC_GLOBAL_DEL or \$XML_DOC_SESSION_DEL is issued against a global or session XMLDoc, respectively, that has a XMLDoc identifier associated with it, that XMLDoc identifier is INITIALIZED. For example, after

```
%DOC = $XML_DOC_GLOBAL('VELOCIRAPTOR')  
%RC = $XML_DOC_GLOBAL_DEL('VEL*')
```

the underlying XMLDoc contains only the ROOT node, and any nodelists associated with %DOC are FREEd.

- Argument one (*name_pattern*) missing or equal to the null string
 - Session not open for \$XML_DOC_SESSION_DEL

\$XML_DOC_GLOBAL_DEL and \$XML_DOC_SESSION_DEL request cancel errors

\$XML_DOC_GLOBAL_DEL will not delete session XMLDocs and \$XML_DOC_SESSION_DEL will not delete global XMLDocs. A \$XML_DOC_SESSION_DEL call when there is no session open causes a request cancellation. For more information about sessions see the *Sirius Functions Reference Manual*.

To create a global or session XMLDoc, see “\$XML_DOC_GLOBAL and \$XML_DOC_SESSION: Session/global XMLDocs” on page 39.

4.10 \$XML_DOCID: Get ID of XMLDoc associated with nodelist

\$XML_DOCID gets the XMLDoc ID associated with a nodelist.

```
%doc = $XML_DOCID(n1)  
  
n1 Identifier of nodelist.
```

\$XML_DOCID arguments

\$XML_DOCID takes one argument and returns the ID of the XMLDoc associated with nodelist *nl* or, if *nl* is FREE, it returns zero. It can be used when necessary with functions such as “[\\$XML_INFO: Get XMLDoc option setting](#)” on page 49 to refer to an XMLDoc, given a nodelist. It is not necessary to use \$XML_DOCID merely to use a nodelist with an absolute XPath expression; the nodelist itself can be used for that. For example,

```
%N = $XML_NL
...
%V = $XML_VAL(%N, '//customer_name')
```

can be used, rather than

```
%N = $XML_NL
...
%V = $XML_VAL($XML_DOCID(%N), '//customer_name')
```

- Argument one (*nl*) not a valid nodelist

\$XML_DOCID request cancel errors

4.11 **\$XML_DOC2STR: Serialize XMLDoc as UTF-8 string**

\$XML_DOC2STR is used to convert an XMLDoc to the UTF-8 text string representation of the XML document. (This process is called **serialization**, because the text representation of a document is called the **serial** form.)

```
%UTF8_str = $XML_DOC2STR(doc, option)
```

doc Identifier of XMLDoc to be serialized.

option Can be one of the following:

- ALLOWXMLDECL | NOXMLDECL
ALLOWXMLDECL, which is the default, indicates that the “<?xml version=...?>” declaration is produced if it had been set (see “[\\$XML_SET: Change XMLDoc option setting](#)” on page 80).
NOXMLDECL, indicates that the “<?xml version=...?>” declaration is not produced.
- NOEMPTYELT
This indicates that an empty element is serialized with start tag followed by an end tag, for example:

```
<middle_name>  
</middle_name>
```

If this option is not specified, the default is to serialize an empty element with an empty element tag; using the same example as above, this would be:

```
<middle_name/>
```

\$XML_DOC2STR arguments

\$XML_DOC2STR takes two arguments and returns the UTF-8 text string representation of the XML document. The XMLDoc argument must be WELL-FORMED (that is, it must contain an ELEMENT node).

Note that since the result of \$XML_DOC2STR uses the UTF-8 encoding, you can not treat it as an EBCDIC string; for example, the PRINT statement will not produce the displayable character contents of the string. Use “[\\$XML_PRINT: Print XMLDoc or sub-tree](#)” on page 75 to display an XML document for debugging, or “[\\$XML_SER: Serialize sub-tree as string](#)” on page 78 with the EBCDIC option to obtain an EBCDIC serialization of an XML document.

If you wish to serialize an XMLDoc and send it as an HTTP response using *Janus Web Server*, use \$WEB_XML_SEND, as documented in the ***Janus Web Server Reference Manual***. For other transport APIs, such as *Janus Sockets* or *Model 204 MQ Series*, \$XML_DOC2STR can be used to serialize an XMLDoc which can then be sent on the transport API.

For example, the following fragment can be used to serialize an XMLDoc to be sent with *Janus Sockets* to an XML server as a POST request using the HTTP client protocol:

```
JANUS DEFINE XMLREQ * CLSOCK 5 REMOTE * *
JANUS START XMLREQ
BEGIN
%S LONGSTRING
* ... create request document in XMLDoc %D
%X = $SOCK_CONN('XMLREQ', -
  'sirius-software.com', 80)
%S = $XML_DOC2STR(%D)
%R = $SOCK_SET(%X, 'LINEND', '0D0A')
%R = $SOCK_CAPTURE(%X, 'ON')
TEXT
POST /{%PATH} HTTP/1.1
Content-Length: {$LSTR_LEN(%S)}

END TEXT
%R = $SOCK_CAPTURE(%X, 'OFF')
%R = $SOCK_SEND(%X, %S, 'BINARY')
```

Notes:

- Depending on the requirements of the HTTP server you are sending to, you will need to determine the contents of the %PATH variable, and you may need to add HTTP request header lines in addition to “Content-Length”.
- Be sure to include the null line in the above TEXT block; this indicates the end of the HTTP header.
- As described in the *Janus Sockets Reference Manual*, in order to use a CLSOCK port as above, you need to have been authorized by a JANUS CLSOCK rule.

- Argument one (*id*) not WELL-FORMED XMLDoc ID
 - Argument two (*option*) not a valid option
 - Insufficient free space in CCATEMP

\$XML_DOC2STR request cancel errors

The string deserialization \$function is “[\\$XML_STR2DOC: Deserialize string into XMLDoc](#)” on page 83. Note that you can use “[\\$XML_PRINT: Print XMLDoc or sub-tree](#)” on page 75 to display a document on the terminal, or to **capture** a displayable version of a document, but \$XML_PRINT is used to insert line breaks and optional indentation, which may not be an accurate deserialization of an XMLDoc.

4.12 \$XML_EXIST: Test for non-empty XPath result

\$XML_EXIST tests whether the result of its argument XPath expression has any nodes.

```
%exist = $XML_EXIST(id, XPath_expr, contx_n)
```

id	Identifier of XMLDoc or non-FREE nodelist. For relative <i>XPath_expr</i> : <ul style="list-style-type: none">• If <i>id</i> is an XMLDoc ID, the DOCUMENT node is the context.• If <i>id</i> is a nodelist ID, it must be non-EMPTY, and the first item is the context.
XPath_expr	An XPath expression which results in a nodelist, which is tested to determine whether it has any nodes. Optional: default is '.', that is, the context node.
contx_n	Position of context node in context nodelist or XMLDoc. Must be: <ul style="list-style-type: none">• 1 if <i>id</i> is an XMLDoc ID;• >= 1 and <= number of nodes in nodelist if <i>XPath_expr</i> is relative and <i>id</i> is a nodelist ID. Optional: default is 1.

\$XML_EXIST arguments

\$XML_EXIST takes three arguments and returns 1 if the number of nodes in the argument XPath result is non-zero; otherwise it returns 0.

You can use \$XML_VAL to look for a null node value, but note that it does not distinguish between a case in which the result of the XPath argument is the empty node set, and a null string value. \$XML_EXIST can be used for this.

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*XPath_expr*) invalid or missing
- Argument three (*contx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- *XPath_expr* relative, and nodelist *id* item number *contx_n* references a node which has been deleted from the XMLDoc
- Insufficient free space in CCATEMP

\$XML_EXIST request cancel errors

Note that, as with all \$functions that have an XPath expression argument, the nodelist must not be EMPTY if a relative XPath expression is used. If you need to test whether a **nodelist** is non-EMPTY, use [“\\$XML_COUNT_NL: Get number of nodes in nodelist” on page 35](#).

Also note that the *XPath_expr* argument is required, which is not the case with most \$XML_xxx functions - usually the default is '.', that is, the context node. However, using the context node for *XPath_expr* would cause this \$function to always return 1, unless the context node does not exist, which would cancel the request.

4.13 **\$XML_FREENL: FREE nodelist**

\$XML_FREENL FREES a nodelist.

<pre>%n1 = \$XML_FREENL(n1) n1 Identifier of nodelist to FREE.</pre>
--

\$XML_FREENL arguments

\$XML_FREENL takes one argument and returns the value of that argument (*nl*).

A FREE nodelist can be passed as the argument to \$XML_FREENL, and can be passed as the “output” or “target” arguments of the following functions:

- \$XML_INSELT
- \$XML_NODES
- \$XML_PLC
- \$XML_RMV_NL

Passing a FREE nodelist as “input”, that is, as other arguments to the above functions, or to other \$XML_xxx functions, will cancel the request.

Except for preparing an initial FREE nodelist when using [“\\$XML_PLC: Place nodes from XPath result onto a nodelist” on page 73](#), you usually won't use \$XML_FREENL, because its only other purpose is to release CCATEMP storage used by nodelists. It isn't necessary at the end of a request, because all nodelists are automatically FREEd then.

A general situation which may merit the use of \$XML_FREENL is when a nodelist is being used temporarily in a block of code (e.g., a subroutine); when you are finished using such a nodelist, and you either have very many moderately large nodelists concurrently with un-needed items, or you have very large nodelists, \$XML_FREENL will free up some CCATEMP space. Reusing the nodelist, e.g., as the target in another call to \$XML_NODES, and various other \$XML_xxx functions, will automatically FREE it;

you do not need to call `$XML_FREENL` in those cases, but there should be no measureable cost of doing so, either.

- Argument one (*id*) not valid nodelist ID

\$XML_FREENL request cancel errors

4.14 \$XML_INFO: Get XMLDoc option setting

`$XML_INFO` gets information about an XMLDoc, that is, one of the option settings that can be changed with `$XML_SET`.

```
%info = $XML_INFO(doc, option)

doc      Identifier of XMLDoc with the requested
         information.

option   One of the valid XMLDoc options that are shown in
         “$XML_SET: Change XMLDoc option setting” on page
         80.
```

\$XML_INFO arguments

`$XML_INFO` takes two arguments and returns the requested option setting of the XMLDoc.

- Argument one (*id*) not a valid XMLDoc
- Argument two (*option*) not a valid option

\$XML_INFO request cancel errors

See “[\\$XML_NSINFO: Get namespace association for XPath prefix](#)” on page 72 to obtain the current URI associated with a prefix; this is a special form of XMLDoc setting.

4.15 **\$XML_INITDOC: EMPTY XMLDoc and FREE its nodelists**

\$XML_INITDOC EMPTYS an XMLDoc, removes any of its option settings and namespace associations, and FREEs all nodelists associated with it.

```
%doc = $XML_INITDOC(doc)

doc Identifier of XMLDoc to EMPTY.
```

\$XML_INITDOC arguments

\$XML_INITDOC takes one argument and returns the value of that argument (**doc**).

The ID of an EMPTY XMLDoc can be passed to most \$XML_xxx functions which accept an XMLDoc ID argument, except in those cases (such as \$XML_PRINT(doc, '/')) which require a WELL-FORMED XMLDoc. Your application may require you to “start over” and create multiple XMLDocs; you can accomplish this either by simply re-executing \$XML_DOC or with \$XML_INITDOC. Except for these odd cases, you usually won't use \$XML_INITDOC because its only other purpose is to release CCATemp storage used by XMLDocs. It isn't necessary at the end of a request, because all XMLDocs are automatically EMPTyd then (along with FREEing all nodelists).

- Argument one (*id*) not valid XMLDoc ID

\$XML_INITDOC request cancel errors

4.16 **\$XML_INSATT: Insert attribute**

\$XML_INSATT inserts either an ATTRIBUTE or NSATTRIBUTE of the head of the argument XPath result.

```
%id = $XML_INSATT(id, name, value, -  
  XPath_expr, ctx_n)
```

id	Identifier of XMLDoc or non-FREE nodelist. For relative <i>XPath_expr</i> : <ul style="list-style-type: none">• If <i>id</i> is an XMLDoc ID, the DOCUMENT node is the context.• If <i>id</i> is a nodelist ID, it must be non-EMPTY, and the first item is the context.
name	Attribute name; string. If <i>name</i> begins with the string “ <i>xmlns</i> ”, it is handled as specified in “ Inserting xmlns attribute names ” on page 52; otherwise a node of type ATTRIBUTE is inserted.
value	Attribute value; string; note that this value is stored, without any normalization, entity substitution, etc.
XPath_expr	An XPath expression which results in a nodelist, the head of which is the parent of the inserted attribute node. Optional: default is '.', that is, the context node.
ctx_n	Position of context node in context nodelist or XMLDoc. Must be: <ul style="list-style-type: none">• 1 if <i>id</i> is an XMLDoc ID;• >= 1 and <= number of nodes in nodelist if <i>XPath_expr</i> is relative and <i>id</i> is a nodelist ID. Optional: default is 1.

\$XML_INSATT arguments

\$XML_INSATT takes five arguments and returns the value of its first argument (*id*).

