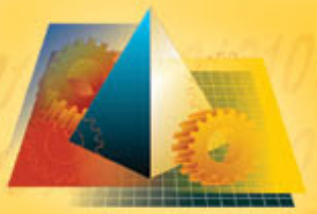


Clarion

Clarion XML Support and Reference Guide

COPYRIGHT 1994- 2009 SoftVelocity Incorporated. All rights reserved.

This publication is protected by copyright and all rights are reserved by SoftVelocity Incorporated. It may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from SoftVelocity Incorporated.

This publication supports Clarion. It is possible that it may contain technical or typographical errors. SoftVelocity Incorporated provides this publication “as is,” without warranty of any kind, either expressed or implied.

SoftVelocity Incorporated
2335 East Atlantic Blvd.
Suite 410
Pompano Beach, Florida 33062
(954) 785-4555
www.softvelocity.com

Trademark Acknowledgements:

SoftVelocity is a trademark of SoftVelocity Incorporated.
Clarion™ is a trademark of SoftVelocity Incorporated.
Microsoft®, Windows®, and Visual Basic® are registered trademarks of Microsoft Corporation.
All other products and company names are trademarks of their respective owners.

Printed in the United States of America (0107)

Contents:

Introduction

Purpose

The Clarion XML support encapsulates a standard XML parser interface and provides easy to use import/export related API functions and classes. The XML support utilizes the CenterPoint XML C++ class library for reading and writing XML documents using DOM (Document Object Model) Level 1 and Level 2 conforming interfaces. CenterPoint XML is based on expat, the well-known XML Parser Toolkit.

There are three entry points provided so that the developer can work with XML data at the level he/she chooses.

1. **Work directly with the DOM methods prototyped in *cpxml.inc***
2. **Use the Procedure based API**
    - **FileToDOM**, **QueueToDOM**, **ViewToDOM** – provides export data from the corresponding Clarion's structures to XML DOM Document object
    - **DOMToFile, DOMToQueue** – provide import data from XML DOM Document object to the corresponding XML structures
    - **XMLFileToDOM, XMLStringToDOM** – provide creating a DOM Document object based on XML data from XML file/string
    - **DOMToXMLFile, DOMToXMLString** – provide storing a DOM Document data to an XML file/string.
3. **Use the OOP Classes**
    - The classes encapsulate all the functionality of the Procedure based API, and also provide extended functionality.

Definitions, acronyms and abbreviations

DOM – Document Object Model (in this document – document object model used by XML parser)

XML – eXtensible Markup Language

API – Application Programming Interface

References

For more information regarding DOM and XML, visit HTTP://www.w3.org for the latest Level 1 and Level 2 DOM specifications.

Document overview

This document provides a description of the functionality of the Clarion/XML extension API. The functionality describes both a procedure based API and describes the more advanced functionality of the Object Oriented API.

General description

The XML Export/Import Support API is implemented as set of Clarion's functions and classes located in the cpxml.inc/cpxml.clw source files and the C70cpxml.dll. The C70cpxml.dll contains the XML parser and wrapper functions to work with the XML parser from within Clarion.

The following files need to be distributed for your application to work with the XML Support API:

- C70cpxml.dll
- C70xmlty.dll

In addition the following pragma defines need to be specified in your project file:

XMLLibDllMode => 0

XMLLibLinkMode => 1

The template support automatically includes these project defines, but you will need to add this manually for any hand coded projects.

Local linking of the XML Export/Import Support API is currently not supported.

OOP Clarion/XML extension

Introduction

This section provides a detailed description of the features related to the export and import of Clarion's complex data types such as FILE, VIEW, QUEUE and GROUP to and from XML format using the supplied classes. The described functionality is not equal to the procedure based approach described in the Procedure Based API section. The Clarion XML Import/Export API OOP implementation provides a more flexible approach and extends the functionality of the procedure based API. Both the OOP and Procedure based implementations are supported, so a developer can use either OOP or the procedure based API in accordance with the required functionality

Purpose

The main purpose of providing a class oriented API is to provide additional functionality and control for the import and export of data from the user's application. Class objects as a data type having state provide a user-friendlier environment to support not only predefined import and export scenarios, but also custom usage.

The Clarion XML Class Model

Clarion XML Import/Export class overview

The main class responsible for import/export is **XMLExchange.** It supports algorithms for import/export from/to XML. It uses a derived **StructWrapper** class for accessing Clarion specific data and can either export them to XML or import XML data into them.

StructWrapper is a base class that publishes an interface for accessing Clarion complex data types. There are separate implementations of the **StructWrapper** class for each Clarion specific data type such as **FileWrapper, GroupWrapper, QueueWrapper, ViewWrapper (TreeViewWrapper)** for FILE, GROUP, QUEUE and VIEW accordingly**.**

XMLNameMap class supports the mapping of Clarion structures field names and XML attribute names. It can be used for two purposes. The first one provides for attribute/tag aliases for tag names used in the XML document instead of the field names defined in the Clarion program. The second functionality provides custom (not default) conversion of the exported values. The Procedure based API also supports the functionality to format data during exchange. The mapping functionality is optional. The mapping information can be defined with the help of **XMLNameMap** class interface or it can be loaded from an XML file by the same manner as importing any other XML structure. **XMLNameMap** class supports partial import/export functionality. In the case where name mapping is not enabled, data exchange is applied for all fields. If a name map is applied, then only fields specified by mapping pairs are imported/exported.

