Virtual XML 1.0

Doc Rev 1.01

VIRTUAL XML DATA STORAGE

1. **Basics**

VXML (Virtual XML) is XML-based database built on Virtual Storage platform as VStorage wrapper.

It inherits all existing Virtual Storage capabilities, such as distributed allocation of the database files, multi-space approach for specific data pools, dump/restore functions, static and dynamic storage extension, transactions, physical data consistency, etc.

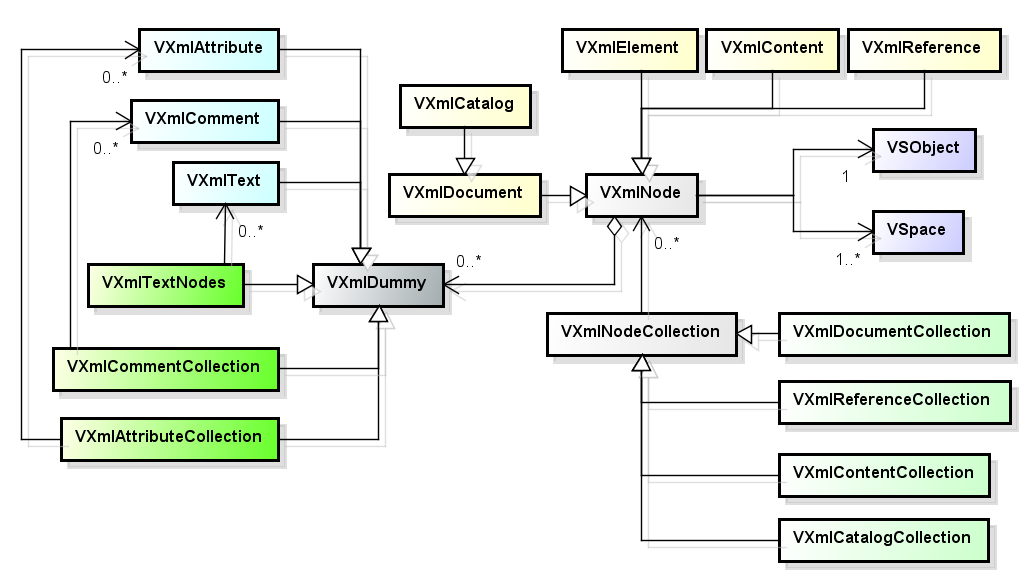
VXML database is NOT a single XML document. It provides the embedded catalog structure that can be managed by user in the specific manner, XML documents are attached to catalog nodes. XQL queries may retrieve XML nodes from the specific document/element or documents list from the catalog structure using internal documents nodes as well as document names in the queries.

So, there are some specific extensions in VXML:

* New node type ‘Catalog’ that allows build the catalog tree.
* New node type ‘Reference’ that allows create reference to the ‘Catalog’ or ‘Document’ objects. References could be used instead of cloning real objects.
* **New node type 'Content' that allows create binary content attached to the document element. Binary content could be stored in the separate Virtual Storage space for easy management. Anyway the data consistency will be guaranteed by the Virtual Storage cross-space transactions.**
* Extended XQL language to search objects within catalog tree as well as within particular document; multi-level search conditions could be defined.
* Data portability. Any document or document element (including all subtree) could be exported using CheckOut method to the file in the internal portable format with assigning unique GUID identifier. This file can be checked in to another database and the source data will become read-only. This capability allows safe data exchange between two or more VXML databases.

VXML also provides its own UI administration tool (VXmlExplorer) that allows manage data spaces for VXML storage as well as individual objects within the database (create, edit, delete).

1. **Architecture and Design**
2. **VXML Object model**



* 1. **Nodes**

Virtual XML includes the root class (VXmlNode) that encapsulates most of the node properties and methods and set of the descendant classes representing the specific of the particular node types.

**VXmlNode** – root node class that contains the core VXML node properties and methods.

**VXmlElement** – class that represents node type ‘element’. Inherited from **VXmlNode**.

**VXmlDummy** – base class that represents internal node types.

**VXmlAttribute** – class that represents node type ‘attribute’. Inherited from **VXmlDummy**.

**VXmlComment** – class that represents node type ‘comment’. Inherited from **VXmDummy**.

**VXmlText** – class that represents node type ‘text’. Inherited from **VXmlDummy**.

**VXmlContent** – class that represents node type ‘content’. Inherited from **VXmlNode**.

**VXmlReference** – class that represents node type ‘element’. Inherited from **VXmlNode**.

**VXmlDocument** – class that represents node type ‘document’ for XML document. Inherited from **VXmlNode**.

**VXmlCatalog** – class that represents node type ‘catalog’ for XML document container. Inherited from **VXmlDocument**.

* 1. **Collections**
     1. **Nodes**

Virtual XML includes the root class **VXmlNodeCollection** that encapsulates most of the properties and methods for node collections and set of the descendant classes representing the specific of the particular node types.

The number of nodes in the collection can be obtained using ‘Count’ property.

Each node in the collection can be referenced by index (0 … Count-1) or node name (case sensitive).

Collection is returned by XQL query (SelectNodes method) or by node properties:

|  |  |
| --- | --- |
| **Node property** | **Collection class** |
| **VXmlNode**.ChildNodes | VXmlNodeCollection |
| **VXmlNode**.ContentNodes | VXmlContentCollection |
| **VXmlCatalog**.ChildNodes | VXmlCatalogCollection |
| **VXmlCatalog**.Documents | VXmlDocumentCollection |
| **VXmlCatalog**.References | VXmlReferenceCollection |

* + 1. **Dummy nodes**

Dummy nodes are not separate objects within storage. They are inherited from **VXmlDummy** class.

Collections are also inherited from VXmlDummy:

|  |  |
| --- | --- |
| **Node property** | **Collection class** |
| **VXmlNode**.Attributes | VXmlAttributeCollection |
| **VXmlNode**.CommentNodes | VXmlCommentCollection |
| **VXmlNode**.TextNodes | VXmlTextCollection |

1. **Virtual Storage integration**

To open VXML storage you must create root ‘catalog’ node object and specify the root Virtual Storage folder as parameter, e.g.:

catalog = new VXmlCatalog(“c:\\data\\my\_xml\_storage”);

VXML uses three Virtual Storage spaces:

**vxmlbase** - contains all VXML nodes data and (optionally) binary content;

**vxmlcontent** - contains binary content of the ‘content’ nodes; if this space is missing then all information is stored in the **vxmlbase** space.

**vxmlindex** - contains supplementary indexes.

Space(s) must exist at this point. You can create VXML spaces in VXML administration tool UI (VXmlExplorer), Virtual Storage administration tool (command-line: VSUtil, UI: VStorageExplorer) or directly using Virtual Storage API (‘Create’ method).

The root catalog object initializes Virtual Storage spaces.

Each node object in VXML is attached to the corresponding ‘VStorage.VSObject’ and uses VSObject space allocation to store all node information and reference. The space is initially allocated when node is created and released when node is removed.

VXML supports VStorage transactions (catalog methods ‘Begin’, ‘Commit’ and ‘RollBack’).

