Lycia Development Suite

Lycia XML Interface
Version 6 – March 2012
Revision number 0.03 BETA

Querix

Querix Lycia

XML Interface

Part Number: 025-006-003-012

Elena Krivtsova / Joss Giffard

Technical Writer / Developer

Last Updated 01 March 2012

Lycia XML Interface

Part Number: 025-006-003-012

Published by:

Querix (UK) Limited. 50 The Avenue, Southampton, Hampshire,
SO17 1XQ, UK

Publication history:

December 2011: Beta edition

Last Updated:

March 2012

Documentation written by:

Elena Krivtsova, Joss Giffard

Notices:

The information contained within this document is subject to change without notice. If you find any problems in the documentation please submit your comments by email to documentation@querix.com.

Table of Contents

Introduction

XML Interface was introduced by Lycia II for the convenience of the data processing. Any configuration files used to be stored in the plain text format which made them difficult to use and often led to bugs in the programs using them due to a large amount of code needed to process a single value. With the introduction of an XML parser integrated into the 4GL code the configuration files can be stored in the XML format and thus utilized in a simpler and cleaner way.

The Lycia XML Interface allows XML documents to be created, loaded and manipulated directly from within 4GL programs. This creates a vast range of possibilities for customizing and localizing your programs.

XML Interface Data Types

To manipulate XML files using 4GL code the special set of data types should be used. These data types are to be used with the corresponding methods which can be called like any other 4GL methods or built-in functions. There are the following objects:

- XMLNODE - represents an XML node based on the W3C XML idea of a node.
- XMLNODELIST - a container type for collections of XMLNODEs.
- XMLDOCUMENT - represents a complete single XML document.
- XMLATTRIBUTELIST - represents the list of a node attributes
- XMLATTRIBUTE - represents the data about an attribute of a node.
- XMLSAXPARSER - implements a SAX parser for 4GL which allows you to use 4GL methods for triggering the SAX events.

XMLNODE

This data type is used to store and manipulate nodes. This data type should be utilized with the set of corresponding methods:

- Methods which read from the specified xml file and retrieve nodes, their children and parents, information about nodes.
- Methods which write into the xml file and add or remove children and parents to or from the specified XML file.
- Methods which read and write node attributes.

XMLNODELIST

This data type serves as a container for node values.

XMLDOCUMENT

This data type represents an XML document. It can store existing XML documents or documents created from scratch using one of the methods.

XMLATTRIBUTELIST

This data type is used to contain the list of the attributes of a node.

XMLATTRIBUTE

Contains the information about a particular node attribute. The attribute information consists of four properties - attribute name, attribute value and attribute namespace (which includes prefix and URI).

XMLSAXPARSER

This data type implements a SAX parser for 4GL. It allows you to parse an XML document according to the SAX standards, whereas other data types listed above utilise the DOM standards. This allows you to register 4GL callback functions for a number of SAX events. This data type is very loosely connected to the previously described data types and comprises an alternative XML handler.

Using Methods

All the above mentioned data types are to be used differently, than the standard 4GL data type. They are so called object data types. They can normally be used only together with the special methods, which will be discussed for each data type later in this manual. These methods are similar to built-in

4GL functions and can be either called using the CALL statement, or, if a method returns exactly one value, be used as operands in 4GL expressions.

The methods should be prefixed by the corresponding variable:

```
DEFINE var <XMLDATATYPE>

CALL var.<method>()[RETURNING var]
```

For example:

```
DEFINE var XMLDOCUMENT

CALL var.Load("C:/docs/my_xml.xml")
```

XMLDOCUMENT Methods

The XMLDOCUMENT data type stores an XML file. This data type can be used to get the values of other related data types. For example, the XMLDOCUMENT data type can be used to retrieve an XMLNODE value or an XMLNODELIST value.

Creating and Saving XML Documents

Once a XMLDOCUMENT variable is declared, an empty XML document is implicitly created for the variable. You can add nodes to this document and make other modifications and then save the document to the disk. If you want to work with an existing document, you can initialize the variable using the Load() method.

Load()

This method parses the specified XML file into the XMLDOCUMENT variable used with it. It accepts a two parameter: one of a character data type which should consist of the file name and path to it and a boolean parameter which specifies whether the new lines will be ignored. If the second parameter is set to TRUE, the new lines are ignored, otherwise they are treated as empty nodes. E.g.:

```
CALL xmldocument_var.Load("C:/xml_files/my.xml", TRUE)
```

Pay attention, that if the loaded file contains new line characters, other methods applied to the variable containing such a file may produce incorrect results, if the second parameter is set to FALSE. It is advisable that you use files without the new line symbols in this case.

ToString()

This method returns the referenced document in the form of a character string, it accepts no arguments. In this way you can assign the contents of XMLDOCUMENT variables to other 4GL variables.

```
LET str = xmldoc_var.ToString()
```

FromString()

This method assigns a character string passed to it as a parameter as a value to the referenced XMLDOCUMENT variable. It accepts two parameters. The first parameter is a character string containing the XML code and the second parameter is a Boolean value which defines whether the white spaces contained in the character string, if any, should be ignored during converting it into a XMLDOCUMENT value. If the last argument is set to TRUE, the whitespaces will be ignored. Set this option to TRUE, if the character string contains whitespaces and new line symbols between the node tags, otherwise other methods used to count and retrieve nodes will return incorrect results, treating new line symbols as nodes.

```
CALL xmldoc_var.ToString(str, true)
```

If the character string contains any other nodes before the root node, e.g. a comment node or a DTD node, these nodes will be omitted and will not be assigned to the variable.

Save()

To save the contents of an XMLDOCUMENT variable as an XML file the Save() method is used. It accepts a single parameter of a character data type which contains the name of the destination xml file and the optional path. If the file does not exist, it is created when this method is called.