The set of classes provides the following options:

- A Name mapping facility, which provides the possibility to define formatting information with Clarion's **picture token**. The developer can also override the standard formatting behavior by specifying a user-defined formatting method in the **XMLNameMap** class. The Procedure-based API also supports this functionality.
- The following attributes can be added to the XML's root node:
  - **Timestamp** – contains date/time of export operation
  - **Version** – contains version of the cpXML module used
  - **Userinfo** – contains some information provided by user
  - **Style** - contains XML style (ADO_26 (attribute-based) or ADO_NET (tag-based))

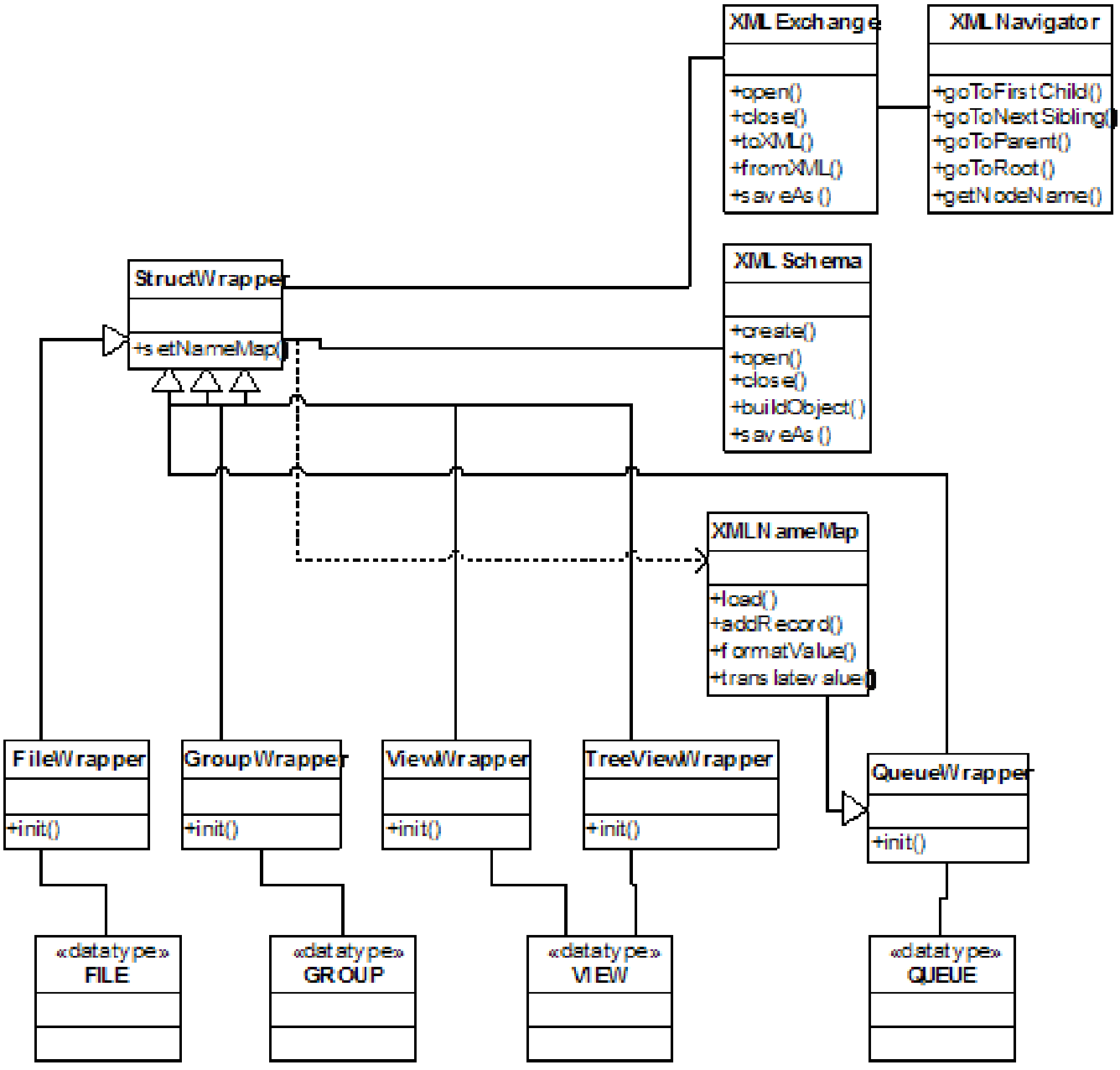
  This additional information can by used by import functions (for example, for version checking and XML style recognizing) and also can be available for user on import stage.
- When the Clarion/XML Classes are used, the DOM Document object is hidden inside **XMLExchange** class, so it's not the user's responsibility to release the Document object.
- The **TreeViewWrapper** class provides export for a VIEW structure with several nested JOINs into XML with tree-like hierarchical structure.

XMLSchemas

Future versions will contain an **XMLSchema** class to support XML schema related operations

Class Diagram

The following diagram illustrates the relations between the main classes and also shows the main methods signatures:



XML structure

The structure of an XML element, produced by the **toXML** method of the **XMLExchange** object depends on the style (**setStyle** method). Two styles are supported attribute-based (**XMLStyle: ADO_26**) and tag-based (**XMLStyle: ADO_Net)**.