When VXML storage is opened the transaction is started by default.

1. **Extended XQL**

VXQL supports XQL queries in ‘SelectNodes’ and ‘SelectSingleNode’ methods.

The only difference between these methods is that ‘SelectSingleNode’ will always select only one (first) node even if more than one node matches search criteria.

The query result depends on current node type:

* Node type ‘**catalog**’

The result depends on the specified XQL prefix (in the beginning of the XQL expression):

* ‘**~**’ - the search will return reference nodes rather than document and catalog nodes (if occurs); otherwise referencing node (document or catalog) will be returned.
* ‘**$**’ - the search will return *only catalog nodes*; documents are out of search criteria scope.
* ‘**#**’ - the search will return *only document nodes*; the document elements will also be analyzed for matching criteria if specified.
* Otherwise - the search will return *only document nodes*; *only document* names and attributes will be analyzed for matching criteria; the document elements will be ignored.
* Node type ‘**document**’ or ‘**element**’

Returns one or more nodes of ‘element’ type matching search criteria if the specific node type is not defined.

Returns the node collection of the specified type if requested (‘text’, ‘comment’, ‘content’) matching the search criteria for ‘element’ nodes.

* Other node types: *XQL is not supported*.
  1. **vXQL syntax**

<prefix>[<scope>]<context><[predicate 1][operation][predicate 2]…[operand][predicate n]…>

* + 1. **Prefix**

Prefix defines search scope (for ‘catalog’ nodes only, otherwise shall be empty)

[**~**] [**$/#**]

* ‘**~**’ - the search will return *only catalog nodes*; documents are out of search criteria scope.
* ‘**$**’ - the search will return *only catalog nodes*; documents are out of search criteria scope.
* ‘**#**’ - the search will return *only document nodes*; the document elements will also be analyzed for matching criteria if specified.
* **No prefix** - the search will return *only document nodes*; *only document* names and attributes will be analyzed for matching criteria; the document elements will be ignored
  + 1. **Context**

Contextexpression defines the scope for query. Context may contain special directives as well as node name pattern:

**///** - search recursively all tree starting from the root node (‘element’ or ‘catalog’);

**//** - search recursively all tree starting from the childs of the current node;

**.//** - search recursively all subtree starting from the current node;

**..** - search subtree starting from the parent node;

**.** - search subtree starting from the current node;

**/** - search subtree starting from the root node;

**first()** - select first node;

**last()** - select last node;

**content()** - select ‘content’ nodes;

***name*** - node name pattern to select (may contain ‘\*’ and ‘?’ characters);

* + 1. **Predicate**

Predicate defines filtering criteria in brackets: ‘[‘ and ‘]’ or ‘{‘ and ‘}’ (they are equal and could be used in multi-level predicates for better visibility.

Predicates could be multi-level. Predicates on the same level are connected by Boolean operation ‘&’(*and*) or ‘|’(*or*). If operand is missing, then assuming ‘&’.

Predicate can be one of the following:

1. **Node name**

<name>

Predicate is true if *child* node with the specified name exists.

Node name can be a pattern including ‘\*’ and ‘?’.

Example:

[‘name’]

{‘family\*’}

[‘a?dr\*’]

[‘\*’]

1. **Node number**

<number>

Predicate is true only if node with the specified number exists in the child nodes list for XPath.

The numbering is starting from 1.

*Note: the number applies to the initial node list, not nodes in the query result.*

Example:

[5]

[19]

1. **Node tag(s)**

[#tag1{#tag2}…{#tagN}]

Predicate is true if at least one node tag exists. One or more tags separated by ‘#’ could be specified.

Tag values could be a pattern including ‘?’ and ‘\*’ characters and are *not* case sensitive.

Example:

[#mytag]

[#t?1#name\*]

1. **Attribute name**

@<name>

Predicate is true if node attribute with the specified name exists.

Attribute name can be a pattern including ‘\*’ and ‘?’.

1. **Compare child node value**

The node value is always treated as string.

<name><op><value>

Where:

<name> - child node name;

<op> - comparison operation, one of:

**=** equal, case sensitive;

**!=** not equal, case sensitive;

**=$** equal, not case sensitive;

**!=$** not equal, not case sensitive;

<value> - value for comparison, wildcards ‘\*’ and ‘?’ are allowed.

Example:

Node1=’\*xt’

mynode !=$ ‘firtst???’

1. **Compare attribute value**

The attribute value can be either string or numeric.

The string value must be quoted.

The numeric value must not be quoted and not contain alpha characters, otherwise error will be considered.

@<name><op><value>

Where:

<name> - attribute name;

<op> - comparison operation, one of:

**=** equal, numeric or string case sensitive;

**!=** not equal, numeric or string case sensitive;

**=$** equal, string not case sensitive;

!**=$** not equal, string not case sensitive;

**>** more than (numeric only);

**<** less than (numeric only);

**>=** equal or more than (numeric only);

**<=** equal, or less than (numeric only);

<value> - value for comparison, wildcards ‘\*’ and ‘?’ are allowed for string values.

Example:

atr1= 55

atr2 <= 17

myattr !=$ ‘chapter ??’

name =$ ‘?nic\*’

* 1. **vXQL sample**

|  |  |
| --- | --- |
| node0005 | Select child node ‘node0005’. |
| node000\* | Select all child nodes with the name starting with ‘node000’. |
| node???5 | Select all child nodes with the name starting with ‘node’ and ending with ’5’ and 3 any chars between. |
| \* | Select all child element nodes. |
| //\* | Select all element nodes in the document. |
| .. | Select parent element node. |
| [1] | Select first child node of the current node. |
| [5] | Select fifth child node of the current node. |
| C\_node[5] | Select fifth child node of the current node if the name is ‘C\_node’. |
| C\_\*[5] | Select fifth child node of the current node if the name starting with ‘C\_\*’. |
| L\*[first()] | Select first child node with the name starting with ’L’. |
| \*[last()] | Select last child node. |
| \*/\*05 | Select all grand-child nodes with the name ending with ‘05’. |
| \*/\*05[1] | Select first grand-child node if its name is ending with ‘05’. |
| \*/\*05[2] | Select second grand-child node if its name is ending with ‘05’. |
| /\*[5]/\*[3] | Select 3rd child of the 5th child of the root element node (grandchild of the root node). |
| v\_0005//\* | Select all descendant nodes of the child node with the ‘v\_0005’ name. |
| v\_0005//\*[5] | Select 5th descendant node of the child node with the ‘v\_0005’ name (look through all levels). |
| //v03\_0005 | Select all nodes in the document with ‘v03\_0005’ name. |
| //v01\_0005/v02\_0003 | Select all nodes with ‘v02\_0003’ which are child of ‘v01\_0005’ nodes through all document. |
| ./ | Select current node. |
| . | Select current node. |
| \*5[@atr01='A01\_06'] | Select all child nodes with the name ending with ’5’ if node has attribute ‘atr01’ with the value ‘A01\_06’. |
| \*5[@atr01='A01\*7'] | Select all child nodes with the name ending with ’5’ if node has attribute ‘atr01’ which value matches pattern ‘A01\*7’. |
| \*[5][@atr07] | Select 5th child node if it has attribute with the name ‘atr07’. |
| \*[5][@atr01='A01\*'] | Select 5th child node if it has attribute ‘atr01’ and its value matches case-sensitive pattern ‘A01\*’. |
| \*[5][@atr01=$'a01\*'] | Select 5th child node if it has attribute ‘atr01’ and its value matches non-case-sensitive pattern ‘a01\*’. |
| l01\_0005[l02\_0003] | Select child node named ‘l01\_0005’ if it has child node (grandchild of the current) named ‘l02\_0003’. |
| \*[1] | [5] | Select 1st and 5th child nodes. |
| \*[[1]|[5]] & [ [@atr5='A01\_0005'] & [@atr3=’A01\_0003'] ] | Select 1st or 5th child nodes if they have both ‘atr3’ and ‘atr5’ attributes with the specified values. |
| \*{ { [1] | [5] } | [ [@atr1=50] & [node2 = 99] } | Select 1st or 5th child nodes or any child node that has ‘atr1’ attribute with the numeric value ‘50’ or child element node (grandchild of the current) with the numeric value ‘99’. |