Here is an example of an xml file created in a 4GL application and then saved to disk:

```
MAIN
  DEFINE document XMLDOCUMENT
  DEFINE node XMLNODE
  DEFINE comment_node XMLNODE
  DEFINE child_node XMLNODE
  DEFINE str STRING
 #first some nodes and child nodes are created
  LET node = document.CreateNode("root")
  LET comment_node = document.CreateComment("Sample comment")
  LET child_node = document.CreateElement("child")
  CALL child_node.SetNodeValue("Some random text here")
# then they are all gathered together into a root node
  CALL node.AppendChild(comment_node)
  CALL node.AppendChild(child_node)
# the root node is assigned to a string
  LET str = node.toString()
# Then this string is used to replace the empty content of the
# XMLDOCUMENT variable
  CALL document.fromString(str)
# and the XMLDOCUMENT variable is saved into an XML file
  CALL document.Save("out.xml")
END MAIN
```

Here is the text of the XML file "out.xml" the program above generates:

```
<?xml version="1.0"?>
```

```
<root>
  <!--Sample comment-->
  <child>Some random text</child>
</root>
```

Accessing XMLDOCUMENT Variables

The following methods manipulate the value of the associated XML document, retrieve its parts and initialize variables of the XMLNODE and XMLNODELIST data types.

GetDocumentElement()

This method accepts no parameters and returns the root element of the XML document associated with the variable it is used with. It returns the object of the XMLNODE data type which contains the contents of the document element. E.g.:

```
LET xmlnode_var =  xmldocument_variable.GetDocumentElement()
```

GetFirstDocumentNode()

This method accepts no parameters and returns the first node of the XML document associated with the variable. The returned value is of the XMLNODE data type which contains the first node regardless of the node type. E.g.:

```
LET xmlnode_var = xmldocument_var.GetFirstDocumentNode()
```

In the case of the xml file example below, the method will return the first node following the comment node. Any comment node preceding the root node will be ignored:

```
<!-- This is the comment node -->
<test>
<one>
  <blah/>
</one>
<one>
  <blah>
     This is the text included into blah node.
  </blah>
</one>
</test>
```

GetLastDocuementNode()

This method accepts no parameters and returns the last node of the XML document associated with the variable. The returned value is of the XMLNODE data type which contains the last node regardless of the node type. E.g.:

```
LET xmlnode_var = xmldocument_var.GetLastDocumentNode()
```

GetDocumentNodesCount()

This method returns the total number of nodes in the XML document associated with the used variable. The data type of the returned value is INTEGER. It counts all the nodes in the document regardless of the node type and the degree of nesting. E.g.:

```
LET int_var = xmldocument_var.GetDocumentNodesCount()
```

For the following XML file it should return 5:

```
<test>
<one>
  <blah/>
</one>
<one>
  <blah>
     This is the text included into blah node.
  </blah>
</one>
</test>
```

GetDocumentNodeItem()

This method returns the XMLNODE of the specified index, or NULL, if the node with such index does not exist in the document associated with the variable. It accepts a single parameter of the INTEGER data type which represents the node index. E.g.:

```
DISPLAY xmldocument_var.GetDocumentNodeItem(5)
```

GetElementsByTagName()

This method returns the contents of the node which name is specified as the method parameter. Thus it accepts a single parameter of a character data type. This method returns a value of the XMLNODELIST data type which contains all the nodes whose names are the same as the specified parameter. E.g.:

```
LET xlmnodelist_var = xmldocument_var.GetElementsByTagName("one")
```

GetElementsByTagNameNS()

This method returns the contents of the node which name is specified as the method parameter and with the given namespace. It accepts two parameters of a character data type: The name of the node and the namespace URI. This method returns a value of the XMLNODELIST data type which contains all the nodes whose names and namespaces are the same as the specified parameters. E.g.:

```
LET xlmnodelist_var = xmldoc.GetElementsByTagNameNS("one",
"http://www.w3.org/2001/XMLSchema")
```

GetElementByID()

This method returns a value of the XMLNODE data type that matched the ID given as the parameter. If there is no node with such ID, it returns NULL. It accepts a single parameter of a character data type which represents the element ID. E.g.:

```
LET xmlnode_var = xmldocument_var.GetElementByID("123id")
```

Note, that the ID node can have any name, not necessarily "id". For the node to be identified as an ID node it must be defined as type 'xml:id'. In the most simple example, this can be done with a DTD, for example:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE doc [

<!ELEMENT doc (test)>

<!ELEMENT test (#PCDATA)>

<!ATTLIST test id ID #IMPLIED>

]>

<doc>

<test id="123">potatoes</test>

</doc>
```

Manipulating XMLDOCUMENT Variables

The following methods are used to modify the content of the XMLDOCUMENT variable directly by adding or removing nodes.

ImportNode()

This method creates a new node under the ownership of the document as a copy of the node passed as the parameter.

It accepts two parameters: the first parameter of the XMLNODE data type should contain the node to be copied and the "recursive" parameter of the BOOLEAN data type which is optional. The recursive parameter specifies whether only the root node will be imported or the child nodes also. If it parameter is set to TRUE, the child nodes are also imported. The default value of this parameter is TRUE, so if it is omitted, the children are copied together with the parent node. E.g.:

```
LET xmlnode_var = xmldocument_var.ImportNode(new_node_var,0)
```

This method returns a value of the XMLNODE data type which represents the imported node. It does not specify where within the document the copy of the node should exist, so to actually add the imported node to the document you need to use other methods. For example you can append the copied node to an existing node within the document:

```
LET n = d.importNode(somenode)  -- creates a copy of "somenode"

LET de = d.GetDocumentElement() -- retrieves an existing node

CALL de.AppendChild(n)          -- adds the copy inside the existing

                                -- node as its last child
```

Clone()

This method returns a value of the XMLNODE data type which will contain the exact copy of the root node of the document, referenced by the variable. It accepts no parameters. E.g.:

```
CALL xmldocument_var.Clone()
```

PrependDocumentNode()

This method adds the node passed to it as a parameter to the beginning of the XML document referenced by the variable. It accepts a single parameter of the XMLNODE data type. E.g.:

```
CALL xmldocument_var.PrependDocumentNode(xmlnode_var)
```

InsertBeforeDocumentNode()

This method inserts the new node into the document referenced by the variable immediately before the node specified as a parameter. It accepts two parameters, both of which are of the XMLNODE data type. The first parameter is the XMLNODE variable containing the new node to be inserted. The second parameter is the XMLNODE variable containing a node which already exists within the document. The new node is added before the existing one. Here is the syntax:

```
xmldocument_var.InsertBeforeDocumentNode(new_node, existing_node)
```

RemoveDocumentNode()

This method removes the specified node from the referenced document. It accepts a single parameter of the XMLNODE data type which indicates the node to be deleted. E.g.:

```
CALL xmldocument_var.RemoveDocumentNode(existing_node)
```

Creating New Nodes

The following methods do not actually modify the XMLDOCUMENT variable with which they are used. They are mainly used to create different types of nodes which are not directly added to the document referenced by the variable used. You can add the nodes to the document or to other nodes using other appropriate methods.

CreateNode()

This method creates an empty node of the specified name. It accepts a single parameter of a character data type which represents the name of the newly created node. This method returns the newly created node in the form of an XMLNODE value. E.g.:

```
LET xmlnode_var = xmldocument_var.CreateNode("new_node")
```

CreateNodeNS()

This method creates an empty node of the specified name, with the specified namespace prefix and namespace URI. It accepts three parameters of a character data type which represent the name of the newly created node, the namespace prefix and the namespace URI accordingly. This method returns the newly created node in the form of an XMLNODE value. Here is the syntax:

```
xmldocument_var.CreateNodeNS("prefix","node_name","namespaceURI")
```

E.g.:

```
LET new_node = xmldocument_var.CreateNodeNS("health", "body",
"http://www.example.org/health")
```

The above code should result in:

```
<health:body xmlns:health="http://www.example.org/health"/>
```

CreateElement()

This method creates a new element type XMLNODE with the specified name. It accepts a single parameter of a character data type and uses it as the name for the new element. It returns the value of the newly created element which is of the XMLNODE data type. E.g.:

```
LET xmlnode_var = xmldocument_var.CreateElement("new_element")
```

CreateElementNS()

This method creates an empty element of the specified name, with the specified namespace prefix and namespace URI. It accepts three parameters of a character data type which represent the name of the newly created element, the namespace prefix and the namespace URI accordingly. This method returns the newly created element in the form of an XMLNODE value. Here is the syntax:

```
xmldocument_var.CreateElementNS("prefix", "node_name",
"namespaceURI")
```

CreateTextNode()

This method creates a text node with the specified content. It accepts a single parameter of a character data type which contains the text to be added to the text node. It returns the created node in the form of an XMLNODE value. E.g.:

```
LET xmlnode_var = xmldocument_var.CreateTextNode("Text to be added to
the node.")
```

CreateComment()

This method creates a comment node and returns the value of the XMLNODE data type containing the newly created node. It accepts a single parameter of a character data type containing the text of the comment. E.g.:

```
LET xmlnode_var = xmldocument_var.CreateComment("This is a comment.")
```

CreateCDATASection()

This method creates a new CDATA section containing the data passed as the parameter. It accepts a single parameter of a character data type which represents the data. and returns an XMLNODE value containing the newly created CDATA. E.g.:

```
LET xmlnode_var = xmldocument_var.CreateCDATAsection("this is the
CDATA text")
```

The resulting CDATA section will have the following contents:

```
<![CDATA[this is the CDATA text]]>
```

CreateEntityReference()

This method creates a reference and returns the value of the XMLNODE data type containing the newly reference node. It accepts a single parameter of a character data type containing the text of the reference. E.g.:

```
LET xmlnode_var = xmldocument_var.CreateEntityReference("amp")
```

In the example above "amp" will be turned into "&" which will be changed to "&" symbol automatically when the document is processed.

CreateProcessingInstruction()

This method creates a new processing instruction node. It accepts two parameters of a character data type, the first contains the processing target also called the name of the processing instruction, the second contains the processing data. It returns a value of the XMLNODE data type which represents the newly created processing instruction node. E.g.:

```
  LET p_i = doc.CreateProcessingInstruction("xml-stylesheet",
"type=\"text/xsl\" href=\"show_book.xsl\"" )
```

The following processing instruction will be created:

```
  <?xml-stylesheet type="text/xsl" href="show_book.xsl"?>
```

CreateDocumentFragment()

This method creates a new document fragment type node. It accepts no parameters and returns a value of the XMLNODE data type with the newly created node.

```
LET xmlnode_var = xmldocument_var.CreateDocumentFragment()
```

XMLNODE Methods

As it was already mentioned above, the XMLNODE data type can be used together with its specific methods in order to get the information about nodes and modify them. Variables of this data type cannot be actually used to create new nodes from scratch. To create nodes use the methods for creating new nodes together with an XMLDOCUEMNT variable. To initialize XMLNODE variables you will also need to use an XMLDOCUMENT variable with the corresponding methods. The methods described in this section only work with variables initialized previously.

Node Access Methods

The methods in this section are used to retrieve information about the referenced node such as the node type, value, etc..

GetNodeType()

This method returns the type of the referenced node. It returns a string value and accepts no parameters. The table below lists all the node types and the values the method returns depending on the type:

Node Type	Returned Value	XML example
Element	ELEMENT_NODE	<element>
Attribute	ATTRIBUTE_NODE	<xxx attribute="sample">
Text	TEXT_NODE	normal text
Character Data	CHARACTER_DATA	normal text
CDATA Section	CDATA_SECTION_NODE	<!CDATA[function match (a,b)]]>
Entity Reference	ELEMENT_REFERENCE_NODE	& #34
Entity	ENTITY_NODE	
Processing Instruction	PROCESSING_INSTRUCTION_NODE	<?xml version=1.0"?>
Comment	COMMENT_NODE	<!-- comment -->
Document	DOCUMENT_NODE	
Document Type	DOCUMENT_TYPE_NODE	
Document Fragment	DOCUMENT_FRAGMENT_NODE	

GetLocalName()

This method returns the local name of the referenced node in the form of a character string. It accepts no parameters.

```
DISPLAY xmlnode_var.GetLocalName()
```

For a note defined like shown below, the returned value will be "theNode":

```
<theNode>
   <blah/>
</theNode>
```

GetNodeName()

This method returns the name of the referenced node in the form of a character string. It accepts no parameters.