In the first case, the fields of a Clarion structure are represented by attributes: the name of attribute is the same as name of the field and the value of the attribute equal to the corresponding field's value. Prefixes are removed by default.

In the second case, each field is represented by an XML element, with tag name equals to a field's name and content equals to a field's value. Prefixes are removed by default.

For example:

attribute-based representation of a record:

```
<row lastname="Wright" firstname="Carl" phone="2221697"/>
```

tag-based representation of the same record:

```
<Table>
       < lastname>Wright </lastname>
       < firstname>Carl </firstname>
       <phone>2221697</phone>
</Table>
```

The data structure is represented as a sequence of XML elements, and corresponds to the records of the exported structure enclosed in a tag.

For example, for the attribute-based style, the structure can be the following:

```
<data>
   <row lastname="Wright" firstname="Carl" phone="2221697"/>
   <row lastname="Curry" firstname="Charmaine" phone="3339716"/>
</data>
```

Example

The following example is implemented with the set of classes and demonstrates the formatting feature.

Consider the following XML files:

CUSTOMER.XML (Contains information about customers)

```
<?xml version="1.0"?>
<data>
   <row lastname="Wright" firstname="Carl" phone="2221697"/>
   <row lastname="Curry" firstname="Charmaine" phone="3339716"/>
</data>
```

NAMEMAP.XML (Contains information about name mapping)

```
<?xml version="1.0"?>
<data>
<row ClarionName="LastName" XMLName="Name" Picture=""/>
<row ClarionName="Phone" XMLName="Tel" Picture="@P###-##-##P"/>
</data>
```

Code Example:

```
PROGRAM
CC        BYTE             !temporary flag

Customer  QUEUE            !queue to store imported data
LastName   STRING(30)
Phone      STRING(30)
          END

nameMap   XMLNameMap       !object with name mapping information
xExch     XMLExchange      !object to provide export/import operations
qWrp      QueueWrapper     !wrapper object

  CODE
!Import data from file Customer.xml to Customer QUEUE.
!Destination QUEUE contains only LastName, Phone fields so only data for
!these fields are imported

 CC = xExch.open('Customer.xml')

 IF (cc)
  !Initialize the wrapper object
  qWrp.init(Customer)

  !put the data into QUEUE
  xExch.fromXML(qWrp)

!Export data from Customer QUEUE to a file with name mapping and
!formatting

  ! Load name mapping information from NAMEMAP.XML
  CC = nameMap.load('NameMap.xml')
   if (cc)
    qWrp.setNameMap(nameMap)             !attach name mapping information
    xExch.close                          !clear the object
    xExch.create                         !create new XML document
    xExch.toXML(qWrp)                    !put data from Customer to XML
    cc = xExch.saveAs('CustomerData.Xml')!save XML doc to file
     if(NOT cc)
      stop('Exporting error')
     end
   else
    stop('NameMap.xml importing error')
   end
 else
  stop('Customer.xml importing error')
 end
```

The result file *CUSTOMERDATA.XML* contains the following information (note that the format of the fields have been changed):

CUSTOMERDATA.XML

```
<?xml version="1.0"?>
<data>
   <row name="Wright" tel="222-16-97"/>
   <row name="Curry"   tel="333-97-16"/>
</data>
```

Technical Notes

Class based XML support

The XML support consists of several classes that support XML Export/Import functionality.

The main classes are as follows:

StructWrapper

A base class for handling complex Clarion data types. Its methods provide for accessing fields of a Clarion structure at run-time and provide additional information about fields such as a field's name and type. Also, the class may have reference to an **XMLNameMap** object to make available the mapping of Clarion field names to XML names, and for user defined processing of imported/exported values.

The class is a base class for the following wrapper classes: **GroupWrapper**, **QueueWrapper, FileWrapper, ViewWrapper (TreeViewWrapper).** A developer will not use this class directly. Only the derived type-specific classes are used at the user level. So, for each data type you need to export/import, create the associated wrapper class for that Clarion structure.

XMLExchange

Encapsulates import/export specific functionality. It uses the **StructWrapper** methods for accessing individual fields to export or import values to/from XML. **XMLExchange** provides additional methods to customize the import/export process.

XMLNameMap

An auxiliary class to simplify work with name mapping structures. It contains an underlying queue with name mapping and formatting information. It also allows you to specify user-specific custom data formatting functionality.

XMLNavigator

An auxiliary class that allows you to navigate through XML DOM Document stored in **XMLExchange** object.

StructWrapper and Clarion data type-specific classes

The **StructWrapper** class contains several methods. These methods should not be used directly, as the methods are used by **XMLExchange** objects. The only method that should be used by a developer directly is the **Init** type-specific methods in the derived classes. The text below contains the public methods' declarations of the **StructWrapper** class.

getFieldsCount **procedure, signed**

The **getFieldsCount** method returns the number of fields in the Clarion's structure available for import export. In case that mapping is enabled, this number can be less than actual number of the fields in the structure.

getFieldValueByIndex **procedure(unsigned *index*), string**

index The field index. If mapping is enabled, only mapped fields are counted.