1. **Data portability**

Each VXML document or element within the document cold be exported (‘checked out’) to the portable file format that can be imported (‘checked in’) to the same or another VXML database using ‘CheckOut’ method.

* 1. **Check Out node**

The ‘CheckOut’ method will perform the following actions:

1. Assign GUID to the current node. If node already has GUID assigned then existing value will be used.
2. Create portable file containing the full tree of the node. The file name is:

<GUID>.vsnp

Where:

<GUID> - GUID value assigned to this node;

vsnp – file extension

1. Make the current node and all subtree read-only
   1. **Export node**

You can also use ‘Snap’ method to create portable file.

It is similar to ‘CheckOut’ method but remains the node and all subtree writable.

* 1. **Undo Check Out node**

‘UndoCheckOut’ method allows make the node and all subtree writable after checking out. But it retains the GUID value of node unchanged.

This method is useful if you need to cancel ‘CheckOut’ operation for node.

* 1. **Check In node**

You can check in portable file into the VXML database. The following actions will apply:

1. If GUID from the portable file already exists in the VXML database then node associated with this GUID will be used as place for checking in, no matter which node is current.
2. If GUID doesn’t exist in the VXML database then it will be assigned to node checked in.
3. If GUID exists then this node will be replaced by the portable file content otherwise new child node and subtree will be created as child of the current node.
4. The node will become writeable (if it was previously checked out).
5. **API**

The **VXML.dll** is virtual XML storage component which is based on Virtual Storage platform.

It provides such capabilities as:

* Manage catalog structure for XML documents;
* Manage XML documents and nodes;
* Support XQL queries for catalog nodes and documents;
* Data portability between multiple VXML storages;
* Support transactions via Virtual Storage transaction mechanism;
* Binary content management.

Programming language – C#

Framework - .NET Framework 4.5

Dependencies – Vstorage.dll

1. **Public Constants and Business Rules**
   1. public static class DEFX

VXML node types numeric values of the ‘VXmlNode.NodeTypeCode’ field. The corresponding string representation can be obtained using ‘VXmlNode.NodeType’ field.

|  |  |
| --- | --- |
| **Constant** | **Description** |
| public const short NODE\_TYPE\_UNDEFINED = 0 | Undefined |
| public const short NODE\_TYPE\_CATALOG = 1 | ‘catalog’ node type |
| public const short NODE\_TYPE\_DOCUMENT = 2 | ‘document’ node type |
| public const short NODE\_TYPE\_ELEMENT = 3 | ‘element’ node type |
| public const short NODE\_TYPE\_CONTENT = 4 | ‘content’ node type |
| public const short NODE\_TYPE\_REFERENCE = 5 | ‘reference’ node type |
| public const short NODE\_TYPE\_ATTRIBUTE = 100 | ‘attribute’ node type |
| public const short NODE\_TYPE\_COMMENT = 101 | ‘comment’ node type |
| public const short NODE\_TYPE\_TEXT = 102 | ‘text’ node type |
| public const short NODE\_TYPE\_TAG = 103 | ‘tag’ node type |

Other constants:

|  |  |
| --- | --- |
| **Constant** | **Description** |
| public static string GET\_NODETYPE(short type) | Returns string representation of the node type code. |
| public const string KEY\_LOAD\_XML = "loadxml-path.vs.default" | VStorage key for load xml directory |
| public const string KEY\_SNAP = "snap-path.vs.default" | VStorage key for checkout/snap directory. |
| public const string PREFIX\_ATTRIBUTE="%" | VSObject field prefix for attribute name |
| public const string PREFIX\_COMMENT="^" | VSObject field prefix for comment node name |
| public const string PREFIX\_TEXT="&" | VSObject field prefix for text node name |
| public const string NON\_START\_PATTERN | String containing additional characters allowed in the node name |
| public const string START\_PATTERN | String containing characters allowed in the node name 1st symbol |
| public const string NON\_START\_PATTERN | String containing additional characters allowed in the node name |
| public const string XML\_CONTENT\_SPACE\_NAME="vxmlcont" | VStorage space name for binary content |
| public const string XML\_INDEX\_SPACE\_NAME="vxmlindx" | VStorage space name for indexing |
| public const string XML\_SNAP\_FILE\_TYPE = "vsnp" | Portable file type (extension) for Snap/CheckOut |
| public const string XML\_SPACE\_NAME="vxmlbase" | VStorage space name for XML data |

Business rules:

|  |  |
| --- | --- |
| **Constant** | **Description** |
| public static bool BR\_CAN\_HAVE\_REFERENCE(short type) | Returns true if node type can have reference. |
| public static bool BR\_CHECKIN\_IS\_VALID\_TYPE(short parent\_type) | Returns true if ‘CheckIn’ is allowed for node type. |
| public static bool BR\_CHECKOUT\_IS\_VALID\_TYPE(short parent\_type) | Returns true if ‘CheckOut’ or ’Snap’ is allowed for node type. |
| public static bool BR\_CHILD\_IS\_VALID\_TYPE(short parent\_type, short type) | Returns true if node of type ‘parent\_type’ can create child node of type ’type’. |
| public static short[] BR\_CHILD\_VALID\_TYPE\_CODES(short type) | Returns array of the valid types for child node. |
| public static string[] BR\_CREATE\_VALID\_TYPES(short type) | Returns string array of node types that can be created as child nodes for node type. |
| public static bool BR\_INSERT\_IS\_VALID\_TYPE(short target\_type, short type) | Returns true if node of type ‘type’ can be inserted before the node of type’target\_type’. |
| public static bool BR\_NODE\_NAME\_VALID(string name) | Returns true if specified string is valid for node name. |
| public static bool BR\_NODE\_NEED\_NAME(short type) | Returns true if node name is user-defined. |
| public static bool BR\_NODE\_REFERENCE(short type) | Returns true if reference can be created for node type. |
| public static bool BR\_NODE\_RENAME(short type) | Returns true if node of this type can be renamed. |
| public static bool BR\_XML\_IS\_VALID\_TYPE(short parent\_type) | Returns true if load or save XML is allowed for node type. |

1. **Classes**
   1. public class VXmlNode

Root VXML node class.