For example, the following fragment adds the attribute named “option” to the top level element:

```
%I = $XML_INSATT(%X, 'option', 'abc', '/*')
```

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*att_name*) violates rules for XML attribute name, or attribute with same name already present in head of argument XPath result
- *Att_name* begins with `xmlns` and matches the prefix of the name of the parent of the inserted attribute (see “[Inserting xmlns attribute names](#)” on page 52). This error only applies if the NAMESPACE setting of the XMLDoc is ON.
- Argument three (*value*) missing
- Argument four (*XPath_expr*) invalid
- Argument five (*contx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- Head of argument XPath result invalid node type - it must be either:
 - an ELEMENT node
- Insufficient free space in CCATEMP

\$XML_INSATT request cancel errors

Notes:

- \$XML_INSATT has no ***child_num*** argument; the order in which attributes are stored is not under the application's control.
- Various processing of an XMLDoc may be more efficient if the nodes in an XMLDoc are inserted in document order.

There are other \$functions for inserting XML nodes; see:

- “[\\$XML_INSCOM: Insert COMMENT node](#)” on page 53
- “[\\$XML_INSELT: Insert ELEMENT node](#)” on page 55
- “[\\$XML_INSPI: Insert PI node](#)” on page 59
- “[\\$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child](#)” on page 61

4.16.1 Inserting xmlns attribute names

If an insert operation of an attribute is performed, and the name begins with the string “`xmlns`”, then the name must either be simply `xmlns` or it must be of the form `xmlns:prefix`. The result of the insertion depends on the XMLDoc's NAMESPACE setting:

- ON** A node of type NSATTRIBUTE is inserted; the local name of the inserted NSATTRIBUTE node is *prefix*, if that form of name is used, or is the null string, indicating the default namespace.
- IGNORE** A node of type ATTRIBUTE is inserted. When NAMESPACE IGNORE is in effect, local names cannot be accessed; see [“NAMESPACE: set URI for prefix in XPath expressions” on page 82](#) for the various \$XML_xxx restrictions when NAMESPACE is set to IGNORE.
- NONE** Inserting such a node is disallowed.

With either NAMESPACE ON or IGNORE, the expanded name (QNAME) of the inserted NSATTRIBUTE node is the name of the insert operation (“xmlns” or “xmlns:*prefix*”).

The above insert operations are accomplished either with the \$XML_INSATT function or by using the *URI* argument of any of the following:

- [“\\$XML_INSELT: Insert ELEMENT node” on page 55](#)
- [“\\$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child” on page 61](#)

The *URI* argument of these functions is provided to satisfy *Janus SOAP*'s check, when the NAMESPACE setting is ON, that every prefix is declared. When you insert an ELEMENT with a prefixed name, if the prefix is not declared by virtue of appearing as a NSATTRIBUTE node of an ancestor, then *Janus SOAP* would detect an undeclared namespace prefix, if the *URI* argument is not used. The *URI* argument accomplishes the insertion of the “xmlns” attribute node at the same time as the insertion of an ELEMENT node, avoiding this error.

In order to simplify the rules, insertion of an element name's prefix declaration, when that declaration is done within the element itself, can only be accomplished with the *URI* argument. It is thus an error to attempt to use \$XML_INSATT if the *att_name* argument begins with `xmlns` and it “matches” the prefix of the parent of the inserted attribute. By “matches”, we mean that either the name is exactly `xmlns` and the parent is unprefixes, or the name is `xmlns:prefix` and the prefix of the parent is *prefix*. This attempt is an error regardless of the NAMESPACE setting of the XMLDoc.

4.17 **\$XML_INSCOM: Insert COMMENT node**

\$XML_INSCOM inserts a COMMENT node as a child at a specified position among the children of the head of the argument XPath result.

```
%id = $XML_INSCOM(id, value, -  
  XPath_expr, contx_n, child_num)
```

id	Identifier of XMLDoc or non-FREE nodelist. For relative <i>XPath_expr</i> : <ul style="list-style-type: none">• If <i>id</i> is an XMLDoc ID, the DOCUMENT node is the context.• If <i>id</i> is a nodelist ID, it must be non-EMPTY, and the first item is the context.
value	String-value of the comment; string.
XPath_expr	An XPath expression which results in a nodelist, the head of which is the parent of the inserted COMMENT node. Optional: default is '.', that is, the context node.
contx_n	Position of context node in context nodelist or XMLDoc. Must be: <ul style="list-style-type: none">• 1 if <i>id</i> is an XMLDoc ID;• ≥ 1 and \leq number of nodes in nodelist if <i>XPath_expr</i> is relative and <i>id</i> is a nodelist ID. Optional: default is 1.
child_num	Number of existing (non-attribute, of course) child to insert before, or -1; for example, with the following values of <i>child_num</i> , the node will be inserted: -1: after last (non-attr) child 1: before first (non-attr) child 2: before second (non-attr) child etc... This argument is optional: default is -1. It must be less than two plus the number of (non-attr) children of the head of the argument XPath result.

\$XML_INSCOM arguments

\$XML_INSCOM takes five arguments and returns the value of its first argument (*id*).

For example, the following fragment adds the comment “call home” as the last markup in the XML document:

```
%I = $XML_INSCOM(%X, 'call home', '/')
```

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*value*) missing or not a valid XML comment
- Argument three (*XPath_expr*) invalid
- Argument four (*contx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- Head of argument XPath result neither an ELEMENT nor DOCUMENT node
- Argument five (*child_num*) < 1 and not -1, or greater than one plus the number of (non-attr) children of head of argument XPath result
- Insufficient free space in CCATEMP

\$XML_INSCOM request cancel errors

Notes:

- Various processing of an XMLDoc may be more efficient if the nodes in an XMLDoc are inserted in document order.

There are other \$functions for inserting XML nodes; see:

- [“\\$XML_INSATT: Insert attribute” on page 50](#)
- [“\\$XML_INSELT: Insert ELEMENT node”](#)
- [“\\$XML_INSPI: Insert PI node” on page 59](#)
- [“\\$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child” on page 61](#)

4.18 \$XML_INSELT: Insert ELEMENT node

\$XML_INSELT inserts an ELEMENT node as a child at a specified position among the children of the head of the argument XPath result, and can set a target nodelist argument to the inserted ELEMENT. It allows you to insert a TEXT node as the child of the inserted ELEMENT, and to insert a NSATTRIBUTE (or ATTRIBUTE node named xmlns...) of the ELEMENT.

```
%id = $XML_INSELT(targn1, contxid, -  
elt_name, value, -  
XPath_expr, contx_n, URI, child_num)
```

targn1 Identifier of target nodelist to be set to point to the inserted node. This is optional.

contxid Identifier of XMLDoc or non-FREE context nodelist. For relative *XPath_expr*:

- If *contxid* is an XMLDoc ID, the DOCUMENT node is the context.
- If *contxid* is a nodelist ID, it must be non-EMPTY, and the first item is the context.

Optional: default is *targn1*; either *targn1* or *contxid* is required. If *contxid* is missing, *targn1* is used for context, and the above conditions apply to it.

elt_name Element name; string.

value Content of the inserted TEXT node; string; note that this value is stored, without any normalization, entity substitution, etc. Optional; if provided then a TEXT node with this content is inserted as the child of the inserted ELEMENT. Note that TEXT nodes cannot contain the null string, so this argument is ignored if it is the null string.

XPath_expr An XPath expression which results in a nodelist, the head of which is the parent of the inserted ELEMENT node. Optional: default is '.', that is, the context node.

contx_n Position of context node in context nodelist or XMLDoc. Must be:

- 1 if context is an XMLDoc;
- ≥ 1 and \leq number of nodes in nodelist if *XPath_expr* is relative and context is a nodelist.

Optional: default is 1.

URI URI of the inserted element's namespace; this argument is optional and defaults to the null string. If a non-null value is provided, then this \$function inserts, as a sub-node of the inserted element, an attribute node with value *URI*. The name in the attribute insertion operation depends on the form of the name of the inserted element (argument 3,

elt_name):

- If *elt_name* has a prefix followed by a colon, then the name in the attribute insertion operation is “*xmlns:prefix*”.
- Otherwise, the name in the attribute insertion operation is “*xmlns*”, for the default namespace.

See “[Inserting xmlns attribute names](#)” on page 52.

child_num Number of existing (non-attribute, of course) child to insert before, or -1; for example, with the following values of *child_num*, the node will be inserted:

-1: after last (non-attr) child
1: before first (non-attr) child
2: before second (non-attr) child
etc...

This argument is optional: default is -1. It must be less than two plus the number of (non-attr) children of the head of the argument XPath result.

\$XML_INSELT arguments

\$XML_INSELT takes eight arguments and returns the value of its first argument (*targnl*), or returns zero if the first argument is omitted. *Targnl*'s nodelist is FREEed after the XPath result is obtained (in case *contxid* is omitted or is the same nodelist identifier).

Usually, the only reason to supply the *value* argument is if you want to insert an ELEMENT node with a TEXT child **and** you want to insert one or more attributes of the ELEMENT. For example:

```
%N = $XML_NL
%T = $XML_INSELT(%N, %NL, 'weight', '10', '/')
%T = $XML_INSATT(%N, 'unit', 'gram')
%T = $XML_FREENL(%N)
```

If you want to insert an ELEMENT node with a TEXT child and you **do not** want to insert any attributes, use the \$XML_INSTEXT function with its optional *elt_name* argument (see “[\\$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child](#)” on page 61). For example:

```
%T = $XML_INSTEXT(%NL, 'weight', '10g')
```

\$XML_INSTEXT does not require the additional nodelist that \$XML_INSELT requires for its target nodelist argument.

- Argument one (*targnl*), if present, not valid nodelist ID
- Argument two (*contxid*), if present, neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist; if *contxid* is missing, *targnl* is used for context, and these conditions are checked for it
- Both *targnl* and *contxid* missing
- Argument three (*elt_name*) violates rules for XML element name
- Argument five (*XPath_expr*) invalid
- Argument six (*contx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- Head of argument XPath result neither an ELEMENT nor DOCUMENT node or it is a DOCUMENT which already has an ELEMENT
- Argument seven (*URI*) invalid, or not-null when the XMLDoc's NAMESPACE setting is anything except ON
- Argument eight (*child_num*) < 1 and not -1, or greater than one plus the number of (non-attr) children of head of argument XPath result
- Insufficient free space in CCATEMP

\$XML_INSELT request cancel errors

Notes:

- [“\\$XML_NL: Create nodelist” on page 67](#) describes the usual approach to creating a nodelist to be used for the *targnl* argument. Generally, it is a good idea to FREE the target nodelist when you are done processing it; see [“\\$XML_FREENL: FREE nodelist” on page 48](#).
- Various processing of an XMLDoc may be more efficient if the nodes in an XMLDoc are inserted in document order.

The following \$functions insert an ELEMENT node but do not have a target nodelist argument:

- [“\\$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child” on page 61](#) (which has an option to insert an ELEMENT together with a child TEXT node)

There are also other \$functions for inserting XML nodes; see:

- [“\\$XML_INSATT: Insert attribute” on page 50](#)
- [“\\$XML_INSCOM: Insert COMMENT node” on page 53](#)
- [“\\$XML_INSPI: Insert PI node” on page 59](#)

4.19 **\$XML_INSPI: Insert PI node**

\$XML_INSPI inserts a PI (processing instruction) node as a child at a specified position among the children of the head of the argument XPath result.

```
%id = $XML_INSPI(id, targ_name, value, -  
  XPath_expr, contx_n, child_num)
```

id	Identifier of XMLDoc or non-FREE nodelist. For relative <i>XPath_expr</i> : <ul style="list-style-type: none">• If <i>id</i> is an XMLDoc ID, the DOCUMENT node is the context.• If <i>id</i> is a nodelist ID, it must be non-EMPTY, and the first item is the context.
targ_name	Target of the processing instruction; string.
value	String-value of the processing instruction; string.
XPath_expr	An XPath expression which results in a nodelist, the head of which is the parent of the inserted PI node. Optional: default is '.', that is, the context node.
contx_n	Position of context node in context nodelist or XMLDoc. Must be: <ul style="list-style-type: none">• 1 if <i>id</i> is an XMLDoc ID;• ≥ 1 and \leq number of nodes in nodelist if <i>XPath_expr</i> is relative and <i>id</i> is a nodelist ID. Optional: default is 1.
child_num	Number of existing (non-attribute, of course) child to insert before, or -1; for example, with the following values of <i>child_num</i> , the node will be inserted: -1: after last (non-attr) child 1: before first (non-attr) child 2: before second (non-attr) child etc... This argument is optional: default is -1. It must be less than two plus the number of (non-attr) children of the head of the argument XPath result.

\$XML_INSPI arguments

\$XML_INSPI takes six arguments and returns the value of its first argument (*id*).

For example, the following fragment adds a PI as the first markup of the XML document:

```
%I = $XML_INSPI(%X, 'xml-styleSheet', -  
  'type="text/xsl" href="transform.xsl"', -  
  '/', , 1)
```

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*targ_name*) missing or violates rules for XML processing instruction target
- Argument three (*value*) missing or not a valid XML processing instruction value
- Argument four (*XPath_expr*) invalid
- Argument five (*ctx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- Head of argument XPath result neither an ELEMENT nor DOCUMENT node
- Argument six (*child_num*) < 1 and not -1, or greater than one plus the number of (non-attr) children of head of argument XPath result
- Insufficient free space in CCATEMP

\$XML_INSPI request cancel errors

Notes:

- Various processing of an XMLDoc may be more efficient if the nodes in an XMLDoc are inserted in document order.