```
DISPLAY xmlnode_var.GetNodeName()
```

GetNamespaceURI()

This method returns the namespace URI if the referenced node in the form of a character string. It accepts no parameters.

GetNodeValue()

This method returns the value of the referenced node in the form of a character string. It accepts no parameters.

GetPrefix()

This method returns the namespace prefix of the referenced node in the form of a character string. It accepts no parameters.

IsAttached()

This method defines whether the referenced node is attached to the root element of the XMLDOCUMENT. If it is, the method returns TRUE, otherwise it returns FALSE.

```
DISPLAY xmlnode_var.IsAttached()
```

Node Manipulation Methods

These methods allow you to change the node characteristics.

SetNodeValue()

This method replaces the value of the referenced node with the value supplied as the method parameter. If the node did not contain any value, it just inserts the new value. It accepts a single parameter of a character data type.

```
DISPLAY xmlnode_var.SetNodeValue("this is the new value")
```

SetPrefix()

This method replaces the existing namespace prefix of the referenced node with the one passed as the method parameter. If the node did not have a prefix previously, the method sets it. It accepts a single parameter of a character data type which is the new prefix.

```
DISPLAY xmlnode_var.SetPrefix("newprefix")
```

Nested Nodes Access Methods

The methods in this section can be used to retrieve any nodes nested within the referenced node. They affect only the nodes of the first level of nesting.

HasChildNodes()

This method checks whether the node associated with the variable has any child nodes attached and returns TRUE, if it does. It returns FALSE, if the node has no children.

```
DISPLAY xmlnode_var.HasChildNodes()
```

GetChildCount()

This method returns an integer value which represents the number of children attached to the node associated with the variable. It calculates only the nodes attached to the referenced node directly and does not calculate the nested nodes the children may have. E.g.:

```
DISPLAY xmlnode_var.GetChildCount()
```

GetChildNodeItem()

This method returns the child node of the node associated with the index specified. If there is no child node with this index - it returns NULL. This method accepts a single parameter of the INTEGER data type which represents the child node index. The node indexes are assigned in the following way: the first child node has index value 1 and all the rest of the nodes have the indexes which corresponds to their position regarding the parent node. It returns the child node in the form of an XMLNODE value. E.g.:

```
LET child_node = xmlnode_var.GetChildNodeItem(4)
```

GetParentNode()

This method returns the parent node of the node associated with the XMLNODE variable used, or NULL if the node has no parent. It accepts no parameters. E.g.:

```
LET par_node = xmlnode_var.GetParentNode()
```

In case if the xmlnode_var variable contains the node called "node", this method will return another XMLNODE value containing the node called "parent, if the parent node is defined like this:

```
<parent>

   <node/>

</parent>
```

GetFirstChild()

This method returns the first child node of the node associated with the XMLNODE variable used, or NULL if the node has no children. It accepts no parameters. E.g.:

```
LET child_node = xmlnode_var.GetFirstChild()
```

The par_node variable will contain the value of the "first_child" node, if the xmlnode_var contains the value of the "node" node defined like this:

```
<node>

  <first_child/>
```

```
  <second_child/>

  <last_child/>

</node>
```

GetFirstChildElement()

This method returns the first child of the XMLNODE associated with the variable used that is of type ELEMENT or NULL if the XMLNODE has no children that are ELEMENT types. The returned value is of the XMLNODE data type. E.g.:

```
LET first_child_element = xmlnode_var.GetFirstChildElement()
```

GetLastChild()

This method returns the last child node of the node associated with the XMLNODE variable used, or NULL if the node has no children. It accepts no parameters. E.g.:

```
LET child_node = xmlnode_var.GetLastChild()
```

The par_node variable will contain the value of the "last_child" node, if the xmlnode_var contains the value of the "node" node defined like this:

```
<node>

  <first_child/>

  <second_child/>

  <last_child/>

</node>
```

GetLastChildElement()

This method returns the last child of the XMLNODE associated with the variable used that is of type ELEMENT or NULL, if the XMLNODE has no children that are ELEMENT types. The returned value is of the XMLNODE data type. E.g.:

```
LET last_child_element = xmlnode_var.GetFirstChildElement()
```

GetNextSibling()

This method returns the next sibling of the XMLNODE associated with the variable used. The sibling of the node is the next node of the same nesting level as the referenced node located immediately after the referenced node. If No sibling nodes exist, it will return NULL, otherwise it will return the value of the XMLNODE data type. E.g.:

```
LET sibling = xmlnode_var.GetNextSibling()
```

If the node has no siblings that are located after or below it, this method will return NULL, even if there are siblings above or before the referenced node.

GetNextSiblingElement()

This method returns the next sibling of the XMLNODE associated with the variable used that is of type ELEMENT or NULL, if the XMLNODE has no siblings that are ELEMENT types. The returned value is of the XMLNODE data type. E.g.:

```
LET sibling_element = xmlnode_var.GetNextSiblingElement()
```

If the node has no sibling elements that are located after or below it, this method will return NULL, even if there are sibling elements above or before the referenced node.

GetElementsByTagName()

This method returns all the elements of the referenced node that match the specified tag name. It accepts a single parameter which is the name of the tag. It returns a value of the XMLNODELIST data type containing all the matching elements.

```
LET xmlnodelist_var = xmlnode_var.GetElementsByTagName("mytag")
```

Nested Nodes Manipulation Methods

The methods below are used to add or remove nested nodes.

AppendChild()

This method adds the XMLNODE passed as the parameter as the last child of this XMLNODE associated with the variable. It accepts a single parameter of the XMLNODE data type and returns the added node as an XMLNODE.

```
LET ch_node = p_node.AppendChild(new_child)
```

AppendChildElement()

This method appends a new child element to the end of this XMLNODE. It accepts a single parameter of a character data type which specifies the name of the new element.

```
LET new_node = xmlnode_var.AppendChildElement("new_element")
```

AppendChildElementNS()

This method appends a new child element to the end of this XMLNODE. It accepts three parameters of a character data type. First parameter is the namespace prefix for the new element, the second one is its name and the third one the namespace URI.