The **getFieldValueByIndex** method returns the field value as a string by specified index. The returned value is the result of the transformation defined in the mapping (if enabled), or the default conversion to a string.

setFieldValueByIndex **procedure(unsigned *index*, string *value*), unsigned**

index Clarion name of the field
value The string value to be set

The **setFieldValuebyIndex** method returns *CPXMLErr:NoError* if a value has been set, or *CPXMLErr:FieldNotFound* if a value has not been set. If mapping is enabled only mapped fields are counted.

getFieldValueByName **procedure(string *field*),string**

field The Clarion name of the field

The **getFieldValueByName** method returns the field value as a string from the specified *field* name. The value returned is a result of transformation defined in the mapping (if enabled), or by default conversion to string. An empty string is returned in case there is no field corresponding to the specified field name.

getFieldValueByXMLName **procedure(string *field*),string**

field The XML name of the field

The **getFieldValueByXMLName** method returns the field value as a string from the specified *field* name. The returned value is result of transformation defined in the mapping (if enabled), otherwise default conversion to string is applied. An empty string is returned in case there is no field corresponding to specified field name.

setFieldValueByName **procedure(string *field*,string *value*), unsigned**

field Clarion name of the field
value The string value to be set

The **setFieldValueByName** method returns *CPXMLErr:NoError* if the *value* has been set, or *CPXMLErr:FieldNotFound* otherwise. If mapping is enabled, only mapped fields are processed.

setFieldValueByXMLName procedure(string *field*,string *value*), unsigned

field The XML name of the field
value The string value to be set

The **setFieldValueByXMLName** returns *CPXMLErr:NoError* if the *value* has been set, or *CPXMLErr:FieldNotFound* otherwise. If mapping is enabled, only mapped fields are processed.

getFieldLabel procedure(unsigned *cou*),string

cou – index of the field

The **getFieldLabel** method returns the Clarion label of the field identified by the field's index *cou* value.

getXMLFieldLabel procedure(unsigned *cou*),string

cou – index of the field

The **getXMLFieldLabel** returns the XML label of the field.

getFieldType procedure(unsigned cou),unsigned

The **getFieldType** method is necessary to produce XML schema. (Not implemented in this version)

getFieldSize procedure(*unsigned size, *unsigned precision)

The **getFieldSize** method gets size of the field (for strings, DECIMAL etc). (Not implemented in this version)

getCapacity procedure, unsigned, virtual

The **getCapacity** method returns the maximum possible number of rows (capacity) in the corresponding Clarion structure. For example, a GROUP will return 1, and a QUEUE will return -1 (infinite).

startIterate procedure,virtual

The **startIterate** method sets the inner current row counter to its initial state (the beginning of the rows' sequence). This method should be called before the **getNextRow** method.

getNextRow procedure, byte, virtual

The **getNextRow** method gets the next row (record). If successful, **getNextRow** returns *CPXMLErr:NoError*. If not successful, one of the following errors can be returned:

CPXMLErr:EOF
CPXMLErr:IllegalFunctionCall

clearBuffer **procedure,virtual**

The **clearBuffer** method clears the buffer of the Clarion structure.

addRow **procedure,byte,virtual**

The **addRow** method adds a new row to the Clarion structure. If successful, **addRow** returns *CPXMLErr:NoError*. Otherwise, **addRow** returns *CPXMLErr:InternalError*.

setNameMap **procedure(XMLNameMap *map*), byte**
setNameMap **procedure(MapQueue *map*), byte**

map A structure with mapping information for export/import operations

The **setNameMap** methods set name mapping structures that are used on export/import operations. If successful, setnameMap returns *CPXMLErr:NoError.*

getNameMap **procedure(),*XMLNameMap**

The **getNameMap** method returns the pointer to the inner **XMLNameMap** object

isMappingEnabled **procedure,byte**

The **isMappingEnabled** method returns **TRUE** if mapping is enabled.

SetMappingEnabled **procedure(byte flag=true)**

Flag Set to TRUE if mapping is enabled

The **setMappingEnabled** method sets the name mapping *flag*.

Hierarchical support methods

The following methods are necessary for support of hierarchical structures (i.e., a VIEW. The resulting XML file will have a tree-like structure).

getChild **procedure,*StructWrapper,virtual**

The **getChild** method returns the reference to the first child node.

getParent **procedure,*StructWrapper,virtual**

The **getParent** method returns the reference to the parent node.

getNextSibling **procedure,*StructWrapper ,virtual**

The **getNextSibling** method returns the reference to the next sibling node.

updateRowPos **procedure,virtual**

The **getNextSibling** method stores the inner position value to the position of current row.

isRowPosChanged **procedure,byte,virtual**

The **getNextSibling** method tests if the row position was changed since the last call of the **updateRowPos** method.

Derived Classes

Derived, type-specific classes **GroupWrapper, QueueWrapper, FileWrapper,** and **ViewWrapper (TreeViewWrapper)** may override several virtual methods of the base class. Also, these classes contain a type-specific initialization method that is used to fill inner structures with the necessary information from the corresponding Clarion structure. In the current implementation, this method has the following signature:

Init **procedure(*<type> *clastr* , *removePrefix*=true),byte**

clastr	A pointer to the corresponding Clarion's structure
removePrefix	A flag, set to TRUE, if the prefixes of Clarion labels should be removed.