There are other \$functions for inserting XML nodes; see:

- [“\\$XML_INSATT: Insert attribute” on page 50](#)
- [“\\$XML_INSCOM: Insert COMMENT node” on page 53](#)
- [“\\$XML_INSELT: Insert ELEMENT node” on page 55](#)
- [“\\$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child” on page 61](#)

4.20 \$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child

\$XML_INSTEXT inserts either a TEXT node or an ELEMENT with a TEXT child, as a child at a specified position among the children of the head of the argument XPath result. If you're inserting an ELEMENT, it also allows you to insert a NSATTRIBUTE (or ATTRIBUTE node named xmlns...) of the ELEMENT.

```
%id = $XML_INSTEXT(id, elt_name, value, -  
  XPath_expr, contx_n, URI, child_num)
```

id Identifier of XMLDoc or non-FREE nodelist. For relative *XPath_expr*:

- If *id* is an XMLDoc ID, the DOCUMENT node is the context.
- If *id* is a nodelist ID, it must be non-EMPTY, and the first item is the context.

elt_name Element name; string. If this is provided as a non-null string, it is the name of the new ELEMENT node inserted into the XMLDoc as the parent of the inserted TEXT node. This argument is optional.

value Content of the TEXT node. Note that if this argument is the null string, then no TEXT node is inserted. This argument is optional and defaults to the null string.

XPath_expr An XPath expression which results in a nodelist, the head of which is the parent of the inserted ELEMENT, if *elt_name* is provided as a non-null string, or the parent of the inserted TEXT node otherwise. Optional: default is '.', that is, the context node.

contx_n Position of context node in context nodelist or XMLDoc. Must be:

- 1 if *id* is an XMLDoc ID;
- ≥ 1 and \leq number of nodes in nodelist if *XPath_expr* is relative and *id* is a nodelist ID.

Optional: default is 1.

URI URI of the inserted element's namespace; this argument is optional and defaults to the null string. If a non-null value is provided, then this \$function inserts, as a sub-node of the inserted element, an attribute node with value *URI*. The name in the attribute insertion operation depends on the form of the name of the inserted element (argument 3, *elt_name*):

- If *elt_name* has a prefix followed by a colon, then the name in the attribute insertion operation is "xmlns:*prefix*".
- Otherwise, the name in the attribute insertion operation is "xmlns", for the default namespace.

See ["Inserting xmlns attribute names"](#) on page

52.

`child_num` Number of existing (non-attribute, of course) child to insert before, or -1; for example, with the following values of `child_num`, the insertion will be performed:

-1: after last (non-attr) child
1: before first (non-attr) child
2: before second (non-attr) child
etc...

This argument is optional: default is -1. It must be less than two plus the number of (non-attr) children of the head of the argument XPath result.

\$XML_INSTEXT arguments

\$XML_INSTEXT takes seven arguments and returns the value of its first argument (*id*).

Note that if you insert a TEXT node adjacent to another TEXT node, rather than inserting a new node, the existing TEXT node has the new value concatenated either before or after the existing node's value. For example, the following fragment prints the string "supercalifragilisticxpialodocious":

```
%I = $XML_INSELT(%OUTL, %INL, 'phrase', -  
  'supercalifragilistic')  
%I = $XML_INSTEXT(%OUTL, -  
  'xpialodocious')  
PRINT $XML_VAL(%OUTL, 'text()')
```

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*elt_name*) violates rules for XML element name
- Argument four (*XPath_expr*) invalid
- Argument five (*contx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- If argument three (*elt_name*) is a non-null name, head of argument XPath result not an ELEMENT node
- If *elt_name* is omitted or null, head of argument XPath result not an ELEMENT node
- Argument six (*URI*) invalid, or *URI* is non-null and *elt_name* missing or null.
- Argument seven (*child_num*) < 1 and not -1, or greater than one plus the number of (non-attr) children of head of argument XPath result
- Insufficient free space in CCATEMP

\$XML_INSTEXT request cancel errors

Notes:

- Various processing of an XMLDoc may be more efficient if the nodes in an XMLDoc are inserted in document order.

The following \$functions also insert an ELEMENT node:

- [“\\$XML_INSELT: Insert ELEMENT node” on page 55](#)

There are other \$functions for inserting XML nodes; see:

- [“\\$XML_INSATT: Insert attribute” on page 50](#)
- [“\\$XML_INSCOM: Insert COMMENT node” on page 53](#)
- [“\\$XML_INSPI: Insert PI node” on page 59](#)

4.21 \$XML_LEN: Get length of string-value

\$XML_LEN gets the length of the string-value of a node, including its TEXT descendants.

```
%len = $XML_LEN(id, XPath_expr, contx_n)
```

id	Identifier of XMLDoc or non-FREE nodelist. For relative <i>XPath_expr</i> : <ul style="list-style-type: none">• If <i>id</i> is an XMLDoc ID, the DOCUMENT node is the context.• If <i>id</i> is a nodelist ID, it must be non-EMPTY, and the first item is the context.
XPath_expr	An XPath expression which results in a nodelist, the head of which is the node to process. Optional: default is '.', that is, the context node.
contx_n	Position of context node in context nodelist or XMLDoc. Must be: <ul style="list-style-type: none">• 1 if <i>id</i> is an XMLDoc ID;• >= 1 and <= number of nodes in nodelist if <i>XPath_expr</i> is relative and <i>id</i> is a nodelist ID. Optional: default is 1.

\$XML_LEN arguments

\$XML_LEN takes three arguments and returns the length of the string-value of the head of the argument XPath result. If the argument XPath result is empty, \$XML_LEN returns zero.

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*XPath_expr*) invalid
- Argument three (*contx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- *XPath_expr* relative, and nodelist *id* item number *contx_n* references a node which has been deleted from the XMLDoc
- Insufficient free space in CCATEMP

\$XML_LEN request cancel errors

See “[\\$XML_VAL: Get string-value of node](#)” on page 89 for a discussion of the **string-value** of various node types.

4.22 **\$XML_NAME: Obtain the name of a node**

\$XML_NAME gets name information about the head of the argument XPath result.

<code>%name = \$XML_NAME(id, XPath_expr, contx_n, - option)</code>	
<code>id</code>	Identifier of XMLDoc or non-FREE nodelist. For relative <i>XPath_expr</i> : <ul style="list-style-type: none">• If <i>id</i> is an XMLDoc ID, the DOCUMENT node is the context.• If <i>id</i> is a nodelist ID, it must be non-EMPTY, and the first item is the context.
<code>XPath_expr</code>	The XPath expression which results in a nodelist, the head of which is the node to process. Optional: default is '.', that is, the context node.
<code>contx_n</code>	Position of context node in context nodelist or XMLDoc. Must be: <ul style="list-style-type: none">• 1 if <i>id</i> is an XMLDoc ID;• ≥ 1 and \leq number of nodes in nodelist if <i>XPath_expr</i> is relative and <i>id</i> is a nodelist ID. Optional: default is 1.
<code>option</code>	One of the items from the list of options shown below, indicating the type of name to return. This argument is optional and defaults to 'QNAME', which is the qualified name as specified in the XML document

\$XML_NAME arguments

\$XML_NAME takes four arguments and returns the name (of the requested type) of the head of the argument XPath result. The fourth argument, ***option***, indicates the type of name to return; it can be one of the following:

LOCAL

- The local part of the element-type or attribute name, if the node is an ELEMENT or attribute, respectively. This does not include any namespace prefix or following colon.
- The processing instruction's target, if the node is a PI. (A PI node does not have a namespace prefix.)
- Other node types will return the null string.

This option may only be used if the NAMESPACE setting is ON (see “[\\$XML_SET: Change XMLDoc option setting](#)” on page 80).

QNAME The element-type, attribute name, or processing instruction's target, if the node is an ELEMENT, attribute, or PI, respectively; other node types will return the null string. If the name has a namespace prefix, this is included, as is the following colon. A PI node does not have a namespace prefix.

URI The namespace URI of the element-type or attribute name, if the node is an ELEMENT or attribute, respectively; other node types will return the null string. Note that namespace-aware applications should use the namespace URI rather than the prefix to identify elements.

This option may only be used if the NAMESPACE setting is ON (see “[\\$XML_SET: Change XMLDoc option setting](#)” on page 80).

If the argument XPath result is empty, \$XML_NAME returns the null string.

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*XPath_expr*) invalid
- Argument three (*ctx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- *XPath_expr* relative, and nodelist *id* item number *ctx_n* references a node which has been deleted from the XMLDoc
- Argument four (*option*) invalid
- Insufficient free space in CCATEMP

\$XML_NAME request cancel errors

4.23 \$XML_NL: Create nodelist

\$XML_NL creates a nodelist.

```
%n1 = $XML_NL
```

\$XML_NL has no arguments

\$XML_NL does not take any arguments. It returns the ID of a FREE nodelist (that is, it is not associated with any XMLDoc). A nodelist ID is always a positive number.

Each invocation of any given occurrence of this \$function will return the same nodelist ID. When an occurrence is re-invoked, the nodelist is FREEed (see “[\\$XML_FREEL: FREE nodelist](#)” on page 48).

\$XML_NL does not have any request cancel errors

\$XML_NL errors

4.24 \$XML_NODE_SING: Select singleton nodelist from XMLDoc

\$XML_NODE_SING sets a target nodelist argument to contain the first node selected by an XPath expression.

```
%count = $XML_NODE_SING(targn1, contxid, -
    XPath_expr, contx_n)
```

targn1 Identifier of target nodelist to be set to point to head of result of XPath expression.

contxid Identifier of XMLDoc or non-FREE context nodelist. For relative *XPath_expr*:

- If *contxid* is an XMLDoc ID, the DOCUMENT node is the context.
- If *contxid* is a nodelist ID, it must be non-EMPTY, and the first item is the context.

Optional: default is *targn1*; either *targn1* or *contxid* is required. If *contxid* is missing, *targn1* is used for context, and the above conditions apply to it.

XPath_expr An XPath expression; the *targn1*'s nodelist is set to the first item of this list. Optional: default is '.', that is, the context node.

contx_n Position of context node in context nodelist or XMLDoc. Must be:

- 1 if context is an XMLDoc;
- >= 1 and <= number of nodes in nodelist if *XPath_expr* is relative and context is a nodelist.

Optional: default is 1.

\$XML_NODE_SING arguments

\$XML_NODE_SING takes four arguments and returns one, if the XPath expression selects a node in the XMLDoc, or zero otherwise. *Targnl's* nodelist is FREEed after the XPath result is obtained (in case *contxid* is omitted or is the same nodelist identifier).

You can either re-use some nodelist for the *targnl* argument, or you can create one just for its use, with “[\\$XML_NL: Create nodelist](#)” on page 67.

- Argument one (*targnl*), if present, not valid nodelist ID
- Argument two (*contxid*), if present, neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist; if *contxid* is missing, *targnl* is used for context, and these conditions are checked for it
- Both *targnl* and *contxid* missing
- Argument three (*XPath_expr*) invalid
- Argument four (*contx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- *XPath_expr* relative, and nodelist *contxid* item number *contx_n* references a node which has been deleted from the XMLDoc
- Insufficient free space in CCATEMP

\$XML_NODE_SING request cancel errors

“[\\$XML_NODES: Select list of nodes within XMLDoc](#)” continues selecting nodes for the target nodelist, but otherwise is the same as \$XML_NODE_SING.

\$XML_NODE_SING is new in version 6.4 of *Janus SOAP*.

4.25 \$XML_NODES: Select list of nodes within XMLDoc

\$XML_NODES sets a target nodelist argument to contain a list of nodes as specified by an XPath expression.

```
%count = $XML_NODES(targn1, contxid, -  
    XPath_expr, contx_n)
```

targn1 Identifier of target nodelist to be set to point to result of XPath expression.

contxid Identifier of XMLDoc or non-FREE context nodelist. For relative *XPath_expr*:

- If *contxid* is an XMLDoc ID, the DOCUMENT node is the context.
- If *contxid* is a nodelist ID, it must be non-EMPTY, and the first item is the context.

Optional: default is *targn1*; either *targn1* or *contxid* is required. If *contxid* is missing, *targn1* is used for context, and the above conditions apply to it.

XPath_expr An XPath expression which results in a nodelist; the *targn1*'s nodelist is set to this list. Optional: default is '.', that is, the context node.

contx_n Position of context node in context nodelist or XMLDoc. Must be:

- 1 if context is an XMLDoc;
- ≥ 1 and \leq number of nodes in nodelist if *XPath_expr* is relative and context is a nodelist.

Optional: default is 1.

\$XML_NODES arguments

\$XML_NODES takes four arguments and returns the number of nodes in the XPath expression result. *Targn1*'s nodelist is FREEed after the XPath result is obtained (in case *contxid* is omitted or is the same nodelist identifier).

You can either re-use some nodelist for the *targn1* argument, or you can create one just for its use, with [“\\$XML_NL: Create nodelist” on page 67](#).