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public VXmlNodeCollection AllChildNodes | The collection of all child nodes of all types, including ‘attribute’ nodes. |
| public VXmlAttributeCollection Attributes | The collection of child ‘attribute’ nodes. |
| public bool CheckedOut | True if current node is checked out. |
| public bool CheckedOutTree | True if current node or any node in the subtree is checked out. |
| public VXmlNodeCollection ChildNodes | The collection of child ‘element’ nodes. |
| public VXmlCommentCollection CommentNodes | The collection of child ‘comment’ nodes. |
| public VXmlContentCollection ContentNodes | The collection of child ‘content’ nodes. |
| public VSpace ContentSpace | Vrtual Strorage space for binary content. |
| public VXmlNode FirstChild | First ‘element’ node. |
| public string GUID | CheckOut/Snap GUID of the current node; empty if node has never been checked out. |
| public bool HasAttributes | True if node has at least one attribute. |
| public bool HasChildNodes | True if node has at least one child node (except attribute). |
| public long Id | Unique node Id within storage numeric representation. |
| public long ID | Unique node Id within storage string representation. |
| public VXmlNode LastChild | Last ‘element’ node. |
| public string Name | Node name. |
| public VXmlNode Next | Next node of any type. |
| public VXmlNode NextSibling | Next ‘element’ node. |
| public VSpace NodeSpace | Virtual Storage space for XML data. |
| public string NodeType | Node type string representation. |
| public short NodeTypeCode | Node type numeric representation. |
| public VXmlNode Owner | Owner node for current node (‘document’ or ‘catalog’). |
| public VXmlDocument OwnerDocument | Owner document (for all node types except ‘document’ or ‘catalog’). |
| public VXmlNode ParentNode | Parent node of the current node. |
| public VXmlNode Previous | Previous node of any type. |
| public VXmlNode PreviousSibling | Previous ‘element’ node. |
| public VXmlTagCollection Tags | The collection of child ‘tag’ nodes. |
| public string Text | Concatenation of all text nodes values. |
| public VXmlTextCollection TextNodes | The collection of child ‘text’ nodes. |
| public string Value | Node value. |
| public string Xml | String representation of the node and subtree. |

* + 1. Constructors

No public constructors.

* + 1. Methods

|  |  |  |
| --- | --- | --- |
| **Name** | **Parameter(s)** | **Description** |
| public VXmlNode AppendChild(VXmlNode node) | **node** – node to append | Append existing node as child. |
| public VXmlNode CheckIn(string path) | **path** – full path to ‘vsnp’ file | Check in XML node from the portable file format. |
| public string CheckOut(string path) | **path** – full path to directory where file shall be created (fine name must not be included) | Check out node (export to the portable format file). |
| public VXmlNode Clone() |  | Create clone of the node including all subtree. |
| public VXmlNode CloneNode(bool deep) | **deep** – ‘true’ to clone full tree; otherwise false (default) | Create clone of the node itself of all tree depending on parameter. |
| public VXmlAttribute CreateAttribute(string name, string value) | **name** – attribute name  **value** – attribute value | Create new attribute node. |
| public VXmlAttribute CreateAttributeAt(string xpath, string name, string value) | **xpath** – node location  **name** – attribute name  **value** – attribute value | Create new attribute node at the specified XPath. |
| public VXmlComment CreateComment(string text) | **text** – comment text | Create new comment node. |
| public VXmlComment CreateCommentAt(string xpath, string text) | **xpath** – node location  **text** – comment text | Create new comment node at the specified XPath. |
| public VXmlContent CreateContent(string title, string filename) | **title** – content title  **filename** – full path to the binary content file | Create new content node. |
| public VXmlContent CreateContentAt(string xpath, string title, string filename) | **xpath** – node location  **title** – content title  **filename** – full path to the binary content file | Create new content node at the specified XPath. |
| public VXmlElement CreateElement(string name, string value) | **name** – node name  **value** – node value | Create new element node. |
| public VXmlElement CreateElementAt(string xpath, string name, string value) | **xpath** – node location  **name** – node name  **value** – node value | Create new element node at the specified XPath. |
| public VXmlTag CreateTag(string text) | **text** – tag value. I tag with the specified value already exists, the new node will not be created, existing node will be returned. | Create new tag node. |
| public VXmlText CreateTextNode(string text) | **text** – text value | Create new text node. |
| public VXmlText CreateTextNodeAt(string xpath, string text) | **xpath** – node location  **text** – text value | Create new text node at the specified XPath. |
| public string GetAttribute(string name) | **name** – attribute name | Get attribute value. |
| public string GetAttributeAt(string xpath, string name) | **xpath** – node location  **name** – attribute name | Get attribute value at the specified XPath. |
| public VXmlAttribute GetAttributeNode(string name) | **name** – attribute name | Get attribute node. |
| public VXmlElement GetChildElement(string name) | **name** – node name | Get child element node by name. |
| public VXmlComment GetCommentNode(string name) | **name** – node name | Get comment node by name node. |
| public VXmlNode GetFirstNode(short type) | **type** – node type | Get first child node of the specified type. |
| public VXmlNode GetLastNode(short type) | **type** – node type | Get last child node of the specified type. |
| public VXmlNode GetNode(long id) | **id** – node unique id in the storage | Get node by unique Id. This node doesn’t have be a child node. |
| public VXmlText GetTagNode(string name) | **name** – node name | Get tag node by node name. |
| public VXmlText GetTextNode(string name) | **name** – node name | Get text node by node name. |
| public VXmlNode InsertAfter(VXmlNode node, VXmlNode after) | **node** – node to insert  **after** – child node to insert after | Insert node after the specified child node. |
| public VXmlNode InsertBefore(VXmlNode node, VXmlNode before) | **node** – node to insert  **before** – child node to insert before | Insert node before the specified child node. |
| public string Load(string file, VXmlNode parent = null) | **file** – full path to the XML file  **parent** – node to which new node will be added as child. If null then current node will be parent. | Create XML node(s) from the XML representation in text file. All tree will be created if child nodes present. |
| public string LoadXml(string xmlstring, string name, VXmlNode parent = null) | **xmlstring** – XML string representation  **name** – name for cache; using the name is useful to improve performance if many nodes with the same name and structure created. If name matches cache then previously parsed result will be used.  **parent** – node to which new node will be added as child. If null then current node will be parent. | Create XML node(s) from the XML string representation. All tree will be created if child nodes present. |
| public VXmlNode PrependChild(VXmlNode node) | **node** – node to prepend | Similar to ‘AppendChild’ but node will be added as first child. |
| public void Remove() |  | Remove the current node and all subtree. |
| public void RemoveAll() |  | Remove all child nodes and attributes of the current node. |
| public void RemoveAllAt(string xpath) | **xpath** – node location | Remove all child nodes and attributes at the specified XPath. |
| public void RemoveAllAttributes() |  | Remove all attributes of the current node. |
| public void RemoveAttribute(string name) | **name** – attribute name | Remove attribute of the current node by name. |
| public void RemoveChild(VXmlNode node) | **node** – child node object | Remove child node. |
| public void RemoveNodeAt(string xpath) | **xpath** – node location | Remove node at specified XPath. |
| public void RemoveText(string name) | **name** – node name | Remove text node by name. |
| public void RemoveTag(string tag) | **tag** – tag value | Remove tag node by value. |
| public void RemoveTagByName(string name) | **name** – node name | Remove tag node by name. |
| public void ReplaceChild(VXmlNode newChild, VXmlNode oldChild) | **oldchild** – node to be replaced  **newchild** – node for replacement | Replace child node by another node. |
| public string SaveXml(string fname, bool content = true) | **fname** – full path to the XML file  **content** – true: save binary content | Save node and subtree to file in XML format. |
| public VXmlAttributeCollection SelectAttributes(string xpath, string name) | **xpath** – node location  **name** – attribute name or pattern. All attributes by default. | Select node attributes using XPath expression for node. If more than one node matches XPATH expression then 1st node in the search result will be used. |
| public VXmlNodeCollection SelectNodes(string xpath) | **xpath** – node search expression | Select one or more nodes using XPath expression. |
| public VXmlNode SelectSingleNode(string xpath) | **xpath** – node search expression | Select one or more nodes using XPath expression. If more than one node is in the query result then only first will be returned. |
| public void SetAttribute(string name, string value) | **name** – attribute name  **value** – attribute value | Create node attribute or update value of the existing attribute. |
| public void SetAttributeAt(string xpath, string name, string value) | **xpath** – node location  **name** – attribute name  **value** – attribute value | Create node attribute or update value of the existing attribute attributes at the specified XPath. |
| public string Snap(string path = "") | **path** – full path to directory where file shall be created (fine name must not be included) | Similar to ‘CheckOut’ but doesn’t lock node(s). |
| public bool UndoCheckOut() |  | Turns off read-only mode for checked out tree. |