The **Init** method returns *CPXMLErr:NoError* if successful, and *CPXMLErr:InvalidArgument* if the Init was not successful.

Each derived wrapper class contains a **getObject** method that returns the reference to the wrapped structure. The usage is as follows:

getObject **procedure, <type>**

where <*type*> equals the corresponding object's type (QUEUE, FILE etc).

XMLNameMap class and MapQueue structure

The **MapQueue** QUEUE is provided to support name mapping information. It contains mapping pairs <*ClarionName, XMLName*>**,** and a *Picture* field that is used to provide formatting information.

XMLNameMap Class is an alternative for the **MapQueue** structure. The **XMLNameMap** class provides name mapping and formatting information and is used by **StructWrapper** class to support name mapping and formatting features on export/import.

You can load necessary mapping/formatting information from a XML file, or from a **MapQueue** structure.

This class also contains a virtual **formatter** method. You can override it to implement user-defined formatting.

XMLNameMap Class Methods

construct **procedure**
destruct **procedure**

load **procedure(MapQueue *map*),byte**

map The label of the mapping queue

The **load** method loads mapping and formatting information.
If successful, **load** returns *CPXMLErr:NoError*. Otherwise, **load** returns *CPXMLErr:InvalidArgument*.

addRecord **procedure(string *clarionName*, string *xmlName*, string *picture*),byte**

clarionName The name of Clarion field
xmlName The corresponding XML name
picture The picture token for value formatting

The **addRecord** method adds a new record with name mapping and formatting information. If successful, **addRecord** returns TRUE. Otherwise, **addRecord** returns FALSE.

StructWrapper class name mapping and formatting support

The following methods are used by **StructWrapper** class to support name mapping/formatting.

translateName **procedure(string *name*, byte *toXML*=true),string**

name	The name of Clarion or XML field
toXML	If set to TRUE then translate to XML name; otherwise translate to Clarion name

The **translateName** method translates a name to and from an XML name. It returns the string result of the translation.

formatValue **procedure(string *name*, string *value*, byte *toXML*=true),string,virtual**

name	The name of a Clarion or XML field
value	The value to be formatted
toXML	If set to true then formats to XML; performs a deformat otherwise.

The **formatValue** method formats a value to and from the XML form. It returns the string result of the formatting.

Note:
The standard behavior of the function is to apply formatting according to the *Picture* in the *MappingQueue*. You can override the default behavior with your own function.

XMLExchange class

The **XMLExchange** class provides main functionality for export/import. It contains an inner DOM Document object that is used as intermediate storage for exported/imported data. You can load data in the Document object from a **StructWrapper** object, or put data from the Document object to a **StructWrapper** object. This class supports methods to save data to a XML file and to load data from an XML file. Also, you can set/get a number of export/import parameters.

XMLExchange Class Methods

construct procedure

Constructor of the class

destruct procedure

Destructor of the class

open procedure(string *path*),byte

path A valid path to the opened XML file

The **open** method opens an XML file. The method returns *CPXMLErr:NoError* if successful, or *CPXMLErr:XMLReadFail* if not successful**.**

createXML procedure(<string *schemaURL*>),byte

schemaURL The URL of a schema to be included as an attribute of the root node.

The **createXML** method creates a new XML Document. It returns *CPXMLErr:NoError* if successful, or *CPXMLErr:CreateDOMFail* otherwise.

close procedure

The **close** method closes access to XML DOM. After this method any operation with XML is not available.

saveAs procedure(string *path*),byte

path The valid path of the file to save the XML data to.

The **saveAs** method saves content to an XML file. If successful, **saveAs** returns *CPXMLErr:NoError* . Otherwise, it returns *CPXMLErr:XMLWriteFail*.

toXML **procedure(StructWrapper *sw*, <string *nodeName*>,byte *positionFlag*=APPEND),byte**

sw an instance of the **StructWrapper** class to load from

nodeName The name of the created node (optional parameter). The parameter is used only if *positionFlag* =XMLPos:APPEND_CHILD**.**

PositionFlag Determines the position to load data to. Can be the following:

XMLPos:APPEND data is appended to the current node
XMLPos:APPEND_CHILD the child node for current node with name *nodeName* is created. Data is loaded into the new created node.

The **toXML** method loads data from a Clarion structure. It returns *CPXMLErr:NoError* if successful. Otherwise, toXML returns *CPXMLErr:CreateDOMFail***.**

NOTE:
If some data has been previously loaded, then the method just appends the new data to the existing data. You can select XML node (in the **setNode** method) to point at the position where the data should be loaded to.

fromXML **procedure(StructWrapper sw),byte**

sw An instance of the **StructWrapper** class to put the data to

The **fromXML** method puts the data from the DOM Document object to a Clarion's object. It returns *CPXMLErr:NoError* if successful. Otherwise, the method returns *CPXMLErr:ImportFail*.

NOTE:
You can select the necessary XML node for this operation with the **setNode** method.

setNode **procedure(*Node *nd*)**

nd A pointer to the XML DOM Node object.