- Argument one (*targn1*), if present, not valid nodelist ID
- Argument two (*contxid*), if present, neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist; if *contxid* is missing, *targn1* is used for context, and these conditions are checked for it
- Both *targn1* and *contxid* missing
- Argument three (*XPath_expr*) invalid
- Argument four (*contx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- *XPath_expr* relative, and nodelist *contxid* item number *contx_n* references a node which has been deleted from the XMLDoc
- Insufficient free space in CCATEMP

\$XML_NODES request cancel errors

“[\\$XML_NODE_SING: Select singleton nodelist from XMLDoc](#)” on page 68 stops after selecting a single node for the target nodelist, but otherwise is the same as \$XML_NODES.

4.26 \$XML_NS: Set namespace association for XPath prefix

\$XML_NS creates an association between a prefix and a namespace URI, so that the prefix can be used in XPath expressions passed as *XPath_expr* arguments to \$XML_xxx functions.

```
%old = $XML_NS(doc, prefix, URI)
```

doc Identifier of XMLDoc for which the namespace association is done.

prefix A valid prefix, according to the XML Namespaces Recommendation.

URI A valid Universal Resource Identifier, according to the XML Namespaces Recommendation, which the prefix is to be associated with. May also be the null string, which indicates that the prefix no longer has a namespace association.

\$XML_NS arguments

\$XML_NS takes three arguments and returns the URI associated with the prefix prior to changing it.

Note that any prefix used in any *XPath_expr* argument to any \$XML_xxx function must have a non-null association established by \$XML_NS. The association is done for the entire XMLDoc, but it does not change any of the XML contained in the XMLDoc; it is only for the \$XML_xxx functions interpretation of *XPath_expr* arguments.

This \$function may only be used for an XMLDoc whose NAMESPACE setting is ON; see “[\\$XML_SET: Change XMLDoc option setting](#)” on page 80.

- The NAMESPACE setting of the XMLDoc is not ON
- Argument one (*doc*) not a valid XMLDoc
- Argument two (*prefix*) invalid
- Argument three (*URI*) invalid

\$XML_NS request cancel errors

See “[\\$XML_NSINFO: Get namespace association for XPath prefix](#)” to obtain the current URI associated with a prefix.

See “[\\$XML_SET: Change XMLDoc option setting](#)” on page 80 to set other properties of an XMLDoc.

Note: This \$function has not yet been implemented, as of version 6.4 of *Janus SOAP*.

4.27 \$XML_NSINFO: Get namespace association for XPath prefix

\$XML_NSINFO returns the current association between a prefix and a namespace URI; this association lets you use the prefix in XPath expressions passed as *XPath_expr* arguments to \$XML_xxx functions.

```
%curr = $XML_NSINFO(doc, prefix)

doc      Identifier of XMLDoc for which the namespace
         association is to be obtained.

prefix   A valid prefix, according to the XML Namespaces
         Recommendation.
```

\$XML_NSINFO arguments

\$XML_NSINFO takes two arguments and returns the URI associated with the prefix.

Note that any prefix used in any *XPath_expr* argument to any \$XML_xxx function must have a non-null association established by \$XML_NS (see “[\\$XML_NS: Set namespace association for XPath prefix](#)” on page 71). The association is for the entire XMLDoc, but it does not reflect any of the XML contained in the document; it is only for the \$XML_xxx functions interpretation of *XPath_expr* arguments.

This \$function may only be used for an XMLDoc whose NAMESPACE setting is ON; see “[\\$XML_SET: Change XMLDoc option setting](#)” on page 80.

- The NAMESPACE setting of the XMLDoc is not ON
 - Argument one (*doc*) not a valid XMLDoc
 - Argument two (*prefix*) invalid

\$XML_NSINFO request cancel errors

See “[\\$XML_INFO: Get XMLDoc option setting](#)” on page 49 to obtain other properties of an XMLDoc.

Note: This \$function has not yet been implemented, as of version 6.4 of *Janus SOAP*.

4.28 \$XML_PLC: Place nodes from XPath result onto a nodelist

\$XML_PLC merges the nodes from an argument XPath result onto an output nodelist argument. The output nodelist may be FREE, but if not and the context ID for the XPath result is provided, the output nodelist must be on the same XMLDoc as the context ID.

```
%count = $XML_PLC(outn1, contxid, XPath_expr)
```

outn1 Identifier of nodelist to which nodes are added.

contxid Identifier of XMLDoc or non-FREE context nodelist used to determine which nodes are added. For relative *XPath_expr*:

- If *contxid* is an XMLDoc ID, the DOCUMENT node is the context.
- If *contxid* is a nodelist ID, it must be non-EMPTY, and the first item is the context.

Optional: default is *outn1*; either *targn1* or *contxid* is required. If *contxid* is missing, *outn1* is used for context, and the above conditions apply to it.

XPath_expr An XPath expression which results in a nodelist which is merged onto *outn1*'s nodelist. Optional: default is '.', that is, the context node.

\$XML_PLC arguments

\$XML_PLC takes three arguments and returns the number of nodes on *outn1*'s nodelist, after merging any new nodes.

Considered as sets, \$XML_PLC performs the “union” or “OR” operation, and “subtraction” is performed by “[\\$XML_RMV_NL: Remove one nodelist from another](#)” on [page 78](#).

Other set operations can be constructed using these and various XPath expressions. For example, the set of all non-attribute, non-namespace nodes can be obtained using the XPath expression “//” and then, relative to this universe, the complement (“**NOT**”) of a (non-attribute) nodelist can be obtained:

```
%NOT = $XML_NL
%JUNK = $XML_NODES(%NOT, %DOC, '//')
%JUNK = $XML_RMV_NL(%NOT, %INPUT)
```

And the following obtains the intersection (“**ANDing**”) of two (non-attribute) nodelists using the tautology A AND B = A REMOVE NOT B:

```
%NOT = $XML_NL
%JUNK = $XML_NODES(%NOT, %DOC, '//')
%JUNK = $XML_RMV_NL(%NOT, %B)
%JUNK = $XML_RMV_NL(%A, %NOT)
```

- Argument one (*outnl*) not valid nodelist ID or, if argument two (*ctxid*) missing, *outnl* is FREE.
- Argument two (*ctxid*), if present, neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist; if *ctxid* is missing these conditions are checked for *outnl*
- *Outnl* on different underlying XMLDoc than *ctxid*
- Both *outnl* and *ctxid* missing
- Argument three (*XPath_expr*) invalid
- *XPath_expr* relative, and nodelist *ctxid* item number *ctx_n* references a node which has been deleted from the XMLDoc
- Insufficient free space in CCATEMP

\$XML_PLC request cancel errors

This \$function is supported starting with version 6.4 of *Janus SOAP*.

4.29 \$XML_PRINT: Print XMLDoc or sub-tree

\$XML_PRINT formats and displays the contents of an XMLDoc or XMLDoc sub-tree on the user's standard output device, which may be the *Model 204* user's terminal.

```
%junk = $XML_PRINT(id, XPath_expr, contx_n, -  
options)
```

- id** Identifier of XMLDoc or non-FREE nodelist.
For relative *XPath_expr*:
- If *id* is an XMLDoc ID, the DOCUMENT node is the context.
 - If *id* is a nodelist ID, it must be non-EMPTY, and the first item is the context.
- XPath_expr** An XPath expression which results in a nodelist, the head of which is the “top” node of the displayed sub-tree. Optional: default is '.', that is, the context node.
- contx_n** Position of context node in context nodelist or XMLDoc. Must be:
- 1 if *id* is an XMLDoc ID;
 - ≥ 1 and \leq number of nodes in nodelist if *XPath_expr* is relative and *id* is a nodelist ID.
- Optional: default is 1.
- options** A blank delimited string which can contain any combination of:
- ALLOWXMLDECL, which is the default, indicating the “<?xml version=...?>” declaration is produced if it had been set (see “[\\$XML_SET: Change XMLDoc option setting](#)” on page 80). This option is ignored if the node selected by the *XPath_expr* is not the DOCUMENT node. This option is over-ridden if the NOXMLDECL option follows it.
 - NOXMLDECL, indicating the “<?xml version=...?>” declaration is not produced. This option is ignored if the node selected by the *XPath_expr* is not the DOCUMENT node. This option is over-ridden if the ALLOWXMLDECL option follows it.
 - INDENT *n*, where *n* is a non-negative integer, indicating that children and attributes (and the closing “/>” of an empty Element) should be indented *n* spaces from the start of the display of the Element's Start-Tag. The default indent width is 3 spaces.
 - NOEMPTYELT

This indicates that an empty element is serialized with start tag followed by an end tag, for example:

```
<middle_name>  
</middle_name>
```

If this option is not specified, the default is to serialize an empty element with an empty element tag; using the same example as above, this would be:

```
<middle_name/>
```

\$XML_PRINT arguments

\$XML_PRINT takes four arguments and returns zero. If the entire XMLDoc is being displayed (for example, if the XPath expression is '/', selecting the XMLDoc DOCUMENT), then the XMLDoc must be WELL-FORMED (that is, it must contain an ELEMENT node), and the XMLDoc's "<?xml ...>" declaration ("<?xml version='1.0' ...?>") is displayed, if it exists and is allowed by the *option* argument.

\$XML_PRINT can be useful to display a document, or some part of it, usually for debugging purposes.

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*XPath_expr*) invalid
- Argument three (*ctx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- *XPath_expr* relative, and nodelist *id* item number *ctx_n* references a node which has been deleted from the XMLDoc
- Result of *XPath_expr* is DOCUMENT node, and XMLDoc does not have an ELEMENT node
- Argument four (*option*) not a valid option
- Insufficient free space in CCATEMP

\$XML_PRINT request cancel errors

If you wish to accurately serialize an XMLDoc, as opposed to a “displayable” form, use “[\\$XML_DOC2STR: Serialize XMLDoc as UTF-8 string](#)” on page 44. \$XML_PRINT is used to insert line breaks and to perform indentation.

4.30 **\$XML_RMV_NL: Remove one nodelist from another**

\$XML_RMV_NL removes from an output nodelist the nodes which are on another nodelist. The output nodelist may be FREE, but if not, the two nodelists must be on the same XMLDoc.

```
%count = $XML_RMV_NL(outnl, innl)
```

```
outnl  Identifier of nodelist from which nodes are  
        removed.
```

```
innl   Identifier of nodelist containing nodes which are  
        removed.
```

\$XML_RMV_NL arguments

\$XML_RMV_NL takes two arguments and returns the number of nodes on *outnl*'s nodelist, after removing any nodes.

\$XML_RMV_NL only affects the nodes identified on *outnl*'s nodelist; it does not actually remove any nodes from the XMLDoc.

The following example will EMPTY a nodelist:

```
%T = $XML_RMV_NL(%NL, %NL)
```

- Argument one (*outnl*) not valid nodelist ID
- Argument two (*innl*) not a non-FREE nodelist
- *Outnl* on different underlying XMLDoc than *innl*

\$XML_RMV_NL request cancel errors

This \$function is supported starting with version 6.4 of *Janus SOAP*.

4.31 **\$XML_SER: Serialize sub-tree as string**

\$XML_SER is used to convert a sub-tree of an XMLDoc to the UTF-8 or EBCDIC text string representation of the sub-tree. (This process is called **serialization**, because the text representation of a document is called the **serial** form.)

```
%str = $XML_SER(id, XPath_expr, contx_n, -  
options)
```

id	Identifier of XMLDoc or non-FREE nodelist. For relative <i>XPath_expr</i> : <ul style="list-style-type: none">• If <i>id</i> is an XMLDoc ID, the DOCUMENT node is the context.• If <i>id</i> is a nodelist ID, it must be non-EMPTY, and the first item is the context.
XPath_expr	An XPath expression which results in a nodelist, the head of which is the top node of the serialized sub-tree. Optional: default is '.', that is, the context node.
contx_n	Position of context node in context nodelist or XMLDoc. Must be: <ul style="list-style-type: none">• 1 if <i>id</i> is an XMLDoc ID;• >= 1 and <= number of nodes in nodelist if <i>XPath_expr</i> is relative and <i>id</i> is a nodelist ID. Optional: default is 1.
options	A blank delimited string which can contain any combination of: <ul style="list-style-type: none">• EBCDIC This indicates that the serialization should be in EBCDIC rather than UTF-8.• NOEMPTYELT This indicates that an empty element is serialized with start tag followed by an end tag, for example: <pre><middle_name> </middle_name></pre> If this option is not specified, the default is to serialize an empty element with an empty element tag; using the same example as above, this would be: <pre><middle_name/></pre>

\$XML_SER arguments

\$XML_SER takes four arguments and returns the UTF-8 or EBCDIC text string representation of the sub-tree starting at the head of the argument XPath result. Note that, unlike the other serialization \$functions, \$XML_SER never serializes the “XML declaration”.

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*XPath_expr*) invalid
- Argument three (*contx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- *XPath_expr* relative, and nodelist *id* item number *contx_n* references a node which has been deleted from the XMLDoc
- Argument four (*option*) not a valid option
- Insufficient free space in CCATEMP

\$XML_SER request cancel errors

If you wish to obtain a LONGSTRING which is the UTF-8 serialization of an entire XMLDoc, including the “XML declaration”, use “[\\$XML_DOC2STR: Serialize XMLDoc as UTF-8 string](#)” on page 44.

The EBCDIC option is new in version 6.4 of *Janus SOAP*.