* 1. public class VXmlContent : VXmlNode

Binary content management (node type ‘content’).

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public byte[] ContentBytes | Binary content representation as byte array. |
| public string ContentString | Binary content representation as byte string. |
| public string filename | The file name if content is uploaded from file. |
| public string fileref | The file reference (URL) if content has been saved (generated by SaveXml). |
| public long Length | Content length (bytes). |
| public string path | The file path if content is uploaded from file. |
| public long size | The file size in bytes if content is uploaded from file. |
| public string title | Content title. |

* + 1. Constructors

No public constructors.

* + 1. Methods

|  |  |  |
| --- | --- | --- |
| **Name** | **Parameter(s)** | **Description** |
| public void Download(string path) | **filename** – full path and file name to save. If empty then saved file name will be used. If saved name is empty then content will be saved to the current directory with ‘NONAME.content’ name. | Download binary content to file. |
| public void Upload(string file, bool setattr) | **filename** – full path to the file to upload  **setattr** – if true (default) fields ‘filename’ and ‘path’ will be set | Upload binary content from file. |

* 1. public class VXmlReference : VXmlNode

Node reference (node type ‘reference’).

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public long ReferenceId | Unique Id of the original node. |
| public VXmlNode ReferenceNode | Original node object. |

* + 1. Constructors

No public constructors.

* + 1. Methods

N/A

* 1. public class VXmlDocument : VXmlNode

VXML document (node type ‘document’).

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public VXmlNode DocumentElement | Root element of the document. |

* + 1. Constructors

No public constructors.

* + 1. Methods

|  |  |  |
| --- | --- | --- |
| **Name** | **Parameter(s)** | **Description** |
| public VXmlDocument Clone(string new\_name) | **new\_name** – name for new document | Create clone of the document. New name will be assigned to new document. |
| public string Load(string file, VXmlNode parent) | **file** – full path to the XML file  **parent** – node to which new node will be added as child. If null then current node will be parent. | Create XML node(s) from the XML representation in text file. All tree will be created if child nodes present. |
| public string LoadXml(string xmlstring, string name, VXmlNode parent) | **xmlstring** – XML string representation  **name** – name for cache; using the name is useful to improve performance if many nodes with the same name and structure created. If name matches cache then previously parsed result will be used.  **parent** – node to which new node will be added as child. If null then current node will be parent. | Create XML node(s) from the XML string representation. All tree will be created if child nodes present. |

* 1. public class VXmlCatalog : VXmlDocument

VXML catalog node (node type ‘catalog’).

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public VXmlCatalogCollection ChildNodes | List of child catalog nodes. |
| public VXmlDocumentCollection Documents | List of document nodes for catalog node. |
| public bool HasChildNodes | True if node has at least one child catalog node (or reference to catalog node). |
| public VXmlCatalog OwnerCatalog | Owner catalog node. |
| public VXmlReferenceCollection References | List of reference nodes for catalog node. |
| public VSEngine Storage | Virtual Storage VSEngine object. |
| public title Storage | Catalog node title |

* + 1. Constructors

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| public VXmlCatalog() | This constructor can be used if you need to initialize the ne new storage (‘InitializeStorage’ method). |
| public VXmlCatalog(string root\_path) | Create root catalog object only. ‘**root\_path**’ parameter specifies path to Virtual Storage root catalog. If storage doesn’t exist then it will be created with the default parameters (XML space – 5 Mb, XML space auto extension – 5 Mb; content space – 5 Mb, content space auto extension – 15 Mb). |

* + 1. Methods

|  |  |  |
| --- | --- | --- |
| **Name** | **Parameter(s)** | **Description** |
| public void Begin() |  | Begin transaction. |
| public void Close() |  |  |
| public void Commit() |  | Commit transaction. |
| public VXmlCatalog CreateCatalog(string name, string value) | **name** – catalog node name  **value** – catalog name value (optional) | Create new ‘catalog’ node. |
| public VXmlDocument CreateDocument(string name) | **document** – catalog node name | Create new document. |
| public VXmlReference CreateReference(VXmlNode n) | **n** – node for reference | Create new node reference. |
| public VXmlCatalog GetChildCatalog(string name) | **name** – catalog node name | Get child catalog node by name. |
| public VXmlDocument GetChildDocument(string name) | **name** – document name | Get child document node by name. |
| public void InitializeStorage(string root\_path, int size, int ext = 0, int content\_size = 0, int content\_ext = 0) | **root\_path** – root storage directory in the file system (must exist)  **size** – XML space initial size (Mb)  **ext** – XML space auto extension (Mb), default - 0  **content\_size** – content space initial size (Mb), default - 0  **content\_ext** – content space auto extension (Mb), default - 0 | Create and initialize new storage. |
| public void RemoveAllReferences() |  | Remove all references for this catalog node. |
| public void RemoveCatalog(VXmlCatalog cat) | **cat** – catalog node | Remove child catalog node. |
| public void RemoveDocument(VXmlDocument doc) | **doc** – document | Remove child document. |
| public void RemoveReference(VXmlReference r) | **r** – reference node | Remove child reference node. |
| public void RollBack() |  | Rollback transaction. |