The setNode method sets the current XML node.

getNode **procedure,*Node**

The **getNode** method returns the pointer to the current XML DOM Node object.

getNavigator **procedure,*XMLNavigator**

The **getNavigator** method gets the instance of the XMLNavigator class. It returns a pointer to the XMLNavigator object.

getDocument **procedure,*Document**

The **getNavigator** method returns the reference to XML DOM Document object.

attachDOM **procedure(*Document *doc*)**

*d*oc A pointer to an XML DOM Document

The **attachDOM** method attaches an existing document to the wrapper.

NOTE:
After calling this method all export/import operation will work with the attached document. On destruction/closing of the XMLExchange object the release method for the Document is automatically called.

detachDOM **procedure, *Document**

The **attachDOM** method detaches the wrapper's DOM Document. It returns the pointer to the inner XML DOM Document object.

NOTE:
After call of the method no export/import operations are available. It's the responsibility of the user to release the detached Document.

setStyle **procedure(XMLStyle style)**

style specify the style of generated XML. It can be *XMLStyle:ADO_26* (for attribute-based style) or *XMLStyle:ADO_Net* (for tag-based)

The **attachDOM** method sets the style that is used for import/export operations.

getTimeCreated **procedure, string**

The **getTimeCreated** method returns the string with date and time information assigned to the current node in the following format:

Yyyy/mm/dd hh:mm:ss

getVersion **procedure, string**

The **getVersion** method returns the string with the version information assigned to the current node.

getUserInfo **procedure, string**

The **getUserInfo** method returns the string with user defined information assigned to the current node.

setUserInfo **procedure(string *ui*)**

ui A string containing user-defined information

The **getUserInfo** method assigns the user-defined string to the current XML node

setSpecAttrOn **procedure(byte *flag*=true)**

flag Specify mode. It can be true or false.

The **setSpecAttrOn** method sets the mode (On/Off) for special attributes (Version, style, timestamp etc.).

NOTE:
By default all special attributes are included and added to the node after each toXML method call.

setRootTagName **procedure(string *tagName*)**

tagName The name of the root tag. By default the name of the root tag is *dataroot*.

The **setRootTagName** method sets the root tag name.

setRowTagName **procedure(string *tagName*)**

tagName The name of the row tag. By default the name of the row tag is *row* for style *XMLStyle:ADO_26* and *table* for style *XMLStyle:ADO_Net*.

The **setRowtagName** method sets the name of the tag that is used for a row of data

setMaxRowCount **procedure(unsigned *maxCou*), byte**

maxCou The maximum number of rows to be retrieved. A value of 0 means infinite. *maxCou* cannot be negative.

The **setMaxRowCount** method sets the maximum number of rows to be retrieved. If successful, the method returns *CPXMLErr:NoError* .Otherwise, it will return *CPXMLErr:IllegalParameter*.

XMLNavigator Class

The **XMLNavigator** class provides basic functionality to navigate through an XML Document object contained in the **XMLExchange** object. You can retrieve the reference to the instance of the class with **getNavigator** method of an **XMLExchange** object.

XMLNavigator Class Methods

getXMLExchangeNode procedure()

The **getXMLExchangeNode** method gets the current node of the XMLExchange object and sets the current node of XMLNavigator to the retrieved value

setXMLExchangeNode procedure()

The **setXMLExchangeNode** method sets the current node of the XMLExchange object to the current node of the XMLNavigator.

goToRoot procedure(), byte

The **goToRoot** method moves the inner current node pointer to the root node of the DOM Document. It returns a TRUE if successful, and FALSE otherwise.

goToParent procedure(), byte

The **goToParent** method moves the current node pointer to the parent node relative to the current node value. It returns a TRUE if successful, and FALSE otherwise.

goToFirstChild procedure(), byte

The **goToFirstChild** method moves the current node pointer to the first child node relatively to current node value. It returns a TRUE if successful, and FALSE otherwise.

goToNextSibling procedure(), byte

The **goToNextSibling** method moves the current node pointer to the next sibling node relatively to current node value. It returns a TRUE if successful, and FALSE otherwise.

getNodeName procedure(), string

The **getNodeName** method returns the current node name

getNodeAttributes **procedure(AttrQueue *aq*), byte**

aq The label of a QUEUE (AttrQueue). The AttrQueue contains records with following fields:
AttrName – contains name of an attribute
AttrValue – contains value of the attribute

The **getNodeAttributes** method fills the QUEUE structure with attributes information in the form of pairs:

<*attribute_name,attribute_value*>

getNodeAttributes returns a TRUE if successful, and FALSE otherwise.

Error codes

The following return codes can be returned by methods

Codes	Description
0	No error occurred
1	Error reading an XML file
2	Error writing to an XML file
3	Error: an attempt to use the same field name twice (duplicate name)
4	name cannot be used as XML tag/attribute identifier
5	Error creating XML DOM Document object
6	End Of File
7	Invalid Argument
8	Illegal Function Call
9	Not supported
10	Internal error
11	Field not found
12	General Error
13	Illegal parameter
14	Import Failure

XML based export/import for Clarion's structures

Introduction

Current implementation of the XML support consists of a set of Clarion's functions that provide the following functionality:

- Exporting data from FILE, VIEW or QUEUE structures to an XML DOM Document object
- Importing data from a XML DOM Document object to FILE or QUEUE structures
- Exporting XML DOM Document object's data to a XML file or string
- Importing data from XML file or string to an XML DOM Document object

This set of function provides possibility to export data from FILE, VIEW or QUEUE structures to XML file (or string) and import data from XML file (or string) to FILE or QUEUE structures.