4.32 \$XML_SET: Change XMLDoc option setting

\$XML_SET is used to change various option settings of an XMLDoc; XMLDoc option settings control the operation of various \$XML_xxx functions.

```
%old = $XML_SET(doc, option, value)
```

doc	Identifier of XMLDoc whose option is to be changed.
option	One of the items from the list of options shown below.
value	The value to set the option to; values are described for each of the options in the list below.

\$XML_SET arguments

\$XML_SET takes three arguments and returns the value of the requested setting prior to changing it.

Argument two (*option*) identifies which option setting to change; it can be one of the following:

NAMESPACE This setting determines namespace support in the XMLDoc, with the mapping, in qualified names, of namespace prefixes to namespace URIs. See [“NAMESPACE: set URI for prefix in XPath expressions” on page 82](#) for the values of the NAMESPACE setting. The NAMESPACE setting may not be changed if an XMLDoc contains any nodes other than a DOCUMENT node.

As stated in [“Restrictions” on page 97](#), NAMESPACE NONE is the default setting in version 6.4; NAMESPACE ON will be the default setting when full namespace processing is supported by *Janus SOAP*.

See [“Inserting xmlns attribute names” on page 52](#), which describes the dependency of \$XML_INSATT on the NAMESPACE setting.

VERSION This setting sets the XML **version** declaration of the XMLDoc; its value must be valid as specified in the *W3C XML Recommendation*, or it can be omitted or the null string, to “unset it”, indicating that the version declaration is not set for the XMLDoc. This value corresponds to the value specified in serialized form as “<?xml version=**value**...?>”. Note that VERSION may not be “unset” if either ENCODING or STANDALONE is set; also see note below.

ENCODING This setting sets the XML **encoding** declaration of the XMLDoc. This value corresponds to the value specified in serialized form as “<?xml version=...encoding=**value**...?>”. The values accepted are any upper/lower-case variations of the following:

- omitted or the null string, to indicate that the encoding declaration is not set for the XMLDoc
- UTF-8

Note that ENCODING may not be set to a non-null value if VERSION is “unset”; also see note below.

STANDALONE This setting sets the XML **standalone** declaration of the XMLDoc; its value must be valid as specified in the *W3C XML Recommendation*, or it can be omitted or the null string, to indicate that the standalone declaration is not set for the XMLDoc. This value corresponds to the value specified in serialized form as “<?xml version=...standalone=**value**?>”. Note that STANDALONE may not be set to a non-null value if VERSION is “unset”; also see note below.

Note:

For **VERSION**, **ENCODING**, and **STANDALONE**, although the “<?xml ...?>” declaration has the same appearance as a processing instruction, it is not a processing instruction. Also note that the serialization of the “<?xml ...?>” declaration is controlled by the *option* argument of the serialization \$function being used, such as “[\\$XML_DOC2STR: Serialize XMLDoc as UTF-8 string](#)” on page 44 or “[\\$XML_PRINT: Print XMLDoc or sub-tree](#)” on page 75. Finally, note that the “<?xml ...?>” declaration can be set for an XMLDoc either by \$XML_SET, or by one of the deserialization \$functions, such as “[\\$XML_STR2DOC: Deserialize string into XMLDoc](#)” on page 83.

- Argument one (*doc*) not a valid XMLDoc
- Argument two (*option*) invalid
- Argument three (*value*) invalid
- Attempt to change NAMESPACE setting and XMLDoc contains ELEMENT node

\$XML_SET request cancel errors

See “[\\$XML_NS: Set namespace association for XPath prefix](#)” on page 71 to set the current URI associated with a prefix; this is a special form of XMLDoc setting.

See “[\\$XML_INFO: Get XMLDoc option setting](#)” on page 49, to obtain the current value of the properties set by \$XML_SET.

4.32.1 NAMESPACE: set URI for prefix in XPath expressions

As stated in “[Restrictions](#)” on page 97, NAMESPACE NONE is the default setting in version 6.4; NAMESPACE ON will be the default setting when full namespace processing is supported by *Janus SOAP*.

The values of the NAMESPACE setting can be:

ON With this setting, the requirements imposed by the XML Namespaces Recommendation are followed by *Janus SOAP*. That is, wherever a prefix occurs in the name of an element or attribute in the document, a namespace declaration must be in effect for that prefix.

IGNORE With this setting, namespaces are not handled in any way by *Janus SOAP*; specifically:

- Every name in the document is treated as a character string, which may include a colon.
- A namespace declaration in the document is simply treated as an attribute, whose name begins with the characters “xmlns”.
- The request is cancelled if the namespace axis is used in an XPath expression.

- The request is cancelled if a non-null **URI** argument is passed to “\$XML_INSELT: Insert ELEMENT node” on page 55, or “\$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child” on page 61.
- Invoking the \$XML_NS or \$XML_NSINFO function causes a request cancellation.
- Invoking the \$XML_NAME function with a name type of LOCAL or URI causes a request cancellation.

NONE With this setting, namespaces are not allowed in a document; specifically:

- Any name including a colon is causes a request cancellation.
- The request is cancelled if the namespace axis is used in an XPath expression.
- The request is cancelled if a non-null **URI** argument is passed to “\$XML_INSELT: Insert ELEMENT node” on page 55, or “\$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child” on page 61.
- Invoking the \$XML_NS or \$XML_NSINFO function causes a request cancellation.
- Invoking the \$XML_NAME function with a name type of LOCAL or URI causes a request cancellation.

4.33 \$XML_STR2DOC: Deserialize string into XMLDoc

\$XML_STR2DOC is used to convert a text string representation of an XML document to an XMLDoc tree (this process is called **deserialization**, because the text representation of a document is called the **serial** form).

`%rc = $XML_STR2DOC(doc, string, option)`

doc Identifier of the EMPTY XMLDoc which is to be set to the deserialization of the string.

string The text string, to be deserialized into the XMLDoc.

option Can be any valid combination of the following:

- **ERRRET**, to indicate that errors during deserialization are tolerated, the request continues, and the XMLDoc remains EMPTY. If **ERRRET** is not present, any error will cancel the request. Note that some errors still cancel the request; errors tolerated when **ERRRET** is specified are explicitly noted in the table of cancellation errors below.
- **DTD_IGNORE**, to indicate that a “<!DOCTYPE ...>” clause may be present in the document, and that it should be ignored. As of version 6.4 of *Janus SOAP*, the DTD is not processed, so the default behavior, that is, if **DTD_IGNORE** is not present, is to treat “<!DOCTYPE ...>” as a syntax error.
- **WSP_NORM**, to indicate that whitespace in element content not “protected” by the `xml:space="preserve"` attribute, is normalized using the XPath `normalize()` function (leading and trailing whitespace removed, and intermediate strings of whitespace replaced by a single blank character)
- **WSP_PRESERVE**, to indicate that all whitespace characters in element content are preserved (after end-of-line normalization, of course, as specified in “[Normalized line-end](#)” on page 17)

Note:

- No option may be specified twice.
- **WSP_PRESERVE** and **WSP_NORM** are mutually exclusive; if neither is specified, **WSP_NORM** is in effect.
- By “protected” by the `xml:space="preserve"` attribute, we mean an element *E* which either:
 1. has the `xml:space` attribute with the value “preserve”;
 2. is contained in an element *A* with that attribute and value and there is no element

which is a descendent of *A* and an ancestor of *E* with the `xml:space` attribute with the value “default”.

- Since the `xml:space` attribute is not supported in version 6.4 of *Janus SOAP*, the effect in that version is that unless you specify `WSP_PRESERVE`, all `TEXT` nodes will have leading and trailing whitespace removed, and each sequence of whitespace collapsed to a single space character.
- There is no comparable whitespace normalization for the `$XML_INSxxx` functions which create a `TEXT` node (“`$XML_INSELT: Insert ELEMENT node`” on page 55 and “`$XML_INSTEXT: Insert TEXT node or ELEMENT with TEXT child`” on page 61).
- Whitespace normalization applies to the characters in the input serialized string, not the values after entity substitution; see the example below containing “` `”.
- In addition to reducing the space consumed by individual `TEXT` nodes, `WSP_NORM` will cause all-whitespace content between markup to be collapsed to the null string, and therefore not stored as a `TEXT` node. This will further reduce the storage required by the `XMLDoc`, and will speedup `XPath` processing and node access processing.

\$XML_STR2DOC arguments

`$XML_STR2DOC` takes three arguments and returns a zero value if the deserialization is successful or a non-zero value if unsuccessful and the `ERRRET` option is used and the particular error is tolerated.

The `XMLDoc` must be `EMPTY` prior to invoking `$XML_STR2DOC`.

For example, the following fragment creates the `XMLDoc` representation of the indicated XML document:

```
%D = $XML_DOC
%R = $XML_STR2DOC(%D, -
    '<zen>What does the Buddha dog say?</zen>')
```

The following code calls a subroutine which uses the `ERRRET` option:

```
%D FLOAT
%S LONGSTRING
%D = $XML_DOC
... setup the (serialized) document in %S
CALL INTO_XML(%D, %S)
... do interesting things with the XMLDoc
... setup another document in %S
CALL INTO_XML(%D, %S)
...
SUBROUTINE INTO_XML(%D FLOAT, %S LONGSTRING)
%T FLOAT
%T = $XML_INITDOC(%D)
IF $XML_STR2DOC(%D, %S, 'ERRRET') THEN
... error handling code ...
END IF
END SUBROUTINE
```

As stated for the *option* argument above, whitespace normalization applies to the characters in the input serialized string, not the values after entity substitution. Therefore the values of “FOO1” and “FOO2” created by the following two \$XML_STR2DOC invocations are different:

```
* Get EBCDIC tab character:
%T = $X2C('Ø5')
* Element value is the null string:
%X = $XML_STR2DOC(%D, '<FOO1>' WITH %T WITH %T -
  WITH '</FOO1>')
%X = $XML_INITDOC(%D)
* Element value is two tab characters (note,
* character references are ISO-10646):
%X = $XML_STR2DOC(%D, '<FOO2>&#x09;&#x09;' -
  WITH '</FOO2>')
```

If you wish to deserialize a document (which has been POSTed or PUT) using *Janus Web Server*, use \$WEB_XML_RECV, as documented in the *Janus Web Server Reference Manual*. For other transport APIs, such as *Janus Sockets* or *Model 204 MQ Series*, \$XML_STR2DOC can be used to deserialize a document which has been received with the transport API.

\$XML_STR2DOC will accept any of the following types of input, corresponding to the **encoding** specified in the “<?xml version=....?>” declaration, and to the character codes contained in the input string:

ISO-8859-n In this case, the only values allowed for **encoding** are ISO-8859-n, where *n* is from 1 to 9. This **encoding** is required if the input contains any ASCII characters greater than X'7F', and is otherwise ignored.

UTF-8 In this case, the only value allowed for **encoding** is UTF-8. This encoding is optional; if present, it causes the value of \$XML_INFO(doc, 'ENCODING') to return UTF-8.

UTF-16 In this case, the first two bytes of the input must contain the Byte Order Marker (either X'FEFF' for Big Endian or X'FFFE' for Little Endian), and **encoding** must be UTF-16. This encoding is required, and it causes the value of `$XML_INFO(doc, 'ENCODING')` to return UTF-8. (The value returned by `$XML_INFO` in this case may change after version 6.4 of *Janus SOAP*; see [“Restrictions” on page 97.](#))

For example, the following fragment can be used to deserialize an XML document which has been sent to a *Janus Sockets* client as an HTTP response:

```
%S LONGSTRING
%D = $XML_DOC
%CL = -1
REPEAT
  %R = $SOCK_RECVPRS(%X, %T)
  IF %R LE 0 THEN
    LOOP END
  END IF
  IF %T EQ '' THEN
    LOOP END
  ELSEIF $SUBSTR(%T, 1, 16) EQ -
    'Content-Length: ' THEN
    %CL = $SUBSTR(%T, 16)
  END IF
END REPEAT
%RL = $SOCK_RECV(%X, %S, %CL, , 'BINARY')
%S = $XML_STR2DOC(%D, %S)
```

Notes:

- The BINARY option of `$SOCK_RECV` should always be used, so that `$XML_STR2DOC` can recognize the character encoding inherent in the serialized XML document.

- Argument one (*doc*) not an EMPTY XMLDoc ID
 - Argument three (*option*) invalid
 - Insufficient storage (this is tolerated if the ERRRET option is specified)
 - A wide class of errors caused by invalid syntax in the text string representation of the XML document (these are tolerated if the ERRRET option is specified)

\$XML_STR2DOC request cancel errors

The \$function which serializes an XMLDoc as a UTF-8 string is [“\\$XML_DOC2STR: Serialize XMLDoc as UTF-8 string” on page 44.](#)

4.34 **\$XML_TYPE: Obtain the type of a node**

\$XML_TYPE gets the type of node of the head of the argument XPath result.

<code>%type = \$XML_TYPE(id, XPath_expr, contx_n)</code>	
<code>id</code>	Identifier of XMLDoc or non-FREE nodelist. For relative <i>XPath_expr</i> : <ul style="list-style-type: none">• If <i>id</i> is an XMLDoc ID, the DOCUMENT node is the context.• If <i>id</i> is a nodelist ID, it must be non-EMPTY, and the first item is the context.
<code>XPath_expr</code>	The XPath expression which results in a nodelist, the head of which is the node to process. Optional: default is '.', that is, the context node.
<code>contx_n</code>	Position of context node in context nodelist or XMLDoc. Must be: <ul style="list-style-type: none">• 1 if <i>id</i> is an XMLDoc ID;• >= 1 and <= number of nodes in nodelist if <i>XPath_expr</i> is relative and <i>id</i> is a nodelist ID. Optional: default is 1.