* 1. public class VXmlException : Exception

Inherited from: Exception

* + 1. Constructors

|  |  |  |
| --- | --- | --- |
| **Constructor** | **Parameter(s)** | **Description** |
| public VXmlException(int code, string message\_ext) | **code** – error code  **message\_ext** – message to append to the default system message (empty by default). | VXML exception object. Any exception thrown has its own code and message. |

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public int ErrorCode | Error code for exception |
| public string Message | Full error message for exception |

* + 1. Error codes and messages

|  |  |  |
| --- | --- | --- |
| **Constant** | **Message** |  |
| public int E0001\_CONTENT\_FILE\_NOT\_FOUND\_CODE = 1 | Content file not found | The specified file is not found in the ‘CreataContent’ method |
| public int E0002\_NODE\_IS\_UP\_TREE\_CODE = 2 | Node is up in the tree | Node for append/insert/replace is parent for current node |
| public int E0003\_NODE\_IS\_FROM\_DIFFERENT\_DOC\_CODE = 3 | Node is up from different document | Node for append/insert/replace/remove is not in the current document |
| public int E0004\_INVALID\_NODE\_TYPE\_CODE = 4 | Invalid node type | Node type is not applicable in the current operation |
| public int E0005\_NOT\_A\_CHILD\_NODE\_CODE = 5 | Not a child node | The specified node is not a child node in replace/remove/insert operation |
| public int E0006\_CATALOG\_INVALID\_OP\_CODE = 6 | Invalid operation, applicable for root catalog element only | Transaction management methods or ‘Close’ method is invoked not by the root catalog node |
| public int E0007\_OLD\_EQUAL\_NEW\_CODE = 7 | Operands are the same node | Node in insert/replace operations are the same |
| public int E0008\_INVALID\_CHAR\_CODE = 8 | Invalid character in the node name | Invalid character in the node name |
| public int E0009\_INVALID\_TYPE\_CODE = 9 | Invalid node type | Node type is invalid for operation (create node, check out) |
| public int E0010\_ROOT\_EXISTS\_CODE = 10 | CreateNode: root node already exists | Attempt to create root element for document when it already exists |
| public int E0011\_XML\_CREATE\_INVALID\_TYPE\_CODE = 11 | Create from template - node type is not element | The nNode type initiated ‘Load’ or ‘LoadXml’ method is not ‘element’ |
| public int E0012\_PATH\_NOT\_FOUND\_CODE = 12 | Path is not found | File path is not found in CheckOut operation |
| public int E0013\_ALREADY\_CHECKED\_OUT\_CODE = 13 | "Current node, parent or child node is checked out | Attempt of any operation affected checked out node |
| public int E0014\_UNSUPPORTED\_CHECKOUT\_VERSION\_CODE = 14 | Unsupported CheckOut format version | ‘CheckIn’ found that portable file format is not compatible with the current VXML version |
| public int E0015\_CHECKIN\_ERROR\_CODE = 15 | CheckIn error: | Error during ‘CheckIn’, the particular details are provided in the message |
| public int E0016\_XQL\_ERROR\_CODE = 16 | XQL error: | XQL format error in ‘SelectNodes’ or ‘SelectSingleNode’, the particular details are provided in the message |
| public int E0017\_DOC\_EXISTS\_CODE = 17 | Node with the specified name already exists | Duplicate name when trying to rename node attribute |
| public int E0018\_DOC\_NAME\_MISSING\_CODE = 18 | Document name is missing | New document name is not specified when cheating document clone |
| public int E0019\_SPACE\_MISSING\_CODE = 19 | VXML Node space is not found | Virtual Storage space for xml data is not found when initiating root catalog object |
| public int E0020\_XML\_PARSE\_ERROR\_CODE = 20 | VXml Parser - parse error | Unexpected XML parsing error in ‘Load’ or ‘LoadXml’ methods |
| public int E0021\_INVALID\_NODE\_FIELD\_CODE = 21 | Invalid node field (internal error) | Possible reason: physical storage structure is damaged |
| public int E0022\_NOT\_VSXML\_NODE\_SPACE\_CODE = 22 | Space owner is not VSXML or undefined | The specified VXML storage is actually designed to another application or damaged |
| public int E0023\_NOT\_EMPTY\_UNDEFINED\_SPACE\_CODE = 23 | Space owner is undefined but space is not empty | The specified VXML storage is ready to be initialized for VXML but actually is not empty |
| public int E0024\_ONE\_NODE\_MUST\_BE\_SELECTED\_CODE = 24 | One node must be selected | Attempt to perform ‘At’ operation but XQL returned mode than one node for the specified XPath |
| public int E0025\_XML\_FILE\_ERROR\_CODE = 25 | VXml Parser - file read error | Unexpected error when reading XML file in ‘Load’ method |
|  |  |  |

* + 1. Methods

|  |  |  |
| --- | --- | --- |
| **Name** | **Parameter(s)** | **Description** |
| public static string GetMessage(int code) | **code** – error code | Static method, returns the basic error message for error code. |

* 1. public class VXmlDummy

Parent class for internal node types and their collections.

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public long Id | Always 0. |
| public string ID | Always “N/A”. |

* + 1. Constructors

No public constructors.

* + 1. Methods

No public methods.

* 1. public class VXmlAttribute : VXmlDummy

Class for ‘attribute’ node type.

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public string Name | Attribute name. |
| public string NodeType | Node type string representation. |
| public short NodeTypeCode | Node type numeric representation. |
| public string Value | Attribute value. |

* + 1. Constructors

No public constructors.

* + 1. Methods

No public methods.

* 1. public class VXmlComment : VXmlDummy

Class for ‘comment’ node type.

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public string Name | Node name, system-generated. |
| public string NodeType | Node type string representation. |
| public short NodeTypeCode | Node type numeric representation. |
| public string Value | Comment value. |

* + 1. Constructors

No public constructors.

* + 1. Methods

No public methods.

* 1. public class VXmlText : VXmlDummy

Class for ‘text’ node type.

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public string Name | Node name, system-generated. |
| public string NodeType | Node type string representation. |
| public short NodeTypeCode | Node type numeric representation. |
| public string Value | Text value. |

* + 1. Constructors

No public constructors.

* + 1. Methods

No public methods.

* 1. public class VXmlTag : VXmlDummy

Class for ‘tag’ node type.

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public string Name | Node name, system-generated. |
| public string NodeType | Node type string representation. |
| public short NodeTypeCode | Node type numeric representation. |
| public string Value | Tag value. |

* + 1. Constructors

No public constructors.

* + 1. Methods

No public methods.

* 1. public class VXmlAttributeCollection : VXmlDummy

Node attributes collection.

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public int Count | The number of nodes in the collection. |
| public VXmlAttribute this[int index] | Attribute node by index. |
| public VXmlAttribute this[string name] | Attribute node by node name |

* + 1. Constructors

No public constructors.

* + 1. Methods

No public methods.