The implementation of these functions is based on the C++ library from CenterPoint that provides basic functionality to work with XML data. The classes from the library are available from Clarion through a set of wrapper functions.

User model

The current set of functions includes the following:

- **FileToDOM**, **QueueToDOM**, **ViewToDOM** – provide export data from the corresponding Clarion's structures to XML DOM Document object
- **DOMToFile, DOMToQueue** – provide import data from XML DOM Document object to the corresponding XML structures
- **XMLFileToDOM, XMLStringToDOM** – provide creating a DOM Document object based on XML data from XML file/string
- **DOMToXMLFile, DOMToXMLString** – provide storing a DOM Document data to an XML file/string.

1. The export/import functions (**FileToDOM**, **QueueToDOM**, **ViewToDOM, DOMToFile, DOMToQueue)** support an additional optional parameter that contains name mapping information (If a user omits the parameter then these functions work without any name mapping). The name mapping parameter – is a QUEUE structure with string fields (mapping pair) **ClarionName** (contains name of Clarion's field)**, XMLName** (contains the corresponding name for XML file). The developer can fill the QUEUE structure "manually" with the **add function** or import the structure from an existing XML file (using the **XMLToDOM, DOMToQueue** functions).
2. To export/import GROUP structure a pair of function are provided:
    - **GroupToDOM** – export data from a GROUP structure to XML DOM Document object.
    - **DOMToGroup** – import data from a XML DOM Document object to GROUP structure

Comments:

All functions **xxxToDOM** return a pointer to a **Document** object. When the developer is done with the Document object it's necessary to free memory by calling the **Release** method. Typical code looks like this:

```
Doc   &Document, auto
  …
  Code
  Doc=GroupToDOM(GR)
  IF(NOT Doc& = NULL)
  …
  Doc.Release
  END
```

Overloaded Wrapper functions

The described set of functions provides necessary functionality to import and export XML data. However, to simplify work the following wrapper functions are provided:

- **ToXMLFile** – exports data of FILE, VIEW, QUEUE or GROUP structure to the given XML file (the implementation includes several version of overloaded function with a different parameter for each type of Clarion's structure (FILE, VIEW, QUEUE or GROUP)
- **FromXMLFile** – imports data of FILE, QUEUE or GROUP structure from the given XML file (this function is also overloaded).

***GetLastError** function*

To provide more detailed information about errors the special function **GetLastError is** provided. The function returns the last error code produced by the last XML function call.

See the error table for a complete list of possible errors.

Example

Let's consider the following XML files:

CUSTOMER.XML (Contains information about customers)

```
<?xml version="1.0"?>
<data>
	<row lastname="Wright" firstname="Carl" phone="222-1697"/>
	<row lastname="Curry" firstname="Charmaine" phone="333-9716"/>
</data>
```

NAMEMAP.XML (Contains information about name mapping)

```
<?xml version="1.0"?>
<data>
	<row ClarionName="LastName" XMLName="Name"/>
	<row ClarionName="Phone" XMLName="Tel"/>
</data>
```

Clarion code:

```
CC byte
Customer QUEUE                !queue to storing imported data
LastName  STRING(30)
Phone     STRING(30)
         END

NameMap QUEUE(MapQueue)       !queue with name mapping information
        END

  code
!Import data from file Customer.xml to Customer QUEUE.
!Destination QUEUE contains only LastName, Phone fields so data  only for these !fields is imported
  cc = FromXMLFile(Customer,'Customer.xml')
   if (CC)
   !Export data from Customer QUEUE to a file with name mapping

   ! Loading name mapping information from NAMEMAP.XML
    CC=FromXMLFile(NameMap,'NameMap.xml')
    IF(CC)
     CC = ToXMLFile(Customer,'CustomerData.xml',NameMap)
     IF(NOT CC)
      STOP('Exporting error')
     END
    ELSE
     STOP('NameMap.xml importing error')
    END
   ELSE
    STOP('Customer.xml importing error')
   END
```

The result file CUSTOMERDATA.XML is the following:

CUSTOMERDATA.XML

```
<?xml version="1.0"?>
<data>
       <row name="Wright" tel="222-1697"/>
       <row name="Curry"   tel="333-9716"/>
</data>
```

Template Support (optional)

Although the implemented functions are fairly easy to use it's possible to utilize the **Export To XML/Import From XML** template to further simplify use of those functions.

At design-time the developer provides the following information in the template's property dialog:

1. Type of operation (*Export* or *Import*).
2. The name of structure to be exported.
3. If "name mapping" should be used for the export/import operation.

- The generated code will call the File Open dialog to specify the name of an XML file for export/import operation, an optional dialog to specify XML file with name mapping information and the call to the corresponding function (**ToXMLFile** – for export or **FromXMLFile** – for import).

Built-in Caveats and Limitations

Here are the following limitations:

1. Import data from XML files to VIEW structure isn't supported because of a VIEW structure can't be written to.

2. The imported XML file format should be the same as format of file generated by exporting functions (now two types of format are supported - attribute-based and tag-based)

3. Error handling in the version has the following restrictions:
   - No error is produced in the case of type incompatibility (for example if a string data (with non numeric value) is assigned to numeric field on import)
   - If some field is missing in the structure that is used in import operation then data related to the field is just skipped. No error code returned.