\$XML_TYPE arguments

\$XML_TYPE takes three arguments and returns the type of node ('ATTRIBUTE', 'DOCUMENT', 'ELEMENT', etc.) of the head of the argument XPath result. If the argument XPath result is empty, \$XML_TYPE returns the null string.

- | |
|---|
| <ul style="list-style-type: none">• Argument one (<i>id</i>) neither XMLDoc nor non-FREE nodelist ID or, if <i>XPath_expr</i> is relative, is the ID of an EMPTY nodelist• Argument two (<i>XPath_expr</i>) invalid• Argument three (<i>contx_n</i>) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist• <i>XPath_expr</i> relative, and nodelist <i>id</i> item number <i>contx_n</i> references a node which has been deleted from the XMLDoc• Insufficient free space in CCATEMP |
|---|

\$XML_TYPE request cancel errors

4.35 **\$XML_VAL: Get string-value of node**

`$XML_VAL` obtains a substring of the string-value of a node, including its TEXT descendants.

```
%str = $XML_VAL(id, XPath_expr, contx_n)
```

<code>id</code>	Identifier of XMLDoc or non-FREE nodelist. For relative <i>XPath_expr</i> : <ul style="list-style-type: none">• If <i>id</i> is an XMLDoc ID, the DOCUMENT node is the context.• If <i>id</i> is a nodelist ID, it must be non-EMPTY, and the first item is the context.
<code>XPath_expr</code>	An XPath expression which results in a nodelist, the head of which is the node to process. Optional: default is '.', that is, the context node.
<code>contx_n</code>	Position of context node in context nodelist or XMLDoc. Must be: <ul style="list-style-type: none">• 1 if <i>id</i> is an XMLDoc ID;• >= 1 and <= number of nodes in nodelist if <i>XPath_expr</i> is relative and <i>id</i> is a nodelist ID. Optional: default is 1.

\$XML_VAL arguments

`$XML_VAL` takes three arguments and returns the string-value of the head of the argument XPath result.

Note that the string-value of a node depends on its type:

ATTRIBUTE

the normalized attribute value (see [“Normalized attribute value” on page 18](#))

COMMENT or TEXT

the content of the node

ELEMENT or DOCUMENT

the concatenation of the value of all its TEXT descendants, in document order

NSATTRIBUTE

the namespace URI

PI the portion of the processing instruction following its target and any whitespace immediately following the target.

If the argument XPath result is empty, \$XML_VAL returns the null string.

- Argument one (*id*) neither XMLDoc nor non-FREE nodelist ID or, if *XPath_expr* is relative, is the ID of an EMPTY nodelist
- Argument two (*XPath_expr*) invalid
- Argument three (*ctx_n*) not 1, if context is an XMLDoc, or specified as greater than number of nodes in context nodelist
- *XPath_expr* relative, and nodelist *id* item number *ctx_n* references a node which has been deleted from the XMLDoc
- Insufficient free space in CCATEMP

\$XML_VAL request cancel errors

4.36 **\$WEB_XML_RECV: Receive XML document using Janus Web**

The *Janus Web Server* \$function \$WEB_XML_RECV is used to receive an HTTP PUT or POST request which contains the text string representation of an XML document and convert that to an XMLDoc tree. (This latter process is called **deserialization**, because the text representation of a document is called the **serial** form).

For more information pertaining to the deserialization \$functions, which include \$WEB_XML_RECV, see [“Restrictions” on page 97](#).

```
%rc = $WEB_XML_RECV(doc, fieldname, -  
occurrence, p_flag)
```

doc Identifier of target XMLDoc.

fieldname The name of the form field associated with the “<input type=file>” tag in the HTML form, i.e. the value of the “name” parameter in that tag. This argument cannot be present if the upload request was the result of an HTTP PUT rather than a form-based upload. On a form-based upload if neither *fieldname* nor *occurrence* is present, the first (or only) file in the form will be uploaded.

occurrence The number of the form field associated with the “<input type=file>” tag in the HTML form. This argument must be 1 or not present if the upload request was the result of an HTTP PUT rather than a form-based upload. On a form-based upload if neither *fieldname* nor *occurrence* is present, the first (or only) file in the form will be uploaded. If *occurrence* is present but *fieldname* is not, the *occurrence* number file is uploaded regardless of its name. If both *occurrence* and *fieldname* are present, the *occurrence* number file with name *fieldname* is uploaded.

p_flag Can be any valid combination of the following:

- **ERRRET**, to indicate that errors during deserialization are tolerated, the request continues, and the XMLDoc remains EMPTY. If **ERRRET** is not present, any error will cancel the request. Note that some errors still cancel the request; errors tolerated when **ERRRET** is specified are explicitly noted in the table of cancellation errors below.
- **DTD_IGNORE**, to indicate that a “<!DOCTYPE ...>” clause may be present in the document, and that it should be ignored. As of version 6.3 of *Janus SOAP*, the DTD is not processed, so the default behavior, that is, if **DTD_IGNORE** is not present, is to treat “<!DOCTYPE ...>” as a syntax error.
- **WSP_NORM**, to indicate that whitespace in element content not “protected” by the `xml:space="preserve"` attribute, is

normalized using the XPath `normalize()` function (leading and trailing whitespace removed, and intermediate strings of whitespace replaced by a single blank character)

- `WSP_PRESERVE`, to indicate that all whitespace characters in element content are preserved.

Notes:

- No option may be specified twice.
- `WSP_PRESERVE` and `WSP_NORM` are mutually exclusive; if neither is specified, `WSP_NORM` is in effect.
- See “[\\$XML_STR2DOC: Deserialize string into XMLDoc](#)” on page 83 for more information about the `WSP_NORM` and `WSP_PRESERVE` options.

\$WEB_XML_RECV arguments

`$WEB_XML_RECEIVE` takes four arguments and returns a zero value if the deserialization is successful or a non-zero value if unsuccessful and the `ERRRET` option is used and the particular error is tolerated.

The `XMLDoc` must be `EMPTY` prior to invoking `$WEB_XML_RECV`.

For example, the following fragment creates the `XMLDoc` representation of the XML document received by a `PUT` or `POST` request:

```
%D = $XML_DOC
%R = $WEB_XML_RECV(%D)
```

The following subroutine uses the `ERRRET` option:

```
SUBROUTINE INTO_XML(%D FLOAT)
%T = $XML_INITDOC(%D)
IF $WEB_XML_RECV(%D, , , 'ERRRET') THEN
  ... error handling code ...
END IF
END SUBROUTINE
```

`$WEB_XML_RECV` will accept any of the following types of input, corresponding to the **encoding** specified in the “`<?xml version=...?>`” declaration, and to the character codes contained in the input string:

ISO-8859-n In this case, the only values allowed for **encoding** are ISO-8859-n, where **n** is from 1 to 9. This **encoding** is required if the input contains any ASCII characters greater than X'7F', and is otherwise ignored.

UTF-8 In this case, the only value allowed for **encoding** is UTF-8. This encoding is optional; if present, it causes the value of \$XML_INFO(doc, 'ENCODING') to return UTF-8.

UTF-16 In this case, the first two bytes of the input must contain the Byte Order Marker (either X'FEFF' for Big Endian or X'FFFE' for Little Endian), and **encoding** must be UTF-16. This encoding is required, and it causes the value of \$XML_INFO(doc, 'ENCODING') to return UTF-8. (The value returned by \$XML_INFO in this case may change after version 6.3; see “Restrictions” on page 97.)

To deserialize a document that has been received using other transport APIs, such as *Janus Sockets* or *Model 204 MQ Series*, use \$XML_STR2DOC (“\$XML_STR2DOC” on page 83).

Code	Meaning
0	Document successfully deserialized.
-1	Form field not found (if ERRRET specified)
> 0	Error deserializing document (if ERRRET specified)

\$WEB_XML_RECEIVE return codes

Note: These return codes are different from the return codes for similar conditions with \$WEB_LIST_RECEIVE. In particular, that function uses 1 for success and various other values for failure, and it tolerates invocation on a non-web thread, whereas \$WEB_XML_RECV does not.

- Argument one (*doc*) not an EMPTY XMLDoc ID
- Argument two (*fieldname*) or argument three (*occurrence*) specifies non-existent form field (these are tolerated if the ERRRET option is specified)
- Argument four (*p_flag*) invalid
- Insufficient storage (this is tolerated if the ERRRET option is specified)
- A wide class of errors caused by invalid syntax in the text string representation of the XML document (these are tolerated if the ERRRET option is specified)
- Not invoked on a web thread

\$WEB_XML_RECV request cancel errors

To serialize an XMLDoc as the HTTP response, use “[\\$WEB_XML_SEND: Send XML document using Janus Web](#)”.

4.37 **\$WEB_XML_SEND: Send XML document using Janus Web**

`$WEB_XML_SEND` is used to convert an XMLDoc to the text string representation of the XML document. (This process is called **serialization**, because the text representation of a document is called the **serial** form.)

For more information pertaining to the serialization \$functions, which include `$WEB_XML_SEND`, see “[Restrictions](#)” on page 97.

```
%junk = $WEB_XML_SEND(doc, option)
```

doc Identifier of XMLDoc to be serialized and sent; this must be WELL-FORMED (i.e., it must contain an ELEMENT node).

option Can be one of the following:

- `ALLOWXMLDECL | NOXMLDECL`
`ALLOWXMLDECL`, which is the default, indicates that the “`<?xml version=...?>`” declaration is produced if it had been set (see “[\\$XML_SET: Change XMLDoc option setting](#)” on page 80). `NOXMLDECL` indicates that the “`<?xml version=...?>`” declaration is not produced.
- `NOEMPTYELT`
This indicates that an empty element is serialized with start tag followed by an end tag, for example:

```
<middle_name>  
</middle_name>
```

If this option is not specified, the default is to serialize an empty element with an empty element tag; using the same example as above, this would be:

```
<middle_name/>
```

\$WEB_XML_SEND arguments

\$WEB_XML_SEND takes one argument and returns zero. Note that although other “web send” \$functions (such as \$WEB_PROCSEND) tolerate invocation on a non-web thread, \$WEB_XML_SEND does not.

For example, the following fragment serializes and sends an XML document:

```
%R = $WEB_XML_SEND(%D)
```

- Argument one (*id*) not WELL-FORMED XMLDoc ID
- Insufficient storage
- Not invoked on a web thread
- Argument two (*option*) not a valid option

\$WEB_XML_SEND request cancel errors

To deserialize an HTTP request containing an XML document, use “[\\$WEB_XML_RECV: Receive XML document using Janus Web](#)” on page 90.

CHAPTER 5 *Restrictions*

The current version of *Janus SOAP* has some restrictions, which may be removed in subsequent releases. The purpose of this chapter is to impart an understanding of those areas that we consider to be in a state of flux and why.

5.1 NAMESPACE default and limitations

As shown in “[\\$XML_SET: Change XMLDoc option setting](#)” on page 80, there are three possible values for the XMLDoc NAMESPACE setting. Only two of these are supported in version 6.4, however - since full namespace processing is not implemented, NAMESPACE ON is not supported.

When NAMESPACE ON is supported, we plan to let that be the default setting of an XMLDoc; however, until then, we felt it most convenient to let the default be NAMESPACE NONE. This will allow you to process documents which do not involve namespaces without any special compatibility considerations, that is, as long as your documents do not contain namespaces, you can process them exactly the same with both NAMESPACE ON and with NAMESPACE NONE. Therefore, all applications that work with the default NAMESPACE setting in version 6.4 will operate the same when the default is changed to NAMESPACE ON.

The only small compatibility issue which this may raise is that when the default is changed to NAMESPACE ON, if you want to continue to have *Janus SOAP* check that documents do not contain namespaces, you must issue `$XML_SET(id, 'NAMESPACE', 'NONE')` for all such documents.

5.2 XML syntax support

All productions in the *W3C XML Recommendation* are supported in version 6.4 of *Janus SOAP*, except the following:

ISO-10646 XMLDocs are maintained in EBCDIC; this may cause rejection of a document because it contains an ISO-10646 character which cannot be represented in EBCDIC. However, as stated in “[\[De\]Serialization \\$function support](#)” on page 100, UTF-8, UTF-16, and ISO-8859-x encodings are accepted. In a future version, we might maintain XMLDocs using an ISO-10646 encoding, such as UTF-16;

Also, as stated in “[Char and Reference](#)” on page 12, even the control characters (#x0-#x8, #xB, #xC, #xE-#x1F) are allowed, if they have EBCDIC correspondences.

- XML names** The CombiningChar and Extender characters are not allowed in XML names.
- Doctype** Document Type Declarations (“<!DOCTYPE ... >”) are only “tolerated”. Much of the functionality of document type declarations may be better provided using XML Schema, which is planned in a future version of *Janus SOAP*.

In addition, there are the following restrictions on an XML document:

- The **xml:space="preserve"|"default"** attribute is not supported (that is, it may not be used in an XML document).
- The maximum length of a name in an XML document (for example, an element name) is 650 characters.

5.3 XPath syntax support

The XPath “LocPath” production is supported, except for the axes, nodetests, predicates, and functions noted in the following sections; some of these initial restrictions are tied to the namespace limitations noted in “[NAMESPACE default and limitations](#)” on page 97.