```
LET child_elem = xmlnode_var.AppendChildElementNS("prefix",
"elementNode", "//test/URI")
```

PrependChild()

This method adds the XMLNODE passed as the parameter as the first child of this XMLNODE associated with the variable. It accepts a single parameter of the XMLNODE data type and returns the added node as an XMLNODE.

```
LET ch_node = p_node.PrependChild(new_child)
```

PrependChildElement()

This method prepends a new XMLNODE child with specified name as the first child of the XMLNODE associated with the variable. It accepts a single element of a character data type which specifies the name of the prepended child node. This method returns a value of the XMLNODE data type containing the newly prepended node. E.g.:

```
LET child_node = xmlnode_var.PrependChildElement("elementName")
```

If the xmlnode_var variable contains node "TheNode" with the only child called "otherchild", the example above will result in the following text:

```
<TheNode>

  <elementName/>

  <otherchild/>

</TheNode>
```

PrependChildElementNS()

This method prepends a new XMLNODE child with specified name as the first child of the XMLNODE associated with the variable. It accepts three parameters of a character data type. The first parameter is the namespace prefix, the second one is the name of the prepended node, and the third one is the namespace URI. This method returns a value of the XMLNODE data type containing the newly prepended node. E.g.:

```
LET child_node = xmlnode_var.PrependChildElementNS("prefix",
"elementNode", "//test/URI")
```

If the xmlnode_var variable contains node "TheNode" with the only child called "otherchild", the example above will result in the following text:

```
<TheNode>

  <elementNode xmlns:prefix="" xmlns:="//test/URI"/>

  <otherchild/>

</TheNode>
```

AddPreviousSibling()

This method inserts the specified XMLNODE as the previous sibling of the XMLNODE associated with the variable. It accepts a single parameter of the XMLNODE data type and returns an XMLNODE value containing the added node. E.g.:

```
LET child_node = xmlnode_var.AddPreviousSibbling(node_to_add)
```

If the xmlnode_var contains the value of the node called "ThisNode" and the node_to_add variable contains the node called "Sibling", the example above will result in the following text:

```
<ParentOfBoth>
```

```
  <Sibling/>

  <ThisNode/>

</ParentOfBoth>
```

AddNextSibling()

This method inserts the specified XMLNODE as the next sibling of the XMLNODE associated with the variable. It accepts a single parameter of the XMLNODE data type and returns an XMLNODE value containing the added node. E.g.:

```
LET child_node = xmlnode_var.AddPreviousSibbling(node_to_add)
```

If the xmlnode_var contains the value of the node called "ThisNode" and the node_to_add variable contains the node called "Sibling", the example above will result in the following text:

```
<ParentOfBoth>

  <ThisNode/>

  <Sibling/>

</ParentOfBoth>
```

InsertBeforeChild()

This method inserts a new child XMLNODE into the XMLNODE associated with the variable before its existing child node. It accepts two parameters of the XMLNODE data type. The first parameter contains the new node to be inserted. The second parameter contains the child node before which it should be inserted. The method returns an XMLNODE value containing the newly inserted node. E.g.:

```
LET child_node = xmlnode_var.InsertBeforeChild(new_node, ref_node)
```

If we assume that xmlnode_var is associated with the node called "ThisNode", the new_node variable is associated with the "newNode" node and the ref_node variable is associated with the "referenceNode" node, the example above will result in the following code:

```
<ThisNode>

  <randomChild/>

  <newNode/>

  <referenceNode/>

  <anotherChild/>

</ThisNode>
```

InsertAfterChild()

This method inserts a new child XMLNODE into the XMLNODE associated with the variable after its existing child node. It accepts two parameters of the XMLNODE data type. The first parameter

contains the new node to be inserted. The second parameter contains the child node after which it should be inserted. The method returns an XMLNODE value containing the newly inserted node. E.g.:

```
LET child_node = xmlnode_var.InsertBeforeChild(new_node, ref_node)
```

If we assume that xmlnode_var is associated with the node called "ThisNode", the new_node variable is associated with the "newNode" node and the ref_node variable is associated with the "referenceNode" node, the example above will result in the following code:

```
<ThisNode>
  <randomChild/>
  <referenceNode/>
  <newNode/>
  <anotherChild/>
</ThisNode>
```

RemoveChild()

This method removes the specified child node from the XMLNODE associated with the variable. It accepts a single parameter of the XMLNODE data type which specifies the node to be removed.

```
CALL xmlnode_var.RemoveChild(node_to_remove)
```

RemoveAllChildren()

This method removes all the children from within the references XMLNODE and leaves this node empty. It accepts no parameters.

```
CALL xmlnode_var.RemoveAllChildren()
```

ReplaceChild()

This method replaces a child within the referenced node with another child node. It accepts two parameters of the XMLNODE data type. The first parameter is the node which needs to be inserted and the second parameter is the existing node which needs to be replaced. It returns the new replaced node as an XMLNODE value or NULL, if the existing node to be replaced was not found.

```
LET new_node = xmlnode_var.ReplaceChild(new_n,old_n)
```

CreateChildElement()

This method creates a new element node as a child of the current one. The method needs one argument which is to be represented by a character string, specifying the name of the child node to be created. It returns a value of the XMLNODE data type which contains the new child element.

For example, if there is a parent node <one> :

```
<one>
...
```

```
</one>
```

A child node for the node given above can be created the following way:

```
LET childnodevar = parentnode.CreateChildElement("blah")
```

After the method is used, the parent node will get a new child:

```
<one>
...
    </blah>
</one>
```

CreateSiblingElement()

This method creates a sibling element for the current one and inserts it right after the referenced node. The method needs one attribute which is a string specifying the name of the new node. It returns a value of the XMLNODE data type containing the newly created sibling.

For example, if there is a following tree:

```
<one>
...
    </blah>
    </two>
</one>
```

It is possible to add a sibling for the <blah> node using the following syntax (note, that the new node is appended, not inserted):

```
CALL blah.CreateSiblingElement("blah_blah")
```

After the method is called, the structure of the <one> node will be changed:

```
<one>
...
    </blah>
    </blah_blah>
    </two>
</one>
```

Auxiliary Methods

These methods deal with some auxiliary tasks like copying the node content and retrieving various information about the node.

GetOwnerDocument()

This method returns an XMLDOCUMENT which contains the XMLNODE associated with the variable. E.g.:

```
LET nwe_xml_doc = xmlnode_var.GetOwnerDocument()
```

Clone()

This method returns a copy of the referenced node. It has a single parameter of the BOOLEAN data type. If the parameter is set to TRUE, the node is copied with all its children. If it is set to FALSE, the children are not copied.