4. Nested structures aren't supported. (for example, if a GROUP contains another GROUP as member)

5. Export/import structures with fields of ANY type isn't supported

6. Export/import structures with fields with DIM attribute (arrays) isn't supported

7. Data in FILE and QUEUE structure isn't removed before an import operation. So, imported data is just appended to the existing data.

8. If an error occurs during import operation then the state of structure (FILE, QUEUE or GROUP) is undetermined (i.e. structure may contain only partially imported data).

9. Export/import arrays of GROUPs (i.e. GROUP with DIM attribute) isn't supported.

10. Local linking of the XML support libraries is currently not supported.

List of XML Support Functions

This section contains the function list and a description of parameters that follows:

```
XMLStringToDOM      procedure(string xml), *Document
DOMToXMLString      procedure(*Document doc, unsigned Format = Format:AS_IS), string
XMLFileToDOM        procedure(string path), *Document
DOMToXMLFile        procedure(*Document doc, string path),byte
QueueToDOM          procedure(*queue que, <string label>,DOMStyle style=DOMStyle:ADO_26, |
                              byte removePrefix = true,<MapQueue map>), *Document
DOMToQueue          procedure(*Document doc, *queue q, <string label>, DOMStyle  |
                              style=DOMStyle:ADO_26,  <MapQueue map>),byte
FileToDOM           procedure(*file fl, <string label>, DOMStyle style=DOMStyle:ADO_26, |
                              byte removePrefix = true,<MapQueue map>), *Document
DOMToFile           procedure(*Document doc, *file fl, <string label>, |
                              DOMStyle style=DOMStyle:ADO_26, <MapQueue map>),byte
GroupToDOM          procedure(*group gr, <string label>, DOMStyle style=DOMStyle:ADO_26, |
                              byte removePrefix = true, <MapQueue map>), *Document
DOMToGroup          procedure(*Document doc, *group gr, <string label>, |
                              DOMStyle style=DOMStyle:ADO_26, <MapQueue map>),byte
ViewToDOM           procedure(*view vw, <string label>, DOMStyle style=DOMStyle:ADO_26, |
                          byte removePrefix = true, unsigned maxRowCount = 0, <MapQueue map>),|
                            *Document
ToXMLFile           procedure(*file fl, string path, <MapQueue map>,|
                              DOMStyle style=DOMStyle:ADO_26),byte
ToXMLFile           procedure(*queue que, string path, <MapQueue map>,|
                              DOMStyle style=DOMStyle:ADO_26),byte
ToXMLFile           procedure(*view vw, string path, <MapQueue map>,|
                              DOMStyle style=DOMStyle:ADO_26),byte
ToXMLFile           procedure(*group gr, string path,<MapQueue map>,|
                              DOMStyle style=DOMStyle:ADO_26),byte
FromXMLFile         procedure(*file fl, string path, <MapQueue map>,|
                              DOMStyle style=DOMStyle:ADO_26),byte
FromXMLFile         procedure(*queue que, string path, <MapQueue map>,|
                              DOMStyle style=DOMStyle:ADO_26),byte
FromXMLFile         procedure(*group gr, string path, <MapQueue map>,|
                              DOMStyle style=DOMStyle:ADO_26),byte

GetLastError        procedure, long
```

Parameter list for listed functions:

Xml	A string with XML content
Doc	A pointer to an XML DOM Document object
format	XML string format; can be: *Format:AS_IS* *Format:CANONICAL* *Format:REFORMATTED*
path	The valid path to an XML file
que	A pointer to a QUEUE structure to be imported/exported
fl	A pointer to a FILE structure to be imported/exported
gr	A pointer to a GROUP structure to be imported/exported
vw	A pointer to a VIEW structure to be exported
label	The row tag name in XML representation
style	The style of the generated XML file. Can be: **DOMStyle:ADO_26** (attribute-based style) or **DOMStyle:ADO_Net** (tag-based style)
map	The **MapQueue** structure to support name mapping
removePrefix	A Boolean value that indicates if a field's prefix should be removed or not
maxRowCount	The maximum number of rows to be processed

Function Error codes

All described functions (except **XMLStringToDOM** and **DOMToXMLString**) assign an error code to a global variable that can be accessed with **GetLastError** function.
The following error codes can be returned by **GetLastError** function:

Codes	Description
0	No errors occur
1	Error reading an XML file
2	Error writing to a XML file
3	Error: an attempt to use the same field name twice (duplicate name)
4	A name can be used as XML tag/attribute identifier
5	Error creating XML DOM Document object
6	EOF (End of File)
7	Invalid Argument
8	Illegal Function Call
9	Not Supported
10	Internal Error
11	Field Not Found
12	Common Error
13	Illegal Parameter
14	Import Fail
15	Not Successful
16	Illegal Format
17	Illegal Type
18	Illegal Size
19	No Such Field
20	EOS (End of Sequence)
21	Schema Not Supported
22	Inconsistent Schema
23	Check Fail
24	Add Record Failed
25	Invalid Node Name
26	No Current Node
27	Cdata Not Supported
28	Base64 Not Supported