One way to summarize the XPath productions which are not supported in version 6.4 of *Janus SOAP* is to list the XPath operators which are not supported, as shown in the following table:

Unsupported operators	Meaning
+ - * div mod	arithmetic
and or	Boolean
()	precedence
	union

In addition:

- The relational operators (= != < <= > >=) are supported in comparison predicates (“[Comparison test](#)” on page 100) only.
- XPath variables (“\$<var_name>”) are not supported.
- The numeric constants “NaN” and “+/- infinity”. are not supported.

- There is a limitation on the size of an XPath expression; a rough guess is, for example, 26 steps if each has an NCName nodetest.

5.3.1 XPath axis support

The following table lists the XPath axes supported and those not supported in version 6.4 of *Janus SOAP*:

Supported	Unsupported
self	ancestor
child	ancestor-or-self
parent	descendant
attribute	descendant-or-self
	following
	following-sibling
	namespace
	preceding
	preceding-sibling

5.3.2 XPath node test support

All XPath node tests except *NCName:** are supported in version 6.4 of *Janus SOAP*:

5.3.3 XPath predicate support

The only XPath predicates supported in version 6.4 of *Janus SOAP* are a simple position test, use of a location path expression for an existence test, and use of a location path expression with a comparison to a string literal.

Within a single step, only one predicate may occur, and location paths within predicates cannot themselves contain predicates.

Prior to version 6.4 of *Janus SOAP*, there was no predicate support.

5.3.3.1 [<n>] predicate

The “simple position test - [<n>]” is supported, for example:

```
/book/chapter[2]/section[9]/paragraph[3]
```

5.3.3.2 Location path within predicate

You can use a location path in a predicate of an XPath expression. This can take either of two forms, an “existence” test, and a “value comparison” test; simple examples of these are shown in the next two sections. Note that the location path in a predicate can be any supported location path, including multi-step and absolute expressions.

5.3.3.3 Existence test

You can use a location path as a predicate of an XPath expression; the predicate evaluates as true if the resulting nodeset is non-empty. The **usual** (but not only) purpose of this predicate is to select a node if it has at least one attribute or element of a given name; for instance:

```
/active/cust[invoice]/contact[fax]
```

The above example selects all contact children of cust elements, if the cust element has an invoice child element and the contact has a fax child element.

5.3.3.4 Comparison test

You can use a location path followed by a comparison operator and literal as a predicate of an XPath expression; the predicate evaluates as true if the comparison is true of at least one node in the resulting nodeset. The **usual** (but not only) purpose of this predicate is to select a node if it has at least one attribute or element whose value bears the requested relationship to the literal; for instance:

```
/inv/cust[@dt<"20030101"]/contact[state="MA"]
```

The above example selects all contact children of cust elements billed before 2003, if the contact has a state child element with the value “MA”.

5.3.4 XPath function support

Right now, none.

5.4 [De]Serialization \$function support

All the \$functions for serializing and deserializing a document are supported in version 6.4 of *Janus SOAP*. However, the following should be noted:

- Internal representation is in EBCDIC, so the deserialization \$functions may reject a document because it contains an ISO-10646 character which cannot be represented

in EBCDIC. See [“Char and Reference” on page 12](#) for more information about characters in an XML document.

- The different actual encodings (not simply encoding values in the XML declaration) accepted are UTF-8 and UTF-16, and ISO-8859-x, although all of the ISO-8859-x variants are treated the same.
- When the document is serialized, the only encoding which *Janus SOAP* can “control” is UTF-8; therefore, except for the valid encoding declarations described in [“\\$XML_STR2DOC: Deserialize string into XMLDoc” on page 83](#), the only encoding value permitted is UTF-8.

CHAPTER 6 *Messages*

Please refer to ***Sirius Messages Manual*** for messages related to *Janus SOAP*.

APPENDIX A *XPath*

This chapter has information to help you use XPath arguments to various *Janus SOAP* \$functions. Most of the information is taken from the XPath standard, which is the authoritative reference:

<http://www.w3.org/TR/xpath>

A.1 XPath operation

The purpose of XPath is to select a subset of nodes from a document; this is done using an **expression** as described by the `PathExpr` production (these syntax productions for XPath are shown in “XPath syntax” on page 108). The simple form (that is, without parentheses) of a `PathExpr` expression is called a **Location Path** (the `LocPath` production ([1]) in the syntax).

A Location Path consists of a series of **Steps** (`Step` ([4]) production). Each Step operates by taking an input set of nodes from the preceding step, and creating an output set of nodes. The output of the last step is the set of nodes selected by the XPath expression.

An example XPath expression is:

```
pitm[2]/partnum
```

This expression contains two Steps (the slash symbol (/) is used to separate the Steps in a Location Path.).

Often a Step will start with an element name, which selects all the child elements with that name. In the above example, `partnum` children of `pitm` elements are selected. These **child** relationships are one kind of relationship between the input to a Step and the first part of the algorithm; the kinds of relationships, or **Axes**, are shown in the `AxisName` ([6]) production.

The element names in the above example are a form of **Node Test**, described by the `NodeTest` ([7]) production; the Node Test is used to restrict the set of nodes.

Square brackets (“[...]”) in a Step surround another form of restriction, which is called a **Predicate**, given by the production ([8]) with the same name. A Predicate is a much more open-ended type of restriction, allowing various functions and operations, including Booleans.

The operation of a Step is as follows:

1. A Step consists of an Axis, Node Test, and zero or more Predicates.
2. The input to a Step is a set of context nodes.
3. The Axis produces sets of nodes, one set for each context node.
4. Each of these sets is filtered by the Node Test.
5. Each of the resulting sets is filtered by the first Predicate.
6. Each of the sets which are output by a Predicate is filtered by the following Predicate, if any.
7. The final filtered sets are combined (using set union), and the result is the set of nodes which becomes input to the following Step.
8. The result of the final Step is the result of the Location Path.

A.1.1 Axes

The various forms of the `AxisName` ([6]) production generate nodes based on a context node using the simple tree relationships described by the name. For example, `attribute::` (abbreviated `@`) generates the set of all attributes of a context node. The names `following` and `preceding` refer to those nodes which follow, or, respectively, precede, the context node in document order, that is, the order when the document is serialized.

A.1.2 Node tests

The various forms of the `NodeTest` ([7]) production filter nodes as follows:

NodeType '(' ')'

This selects any node that has the respective node type, for example `comment()` selects all Comment nodes in a node set.

'processing-instruction' '(' Lit ')'

This selects any Processing Instruction node if the target name is equal to the value of `Lit`.

The other forms test the **name** of a node, after restricting the type of node depending on the Axis:

- Name tests in the `attribute::` Axis restrict to Attribute nodes.
- Name tests in the `namespace::` Axis restrict to Namespace nodes.

- Name tests in any other Axis restrict to Element nodes.

The name tests then filter the resulting nodes as follows:

***** This selects a node of the selected type with any name.

NCName ':' ***** This selects a node of the selected type which has an associated namespace equal to the URI associated with **NCName**.

QName This selects a node of the selected type which has the same name as **QName**.

Namespace URI associations and QName equality are discussed in [“Names and namespaces - objectives” on page 13](#) and the sections following it.

A.1.3 Predicates

Each node set that is the result of Node Test filtering is input to the series of Predicates in the Step; each Predicate's result sets are passed to the following one, with union of the results of the last Predicate (or the Node Test, if there are no Predicates) forming the result of the Step.

There is a variety of Predicates, and except for a numeric Predicate, a Predicate selects a node if the value of the Predicate, converted to a Boolean, is `true`.

Common forms of predicates are:

- A Location Path expression.
For example, an element name such as `expired-date` selects a node if the result of that Location Path, using the node as its context node, is non-empty.
- A Location Path expression and comparison operator and literal.
For example, `Price > "200"` selects a node if any node in the result of that Location Path holds the specified relationship to the literal.
- A numeric.
If a Predicate p is numeric, it is equivalent to the expression `position()=p`.

The `position()` function is one of a number of functions that can be used in a Predicate, usually together with a comparison, as in `EqExpr` ([23], e.g., “=”) or in `RelExpr` ([24], e.g., “>”). See [“XPath functions” on page 110](#) for more information.

A.2 XPath syntax

This section contains a version of the XPath syntax. See [“XML syntax” on page 9](#) for an explanation of the syntax conventions. The syntax below has been changed from that in the XPath Recommendation in these ways:

- Names of some non-terminals have been changed (for example, “Lit” rather than “Literal”).
- Some productions have been collapsed; this introduces superficial ambiguity that is dealt with as needed, for example, showing the precedence of operators.
- A few of the productions have been moved, to illustrate that the PathExpr is the syntax goal. It is the form of expression that selects a set of nodes, which is the purpose of XPath. In other places in the manual, we refer to an “XPath expression;” specifically, we mean a PathExpr.

For a “cross-reference” to the productions as contained in the XPath Recommendation, see [“XPath syntax cross-reference” on page 111](#).

See also [“XPath syntax used in version 6.4 Janus SOAP” on page 113](#) for the syntax of that part of XPath used by *Janus SOAP* in version 6.3.

```
[A19] PathExpr ::= LocPath
           | PrimaryExpr Predicate+
           | PrimaryExpr Predicate* '/' RelativeLocPath
           | PrimaryExpr Predicate* '//' RelativeLocPath
```

```
[B15] PrimaryExpr ::= Variable | Lit
                | Number | FunctionCall
                | '(' UnaryExpr ')' | '(' Expr ')'
```

```
[C27] UnaryExpr ::= PathExpr ('|' PathExpr)*
                | '-' UnaryExpr | PrimaryExpr
```

```
[1] LocPath ::= RelativeLocPath
            | AbsoluteLocPath
```

```
[2] AbsoluteLocPath ::= '/' RelativeLocPath?
                | '//' RelativeLocPath
```

```
[3] RelativeLocPath ::= Step ('/' Step)*
                | RelativeLocPath '/' Step
```

```
[4] Step ::= '.' /* self::node() */
        | '..' /* parent::node() */
        | (AxisName '::' | '@')? NodeTest Predicate*
```

```
[6] AxisName ::= 'ancestor'
                | 'ancestor-or-self' | 'attribute'
                | 'child' | 'descendant'
```

```

    | 'descendant-or-self' | 'following'
    | 'following-sibling' | 'namespace'
    | 'parent'             | 'preceding'
    | 'preceding-sibling' | 'self'

[7] NodeTest ::= '*' | NCName ':' '*' | QName
             | NodeType '(' ')'
             | 'processing-instruction' '(' Lit ')'

[8] Predicate ::= '[' Expr ']'

[16] FunctionCall ::= FunctionName
                '(' ( Expr ( ',' Expr )*)? ')'

[21] Expr ::= EqExpr ( ('and' | 'or') EqExpr )*
[23] EqExpr ::= RelExpr ( '=' | '!=' ) RelExpr *
[24] RelExpr ::= NumExpr
            ( ('<' | '>' | '<=' | '>=') NumExpr )*

[25] NumExpr ::= UnaryExpr
        ( ('+' | '-' | '*' | 'div' | 'mod') UnaryExpr )*

[29] Lit ::= '"' [¬"]* '"'
        | "'" [¬']* "'"
[30] Number ::= [0-9]+ ( '.' [0-9]* )?
[35] FunctionName ::= QName - NodeType
[36] Variable ::= '$' QName
[38] NodeType ::= 'comment' | 'text'
            | 'processing-instruction' | 'node'

```

For information about XPath functions, see [“XPath functions” on page 110](#).

Syntax notes:

- In [A19], [2], and [3], // is an abbreviation for


```

        /descendant-or-self::node()/
      
```
- When @ is used in a Step ([4]), it is an abbreviation for


```

        attribute::
      
```
- The syntax for Step ([4]) notes that it may begin directly with a NodeTest; in that case, `child::` is implied before the NodeTest.
- The syntax for QName and NCName are given in [“Names and namespaces - syntax” on page 14](#).

- The precedence of expressions from `Expr` ([21]), `EqExpr` ([23]), `RelExpr` ([24]), and `NumExpr` ([25]) is (lowest precedence first):
 1. `or` (Short-circuit evaluation)
 2. `and` (ditto)
 3. `=, !=` (On node-sets geared to one opnd singleton)
 4. `<=, <, >=, >` (ditto) (No string ordering)
 5. `+, -`
 6. `*, div, mod`The operators are all left associative. For example, `3 > 2 > 1` is the same as `(3 > 2) > 1`, which evaluates to false.
- The only forms of `PrimaryExpr` ([B15]) that can create node sets, and so be used in a `PathExpr` ([A19]), are the `id()` function and parenthesized `LocPath` ([1]) (or parenthesized unions of them, with `|` ([C27])).

See also the notes in “[XPath syntax used in version 6.4 Janus SOAP](#)” on page 113.

A.2.1 XPath functions

There are many functions defined for XPath. Here are some of those which return a numeric result:

last() returns the number of nodes in the set which the predicate is filtering

position() returns the position of the node in the set which the predicate is filtering

The position in the set is dependent on the Axis in effect for the Predicate. The following axes use the reverse of document order to arrange the nodes in a node set:

1. `ancestor`
2. `ancestor-or-self`
3. `preceding`
4. `preceding-sibling`

All other axes use document order to arrange the nodes in a node set.