* 1. public class VXmlCommentCollection : VXmlDummy

Node comments collection.

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public int Count | The number of nodes in the collection. |
| public VXmlComment this[int index] | Comment node by index. |
| public VXmlComment this[string name] | Comment node by node name |

* + 1. Constructors

No public constructors.

* + 1. Methods

No public methods.

* 1. public class VXmlTextCollection : VXmlDummy

Text nodes collection.

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public int Count | The number of nodes in the collection. |
| public VXmlText this[int index] | Text node by index. |
| public VXmlText this[string name] | Text node by node name |

* + 1. Constructors

No public constructors.

* + 1. Methods

No public methods.

* 1. public class VXmlTagCollection : VXmlDummy

Tag nodes collection.

* + 1. Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| public int Count | The number of nodes in the collection. |
| public VXmlText this[int index] | Tag node by index. |
| public VXmlText this[string name] | Tag node by node name |

* + 1. Constructors

No public constructors.

* + 1. Methods

No public methods.

1. **VXML Administration Tool**

Administration tool (VXmlExplorer) allows Virtual XML storage management via GUI.

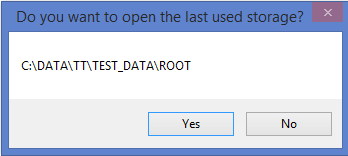
1. **Main menu (‘File’)**

The ‘File’ menu contains two kinds of items:

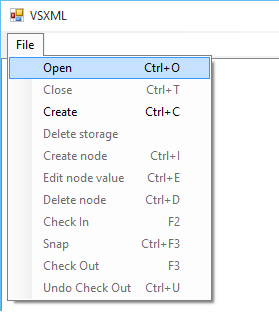
* 1. VXML Storage management (open, close, create)
  2. Node-specific functions when the particular node is selected (also available in the context menu).
  3. Open VXML Storage

*Note: the storage must be created before opening.*

If any storages was opened previously then system will prompt the confirmation to open this storage:

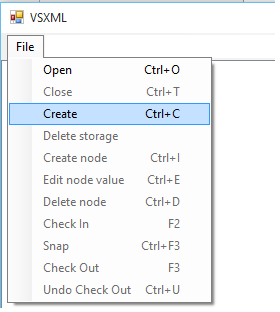


Otherwise you should use ‘Open’ menu and select the storage directory:

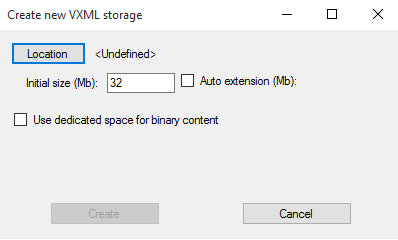


* 1. Create VXML Storage

Use ‘Create’ menu to create new VXML storage:



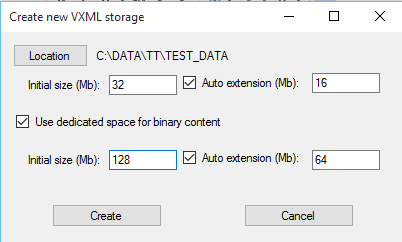
New pop-up window will be opened to define the storage parameters:



‘Location’ is mandatory parameter, you shall specify the root directory for storage.

‘Auto extension’ option can be used if you want to extend XML storage dynamically if the current size is insufficient.

‘Use dedicated storage for binary content’ allows create separate VStorage space for binary content:



Initial size and extension (optional) should be specified for this space.

* 1. Close VXML Storage

Close storage. After the operation is complete all other functions except ‘Open’ will be disabled.

* 1. Delete VXML Storage

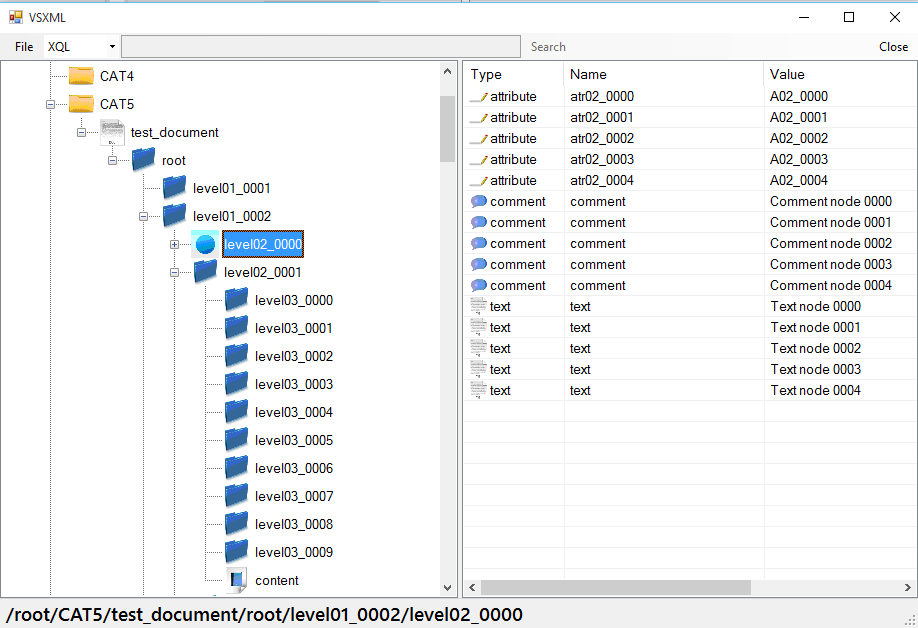
VXML Storage will be closed, XML data and binary content spaces physically deleted from the file system and Virtual Storage catalog.

Make sure you have backup copy of the storage before performing this action because roll back is not supported in this case.

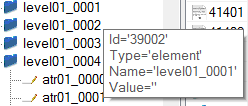
1. **Storage navigation**

The mail screen contains two panels:

* Tree view panel (left) – displays the XML tree in the storage. Node types included: catalog, document, element, content, reference.
* Detail (list) view panel (right) – displays the detail list of nodes for selected node in the tree view. Node types included: text, attribute, comment.



If you position the mouse cursor upon any node in the tree the tooltip text will show the node information:



*Click* on the node in the node tree will populate the list of all child nodes in the list view.

*Doubleclick* on the node in the list view will automatically select the node in the tree view and display all its children in the list view.

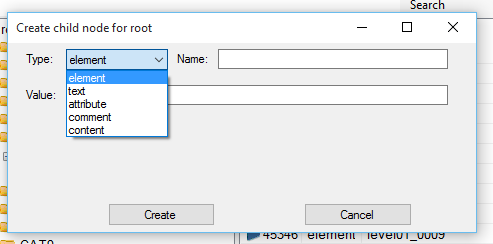
1. **Context menu**

The context is displayed by the mouse right click on the node in the tree view of list view.

The set of menu items may differ depending on view (tree or list), node type or node current state.

* 1. Create Node

The pop-up window for node creation will be opened:



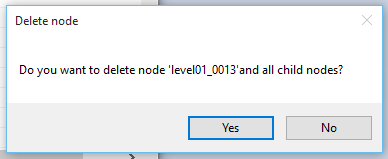
The list of node types for creation may differ depending on current node type.