```
LET new_node = xmlnode_var.Clone(0)
```

ToString()

This method returns the referenced node in the form of a character string. In this way you can assign the contents of XMLNODE variables to other 4GL variables.

```
LET str = xmlnode_var.ToString()
```

DefaultNamespace()

This method returns true if the specified namespace is the default namespace for the referenced XMLNODE. It accepts a single parameter of a character data type which is the namespace in question. It returns TRUE, if the specified namespace is the default one, otherwise it returns FALSE.

```
LET xmlnodelist_var = xmlnode_var.DefaultNamespace("namespace")
```

Attribute Access Methods

The methods in this section are used to retrieve the attributes of the referenced node and the attribute details. The methods which retrieve the attributes proper retrieve them in the form of an XMLATTRIBUTELIST value.

HasAttribute()

This method checks whether the referenced node has an attribute of the specified name. If it does, the method returns TRUE. The method accepts a single parameter of a character data type which is the name of the attribute the presence of which is checked.

```
DISPLAY xmlnode_var.HasAttribute("my_att")
```

HasAttributeNS()

This method checks whether the referenced node has an attribute of the specified name and namespace prefix. The method accepts two parameters of a character data type, the first one is the name of the attribute the presence of which is checked and the second one it the namespace URI of

the attribute. If the referenced node contains the attribute of the specified name and namespace, the method returns TRUE, otherwise it returns FALSE.

```
DISPLAY xmlnode_var.HasAttribute("my_att",
"http://www.w3.org/2001/XMLSchema")
```

HasAttributes()

This method checks whether the referenced XMLNODE has at least one attribute. If it does, the method returns TRUE. It accepts no parameters.

```
DISPLAY xmlnode_var.HasAttributes()
```

GetAttributes()

This method returns the list of the attributes the referenced node has. It accepts no parameters and returns a value of XMLATTRIBUTELIST data type. This data type is a structured one, each item includes attribute name, attribute value, and attribute namespace (consisting of prefix and URI) of a single attribute. Each attribute is recorded as a separate item.

```
LET att_list = node.GetAttributes()
```

GetAttributeCount()

This method returns an integer value representing the number of the attributes the referenced node has. Each attribute together with its value and namespace is counted as a single item. This method accepts no parameters.

```
DISPLAY node.GetAttributeCount()
```

GetAttributeName()

This method returns the name of the attribute with the specified index belonging to the referenced node. It accepts a parameter of an INTEGER data type specifying the attribute index. Attributes acquire indexes depending on their position within the node: the first attribute which is the closest to the node name gets index 1 and so on. The returned value is of a character data type.

```
LET att_name = node.GetAttributeName(2)
```

GetAttributeValue()

This method returns the value of the attribute with the specified index belonging to the referenced node. It accepts a parameter of an INTEGER data type specifying the attribute index. Attributes acquire indexes depending on their position within the node: the first attribute which is the closest to the node name gets index 1 and so on. The returned value is of a character data type, even if the value of the attribute is numeric or of any other nature.

```
LET att_value = node.GetAttributeName(2)
```

Attribute Manipulation methods

The methods in this section are used to add and remove attributes to and from the referenced nodes.

AddAttribute()

This method is used to add an attribute to an existing node. The method requires an argument of an XMLATTRIBUTE data type which specifies the attribute to be added to the node.

You can assign an attribute retrieved from one node to another node, or you can create an attribute from scratch. However, if you want to add a new attribute to a node and not use an exising XMLATTRIBUTE value, you should use SetAttribute() or SetAttributeNS() method.

```
CALL node.AddAttribute(xmlattribute_v)
```

SetAttribute()

This method creates a new attribute for the referenced XMLNODE. It accepts two parameters of a character data type. The first parameter specifies the name of the attribute to add and the second one specifies the value of the added attribute.

```
LET att_node = xmlnode_var.SetAttribute("gender", "female")
```

SetAttributeNS()

This method creates a new attribute for the referenced XMLNODE. It accepts four parameters of a character data type. The first parameter specifies the namespace prefix of the attribute to add and the second one specifies the name of the attribute, the third one specifies the namespace URI and the fourth one specifies the value of the added attribute.

```
LET att_node = xmlnode_var.SetAttributeNS("pref", "gender",
"http://www.example.org/uri", "female")
```

RemoveAttribute()

This method removes the specified attribute from the referenced XMLNODE. It accepts a single parameter of a character data type which is the name of the attribute to remove. If the attribute with the specified name is not found, nothing is removed.

```
CALL xmlnode_var.RemoveAttribute("id")
```

RemoveAttributeNS()

This method removes the specified attribute from the referenced XMLNODE. It accepts two parameters of a character data type which is the name of the attribute to remove and its namespace prefix. If the attribute with the specified parameters is not found, nothing is removed.

```
CALL xmlnode_var.RemoveAttributeNS("id",
"http://www.example.org/uri")
```

XMLNODELIST Methods

The XMLNODELIST data type is mainly used when other methods return more than one node. They are used to store a list of XMLNODE values. However, this data type cannot be considered a structured data type and it cannot be referenced in the way the 4GL records and arrays are. To get individual XMLNODE values from a variable of this data type special methods are used.

GetSize()

This method returns the size of the referenced XMLNODELIST variable. The size indicates how many XMLNODE values the list contains. It accepts no parameters and returns an integer value which corresponds to the number of items. To single out separate nodes from the node list use the GetItem() method.

GetItem()

This method returns the XMLNODE value located at the specified index within this XMLNODELIST or NULL if no node is present at the index. It accepts a single value of the integer data type which is the index.