Note: If a `PathExpr` is parenthesized and followed by Predicates, and if `position()` is used in those predicates, the axis in effect is the `child` axis.

count(node-set) returns the of nodes in the argument node-set

Notice that this function uses a `node-set` type argument. As in this example, a `LocPath` can be passed:

```
/book/chapter[count(section) >= 3]
```

This expression will select all chapters that have three or more sections.

Here are some XPath functions that return a string result:

string(object?)

This function, like several other XPath functions, allows different kinds of arguments: for example, it can be used to convert a number to a string. The `string()` function is implicitly used when a comparison is made between a node and a string value, for example:

```
/book/chapter[@title = 'Introduction']
```

In this case, each node in the node set that is the result of the `@title` PathExpr is converted using `string()`, then compared to the string literal `Introduction`.

The default argument of the `string()` function is the context node of the expression. The `string()` function, when given a node set argument, uses the string value of the first node, in document order, of that node set.

substring(string, number, number?)

Returns the substring of the first argument, starting at the position specified by the second argument, and for the number of characters specified by the third argument (or the remainder of the string, if the third argument is omitted).

As with most XPath expressions, conversions are freely done, so if the first argument is not a manifest string type, it is converted to one using the `string()` function.

There are numerous other XPath functions.

A.2.2 XPath syntax cross-reference

Here is a listing of all first productions contained in various numbered sections and unnumbered sub-sections from the XPath Recommendation. This may be helpful if you wish to cross-reference the productions shown in “[XPath syntax](#)” on page 108 with those in the XPath Recommendation.

- 2 Location Paths
 - [1] LocationPath
- 2.1 Location Steps
 - [4] Step
- 2.2 Axes
 - [6] AxisName
- 2.3 Node Tests
 - [7] NodeTest

- 2.4 Predicates
 - [8] Predicate
- 2.5 Abbreviated Syntax
 - [10] AbbreviatedAbsolutePath
- 3.1 Basics
 - [14] Expr
- 3.2 Function Calls
 - [16] FunctionCall
- 3.3 Node-sets
 - [18] UnionExpr
- 3.5 Numbers
 - [25] AdditiveExpr
- 3.7 Lexical Structure
 - [28] ExprToken
 - ...
 - [39] ExprWhitespace (last production)

A.3 Attributes are not children

One subtle point to observe is that attributes are not children of their parents! As stated in the XPath Recommendation:

2.2 Axes

the descendant axis contains the descendants of the context node; a descendant is a child or a child of a child and so on; thus the descendant axis never contains attribute or namespace nodes

A.4 Order of nodes in nodelist

The order of nodes in an XML document is the order in which they first appear in the serial form of the document. Thus the DOCUMENT node is first, an ELEMENT node occurs before its ATTRIBUTE and NAMESPACE nodes, which appear before the children of the ELEMENT, and so on.

The position() function in XPath filters a node based on its order in a node set, and this order is the same as document order except for the following axes, for which the order is the reverse of document order:

- ancestor
- ancestor-or-self
- preceding
- preceding-sibling

We have implemented the order of nodelists to be document order for simplicity and to be consistent with XSLT. However, an XPath expression that “ends” in a “reverse axis” sacrifices the following relation:

```
... $XMLxxx(... id, expr[position], n...) ...
```

is equivalent to

```
%C = $XML_NODES(%TARG, id, expr, n)
... $XMLxxx(... %TARG, , position...) ...
```

(if the number of nodes used by \$XMLxxx from *expr* is one).

A.5 XPath syntax used in version 6.4 Janus SOAP

This section contains a condensed excerpt of the XPath syntax, showing only those parts of XPath used by *Janus SOAP* in version 6.4. See [“XML syntax” on page 9](#) for an explanation of the syntax conventions. The full syntax is given in [“XPath syntax” on page 108](#).

```
[1] LocPath ::= RelativeLocPath
        | '/' RelativeLocPath?

[3] RelativeLocPath ::= Step ('/' Step)*

[4] Step ::= '.' /* self::node() */
        | '..' /* parent::node() */
        | (AxisName '::' | '@')? NodeTest Predicate*

[6] AxisName ::= 'attribute' | 'child'
        | 'parent' | 'self'

[7] NodeTest ::= '*' | QName | 'node()'
        | 'comment()' | 'text()'
        | 'processing-instruction(' Lit? ')

[8] Predicate ::= '[' PredExpr ']'

PredExpr ::= PositiveInteger
        | PathExpr /* “Existence test” */
        | Comparison

Comparison ::= PathExpr
        ( '=' | '!=' | '<' | '>' | '<=' | '>=' ) Lit

[29] Lit ::= '"' [-"]* '"'
        | "'" [-']* "'"
```

[30] `PositiveInteger ::= [1-9] [0-9]*`

Notes:

- When @ is used in a Step ([4]), it is an abbreviation for

`attribute::`

- The syntax for Step ([4]) notes that it may begin directly with a `NodeTest`; in that case, `child::` is implied before the `NodeTest`.
- The syntax for `QName` is given in “[Names and namespaces - syntax](#)” on page 14.
- A node is selected by a `PositiveInteger` predicate if the position of the node, in the set which the predicate is filtering, is equal to that `PositiveInteger`.
- A node is selected by an `Existence` test if the result of the `PathExpr`, using that node as the context node, is non-empty.
- A node is selected by a `Comparison` if any node in the result of the `PathExpr`, using that node as the context node, holds the specified relationship to the `Lit`.

The `Predicate` term of a Step [4] was introduced with version 6.4 of *Janus SOAP*.

APPENDIX B *References*

B.1 Internet standards

As discussed earlier in this manual, SOAP (Simple Object Access Protocol) is an Internet standard. This section lists some of the XML-related standards documents which are available. A useful introductory subset of this list, and some additional introductory resources, are shown in “[Reading list](#)” on page 4.

The World Wide Web Consortium (or “W3C”) is the body that creates the XML standards, along with other Internet standards, such as HTML, XHTML, and HTTP. Note that the term “Recommendation”, in W3C parlance, means that the standard has been approved by the W3C.

Each document is shown with its title, the status of the standard and the date on which that status was achieved, and the URL which can be used to obtain the document:

Extensible Markup Language (XML) 1.0 (Second Edition)

W3C Recommendation 6 October 2000; <http://www.w3.org/TR/REC-xml>

We refer to this as the **W3C XML Recommendation** throughout this manual.

XML Schema

W3C Recommendation, 2 May 2001

- XML Schema Part 0: Primer; <http://www.w3.org/TR/xmlschema-0/>
- XML Schema Part 1: Structures; <http://www.w3.org/TR/xmlschema-1/>
- XML Schema Part 2: Datatypes; <http://www.w3.org/TR/xmlschema-2/>

XML Path Language (XPath) Version 1.0

W3C Recommendation 16 November 1999; <http://www.w3.org/TR/xpath>

Namespaces in XML

W3C Recommendation on or before 14 January 1999;
<http://www.w3.org/TR/REC-xml-names>

XML Information Set

W3C Proposed Recommendation 10 August 2001;
<http://www.w3.org/TR/xml-infoset>

SOAP Version 1.2

W3C Working Drafts 17 December 2001

- SOAP Version 1.2 Part 1: Messaging Framework;
<http://www.w3.org/TR/soap12-part1/>
- SOAP Version 1.2 Part 2: Adjuncts; <http://www.w3.org/TR/soap12-part2/>

The above documents are among the rich set of documents available from the World Wide Web Consortium; to browse for their complete public set of publications and useful links, go to

<http://www.w3.org/>

Index

\$

- \$Function prototypes ... 31, 33-35, 37, 39, 42-44, 47-50, 53, 55, 59, 61, 64, 66-69, 71-73, 75, 78, 80, 83, 88-90, 94
- \$WEB_XML_RECV(doc, fieldname, occurrence, p_flag) -> %rc ... 90
- \$WEB_XML_SEND(doc, option) -> %junk ... 94
- \$XML_CHECK(id) -> %type ... 33
- \$XML_COP(target_parent, source_subtree, child_num) -> %junk ... 35
- \$XML_COUNT(id, XPath_expr, ctx_n) -> %count ... 34
- \$XML_COUNT_NL(nl) -> %count ... 35
- \$XML_DEL(id, XPath_expr) -> %junk ... 37
- \$XML_DOC -> %doc ... 39
- \$XML_DOC_GLOBAL(name, option) -> %doc ... 39
- \$XML_DOC_GLOBAL_DEL(name_pattern) -> %count ... 42
- \$XML_DOC_SESSION(name, option) -> %doc ... 39
- \$XML_DOC_SESSION_DEL(name_pattern) -> %count ... 42
- \$XML_DOCID(nl) -> %doc ... 43
- \$XML_DOC2STR(doc, option) -> %UTF8_str ... 44
- \$XML_EXIST(id, XPath_expr, ctx_n) -> %exist ... 47
- \$XML_FREENL(nl) -> %nl ... 48
- \$XML_INFO(doc, option) -> %info ... 49
- \$XML_INITDOC(doc) -> %doc ... 50
- \$XML_INSATT(id, name, value, XPath_expr, ctx_n) -> %id ... 50
- \$XML_INSCOM(id, value, XPath_expr, ctx_n, child_num) -> %id ... 53
- \$XML_INSELT(targnl, ctxid, elt_name, value, XPath_expr, ctx_n, URI, child_num) -> %id ... 55
- \$XML_INSPI(id, targ_name, value, XPath_expr, ctx_n, child_num) -> %id ... 59
- \$XML_INSTEXT(id, elt_name, value, XPath_expr, ctx_n, URI, child_num) -> %id ... 61
- \$XML_LEN(id, XPath_expr, ctx_n) -> %len ... 64
- \$XML_NAME(id, XPath_expr, ctx_n, option) -> %name ... 66
- \$XML_NL -> %nl ... 67
- \$XML_NODE_SING(targnl, ctxid, XPath_expr, ctx_n) -> %count ... 68
- \$XML_NODES(targnl, ctxid, XPath_expr, ctx_n) -> %count ... 69
- \$XML_NS(doc, prefix, URI) -> %old ... 71
- \$XML_NSINFO(doc, prefix) -> %curr ... 72
- \$XML_PLC(outnl, ctxid, XPath_expr) -> %count ... 73
- \$XML_PRINT(id, XPath_expr, ctx_n, options) -> %junk ... 75
- \$XML_RMV_NL(outnl, innl) -> %count ... 78
- \$XML_SER(id, XPath_expr, ctx_n, options) -> %UTF8_str ... 78
- \$XML_SET(doc, option, value) -> %old ... 80
- \$XML_STR2DOC(doc, string, option) -> %rc ... 83
- \$XML_TYPE(id, XPath_expr, ctx_n) -> %type ... 88
- \$XML_VAL(id, XPath_expr, ctx_n) -> %str ... 89
- \$XML_PRINT ... 46

A

- Absolute XPath expression ... 22, 25
- AND (intersection) operation on nodelist ... 74
- axis, XPath ... 99, 105-106

C

Complement (NOT) operation on nodelist ... 74
Context node ... 22

D

Deserialization ... 82
Deserialize ... 83, 90
Document order ... 27
Document Type Declaration (DTD) ... 10, 15, 98

E

EMPTY state of nodelist ... 24
EMPTY state of XMLDoc ... 23
Encoding declaration ... 81, 101

F

FREE state of nodelist ... 23

H

Head of argument XPath result ... 26

I

Intersection (AND) operation on nodelist ... 74
ISO-10646 ... 12, 97

J

Janus Sockets ... 93
 Deserialize document ... 93
Janus Sockets Reference Manual ... 45, 86
 Deserialize document ... 86
 Serialize XMLDoc ... 45

M

Model 204 MQ Series ... 45, 86, 93
 Deserialize document ... 93
 Serialize document ... 86
 Serialize XMLDoc ... 45
MQ Series ... 45, 86, 93
 Deserialize document ... 86, 93
 Serialize XMLDoc ... 45

N

node test, XPath ... 99, 105-106
Nodelist ID ... 22

Nodelist states and pseudostates ... 23
 EMPTY state ... 23
 FREE state ... 23
 Non-EMPTY state ... 23
 Non-FREE pseudostate ... 23
Non-EMPTY not WELL-FORMED ... 23
Non-EMPTY not WELL-FORMED state ... 23
Non-EMPTY state of nodelist ... 24
Non-FREE state of nodelist ... 24
NOT (complement) operation on nodelist ... 74

O

OR (union) operation on nodelist ... 74

P

predicate, XPath ... 99, 105, 107

R

Relative XPath expression ... 22

S

Serialize ... 44, 78, 94
Serialized ... 2
Standalone declaration ... 81
String-value ... 26, 89
Subtraction operation on nodelist ... 74

U

Unicode ... 12
Union (OR) operation on nodelist ... 74
unique identifier, element ... 7

V

Version declaration ... 81

W

Web Server \$functions ... 90, 94
WELL-FORMED state of XMLDoc ... 23
W3C XML Recommendation ... 5

X

xml encoding declaration ... 81
XML Schema ... 98
xml standalone declaration ... 81
xml version declaration ... 81
XMLDoc ... 20
XMLDoc ID ... 22

XMLDoc states ... 23
 EMPTY state ... 23
 WELL-FORMED state ... 23

“
“<?xml ...” declaration ... 21, 45, 76