```
LET new_node = xmlnodelist_var.GetItem(12)
```

XMLATTRIBUTELIST Methods

The XMLATTRIBUTELIST values are returned by the GetAttributes() method used with an XMLNODE variable. The attribute list contains a set of XMLATTRIBUTE values which represent the attributes of the referenced node with all their values and namespaces. The only way to retrieve a single attribute from a node is to get the list of attributes using the method mentioned above and then to get a single item from the list. No methods in 4GL retrieve a single attribute from a node.

GetSize()

This method returns the size of the referenced XMLATTRIBUTELIST variable. The size indicates how many XMLATTAIBUTE values the list contains. It accepts no parameters and returns an integer value which corresponds to the number of items. To retrieve the items use the GetItem() method.

```
LET number_of_atts = att_list.GetSize()
```

GetItem()

This method returns the XMLATTRIBUTE value located at the specified index within this XMLATTRIBUTELIST or NULL if no node is present at the index. It accepts a single value of the integer data type which is the index.

```
LET att = att_list.GetItem(5)
```

XMLATTRIBUTE Methods

The XMLATTRIBUTE variables contain the attribute name, value and namespace. They can only be retrieved using the GetItem() method with an XMLATTRIBUTELIST variable. The methods described in this chapter are used for getting the information about a specific attribute and changing this information.

GetName()

This method returns a character string containing the name of the referenced attribute. It accepts no parameters.

```
DEFINE att XMLATTRIBUTE,
       att_name STRING
...
LET att_name = att.Getname()
```

GetValue()

This method returns a character string containing the value of the referenced attribute. It accepts no parameters.

GetNamespaceURI()

This method returns a character string containing the namespace URI associated with the referenced attribute. It accepts no parameters.

GetPrefix()

This method returns a character string containing the namespace prefix of the referenced attribute. It accepts no parameters.

SetName()

This method changes the name of the referenced attribute or assigns the name, if the attribute has none. It accepts a single parameter of a character data type which is the name of the attribute and does not return any value.

```
CALL att.SetName("newatt")
```

SetValue()

This method changes the value of the referenced attribute or assigns the value, if the attribute has none. It accepts a single parameter of a character data type which is the value of the attribute and does not return any value.

```
CALL att.SetValue("male")
```

SetNamespace()

This method changes the namespace of the referenced attribute or assigns the namespace, if the attribute has none. It accepts two parameters of a character data type: the namespace prefix is the first parameter and the namespace URI is the second parameter. This method does not return any value.

```
CALL att.SetNamespace("xsd", "http://www.w3.org/2003/XMLSchema")
```

XMLSAXPARSER Methods

The XMLSAXPARSER datatype implements a SAX parser for 4GL. This allows you to register 4GL callback functions for the following SAX events:

Event	When Triggered	4GL Function Parameters
StartDocument	When ReadXML method loads the document	No parameters
EndDocument	After all the other methods for the loaded document were executed	No parameters
StartElement	When the parser hits the opening tag of an element node	(STRING name, XNLATTRIBUTELIST attributes)
EndElement	When the parser hits the closing tag of the element node	(STRING name)
CharacterData	When the parser hits a text node	(STRING data)
Comment	When parser hits a comment node	(STRING data)
ProcessingInstruction	When the parser hits a processing instruction node	(STRING target, STRING data)
CDATA	When the parser hits the CDATA node	(STRING data)

When the event is trapped by the corresponding method listed below, the specified parameters are passed to the 4GL function referenced by the method.

ReadXMLFile()

This method loads the specified XML file and starts the SAX parser. It accepts a single parameter of a character data type containing the file name and the path to the file, if needed.

```
CALL xmlsaxparcer_var.ReadXMLFile("C:\xml\my_xml.xml")
```

If this method is not executed, all the other methods will have no effect.

This method should be placed in the code after all the other required methods. Other XMLSAXPARSER methods should be executed before the document is loaded, because they actually take effect only during the execution of the ReadXML() method which requires them to be declared beforehand.

StartDocumentFunction()

This method sets the callback function name for the StartDocument SAX event. It accepts a single parameter of a character data type which is the name of the 4GL function to be called on a StartDocument event.

```
CALL xmlsaxparcer_var.StartDocumentFunction("4gl_func")
```

This method does not pass any parameters to the associated 4GL function and it triggered once for each processed document when the processing starts.

EndDocumentFunction()

This method sets the callback function name for the EndDocument SAX event. It accepts a single parameter of a character data type which is the name of the 4GL function to be called on a EndDocument event. It passes no parameters to the associated 4GL function and is triggered only once for each processed document when the document ends.

StartElementFunction()

This method sets the callback function name for the StartElement SAX event. It accepts a single parameter of a character data type which is the name of the 4GL function to be called on a StartElement event. This method passes two parameters to the referenced 4GL function. The first parameter is the name of the element and the second is the list of parameters in the form of an XMLATTRIBUTELIST value.

EndElementFunction()

This method sets the callback function name for the EndElement SAX event. It accepts a single parameter of a character data type which is the name of the 4GL function to be called on a EndElement event. This method passes a single argument to the referenced 4GL function which is the name of the element.

CharacterFunction()

This method sets the callback function name for the CharacterData SAX event. It accepts a single parameter of a character data type which is the name of the 4GL function to be called on a CharacterData event. This method passes a single argument of a character data type to the referenced 4GL function which is the content of the text node.

CDATAFunction()

This method sets the callback function name for the CDATA SAX event. It accepts a single parameter of a character data type which is the name of the 4GL function to be called on a CDATA event. This method passes a single argument of a character data type to the referenced 4GL function which is the content of the CDATA node.

ProcessingInstructionFunction()

This method sets the callback function name for the ProcessingInstruction SAX event. It accepts a single parameter of a character data type which is the name of the 4GL function to be called on a ProcessingInstruction event. This method passes two arguments of a character data type to the referenced 4GL function which are the target and the data of the processing instruction.

CommentFunction()

This method sets the callback function name for the Comment SAX event. It accepts a single parameter of a character data type which is the name of the 4GL function to be called on a Comment event. This method passes a single argument of a character data type to the referenced 4GL function which is the content of the comment node.

How to Use the SAX Parser

The SAX parser methods should be provided with 4GL functions that it will call when an event happens. The 4GL function parameters column in the table at the start of this chapter specifies which parameters will be passed to the 4GL function called.

For example, we want to parse a simple XML document below:

```
<something>
  <potato>this is a vegetable</potato>
</something>
```

Then somewhere in the 4GL code you have the following lines:

```
CALL sax_var.StartElementFunction("start_func")
...
CALL sax_var.CharacterData("char_func")
```

The SAX parser, when hitting the <something> element, will call "start_func" function, because it is specified as the function to be called in the case StartElement event occurs. You function will be given two parameters: a string, containing value "something" and a XMLNODE value containing a list of attributes which will be empty in this case, because the node <something> has no attributes.

Next, it will call your 'StartElement' function again, this time with a string containing <potato> and an empty attribute list.

Next it will call the function "char_func" triggered by the event 'CharacterData', with a parameter of a string - "this is a vegetable". This will continue until the whole file is parsed.

Examples

This section contains examples for both the DOM methods and the SAX methods.

Parsing a File using DOM

This application simply parses the specified XML file and gives a count of the type of nodes that are present within the XML. It requires an xml file passed to it as an argument. Unless you specify the file name and path to it as an argument during the application launching, the program will produce a warning and terminate.

```
GLOBALS

DEFINE nbElt INTEGER

DEFINE nbAttr INTEGER

DEFINE nbComment INTEGER

DEFINE nbPI INTEGER

DEFINE nbTxt INTEGER

DEFINE nbCData INTEGER

END GLOBALS


MAIN

 DEFINE document XMLDOCUMENT

  DEFINE ind INTEGER

  IF num_args() != 1 THEN

    DISPLAY "Usage: ", arg_val(0), " <xmlfile>"

    EXIT PROGRAM

  END IF

  CALL document.Load(arg_val(1), false)

  DISPLAY "Counting doc.."

  CALL CountDoc(document)

  DISPLAY "Results: "

  DISPLAY " Elements: ",nbElt
```

```
  DISPLAY " Attributes:",nbAttr
  DISPLAY " Comments: ",nbComment
  DISPLAY " PI: ",nbPI
  DISPLAY " Texts: ",nbTxt
  DISPLAY " CData: ",nbCData
END MAIN

FUNCTION CountDoc(d)
  DEFINE d XMLDOCUMENT
  DEFINE n XMLNODE
  LET n = d.GetFirstDocumentNode()
  WHILE (n IS NOT NULL)
   CALL CountN(n)
    LET n = n.GetNextSibling()
  END WHILE

END FUNCTION

FUNCTION CountN(n)
DEFINE n XMLNODE
  DEFINE child XMLNODE
  DEFINE next XMLNODE
  DEFINE node XMLNODE
  DEFINE ind INTEGER
  DEFINE name STRING
  DEFINE att_list XMLATTRIBUTELIST
  DEFINE att XMLATTRIBUTE
  IF n IS NOT NULL THEN
```

```
IF n.GetNodeType() == "COMMENT_NODE" THEN
 LET nbComment = nbComment + 1
END IF
IF n.GetNodeType() == "PROCESSING_INSTRUCTION_NODE " THEN
 LET nbPI = nbPI + 1
END IF
IF n.GetNodeType() == "ELEMENT_NODE" THEN
  LET nbElt = nbElt + 1
END IF
IF n.GetNodeType() == "TEXT_NODE" THEN
  LET nbTxt = nbTxt +1
END IF
IF n.GetNodeType() == "CDATA_SECTION_NODE" THEN
LET nbCData = nbCData + 1
END IF

IF n.HasChildNodes() THEN
 LET name = n.GetLocalName()
 LET child = n.GetFirstChild()
  WHILE (child IS NOT NULL)
    CALL CountN(child)
    LET child = child.GetNextSibling()
  END WHILE
END IF
IF n.HasAttributes() THEN
 LET att_list = n.GetAttributes()
 FOR ind = 1 TO n.GetAttributeCount()
    LET att = att_list.GetItem(ind)
```

```
            LET nbAttr = nbAttr + 1

          END FOR

        END IF

      END IF

    END FUNCTION
```

Parsing a File Using SAX

This application simply parses the specified XML file and gives a count of the type of nodes that are present within the XML. It requires an xml file passed to it as an argument. Unless you specify the file name and path to it as an argument during the application launching, the program will produce a warning and terminate.

```
    GLOBALS

    DEFINE nbElt INTEGER

    DEFINE nbAttr INTEGER

    DEFINE nbComment INTEGER

    DEFINE nbPI INTEGER

    DEFINE nbTxt INTEGER

    DEFINE nbCData INTEGER

    END GLOBALS


    MAIN

      DEFINE parser XMLSAXPARSER


      CALL parser.StartElementFunction("element")

      CALL parser.CommentFunction("comment")

      CALL parser.ProcessingInstructionFunction("process")

      CALL parser.CharacterFunction("text")

      CALL parser.CDATAFunction("cdata")


      IF num_args() != 1 THEN
```

```
    DISPLAY "Usage: ", arg_val(0), " <xmlfile>"
    EXIT PROGRAM
  END IF

  DISPLAY "Counting doc.."
  CALL parser.ReadXMLFile(arg_val(1))

 DISPLAY "Results: "
  DISPLAY " Elements: ",nbElt
  DISPLAY " Attributes:",nbAttr
  DISPLAY " Comments: ",nbComment
  DISPLAY " PI: ",nbPI
  DISPLAY " Texts: ",nbTxt
  DISPLAY " CData: ",nbCData

END MAIN

FUNCTION element(name, attris)
  DEFINE name VARCHAR(255)
  DEFINE attris  XMLATTRIBUTELIST
  LET nbElt = nbElt + 1
  LET nbAttr = nbAttr + attris.GetSize()
END FUNCTION

FUNCTION comment(data)
  DEFINE data VARCHAR(255)
  LET nbComment = nbComment + 1
END FUNCTION
```

```
FUNCTION process(data)
  DEFINE data VARCHAR(255)
  LET nbPI = nbPI + 1
END FUNCTION

FUNCTION text(data)
  DEFINE data VARCHAR(255)
  LET nbTxt = nbTxt + 1
END FUNCTION

FUNCTION cdata(data)
  DEFINE data VARCHAR(255)
  LET nbCData = nbCData + 1
END FUNCTION
```