Only allowed node types will be displayed.

New node will be created and added as child of the current node.

* 1. Delete Node

Confirmation pop-up window will be displayed:

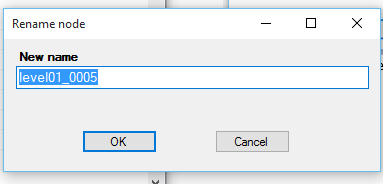


Fter the confirmation this node and all child nodes will be deleted.

* 1. Rename Node

This operation is allowed only for nodes that can be renamed.

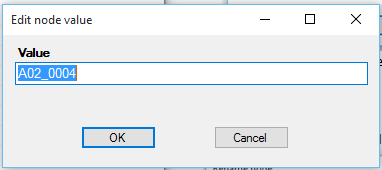
The pop-up window requesting the new node name will be displayed.



After renaming the node will be opened in the tree view.

* 1. Edit Value

The pop-up window requesting the new node value will be displayed.



The current node in the view will remain, displayed value changed.

* 1. Lookup original node

This function is available for ‘reference’ node type only (displayed with ‘~’ prefix.

The original node location will be opened in the tree view for this reference.

* 1. Check In

Check in node from the portable file, standard dialog window will be opened to specify the file location.

If GUID associated in the portable file doesn’t exist in this storage then XML data from the file will be added as a child of the current node. Otherwise the node identified by this GUID will be replaced (this can be not current node).

The checked in node will be opened in the tree view.

* 1. Check Out

Current node will be checked out to the portable file format (‘vsnp’).

Standard dialog window will be opened to specify the folder where this file will be saved.

The name of the file will be equal to GUID for this node (it will be assigned automatically if node have no GUID yet).

The node and all subtree will become read-only if operation is successful.

* 1. Snap

This operation is equal to ‘Check Out’ but only ‘vsnp’ file will be created, the node and all subtree will remain writable.

* 1. Undo CheckOut

Change state to writable for the node and all its subtree if it was checked out.

* 1. Copy node name to clipboard

Place the current node name to the clipboard. This could be useful to create XQL expressions.

* 1. Download Content

This operation is applicable only for ‘content’ node type.

The binary content will be saved to file, standard ‘Save’ dialog window will be displayed.

* 1. View XML

Display the current node and subtree as XML representation in the text box.

The binary content will not be included for ‘content’ nodes.

* 1. Save XML

Save the current node and subtree as XML representation in the text file.

The file name is equal to node name, the file type is ‘xml’.

The binary content will not be included for ‘content’ nodes.

* 1. Save XML wit content

Save the current node and subtree as XML representation in the text file.

The file name is equal to node name, the file type is ‘xml’.

The binary content will be saved in ‘files’ directory, the corresponding references will be added to xml file as ‘fileref’ attribute of the ‘content’ node.

* 1. Load XML

Load XML data from the XML file and append as a child of the current node.

The binary content will be loaded if XML file has ‘fileref’ attribute of the ‘content’ node.

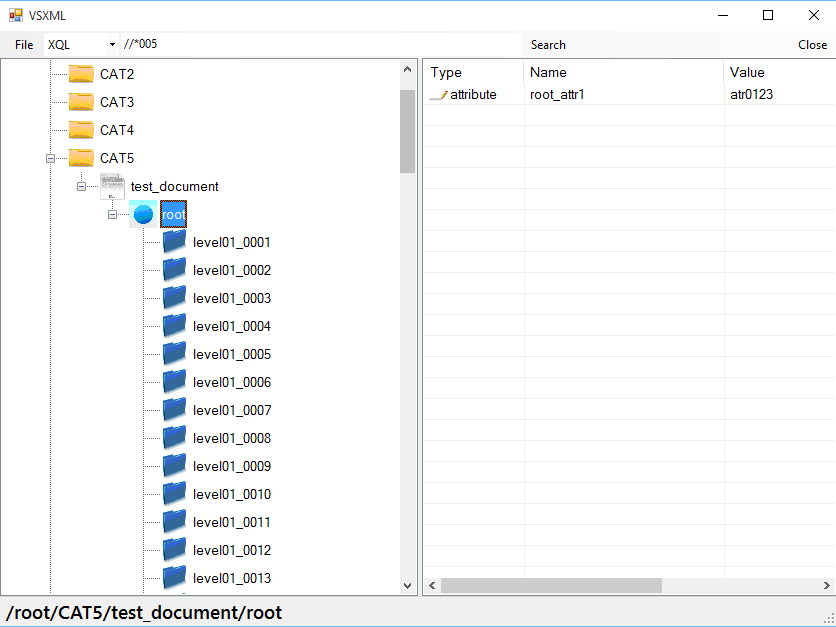
1. **Execute XQL query**

You can enter XQL expression in the text box right of ‘File’ menu.

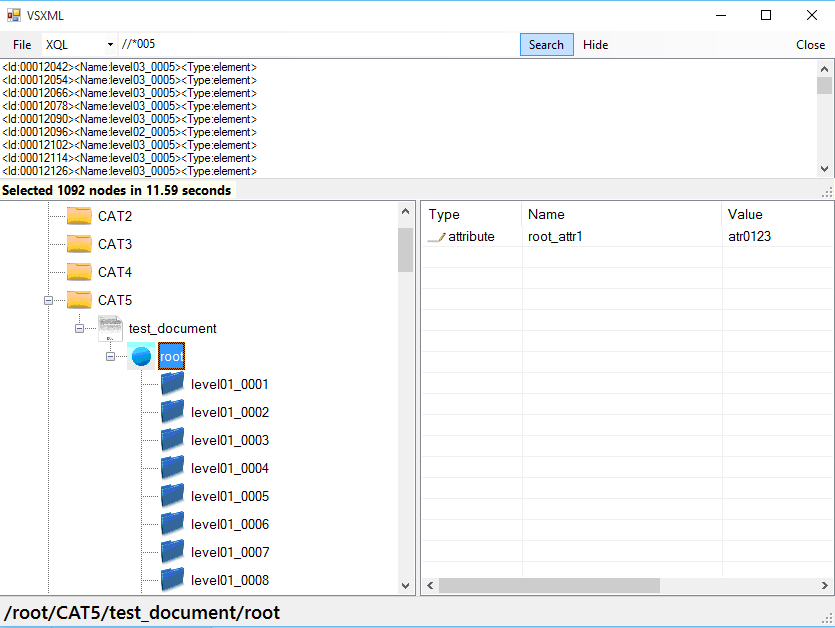
The search will be performed relatively to the current node.

See ‘3. Extended XQL’ chapter for search expression format.

Search starts by pressing ‘Enter’ key or clicking ‘Search’ button on the right of the text box:



After completion the search result panel will be displayed:



Doubleclick on the node in the search result list will open this node in the tree view.

‘Hide’ button will be visible after the query and can be used to hide search result list.

1. **Code sample**
2. Open VXML storage

ROOT = "";

if (path == "")

{

ROOT = VSUILib.VSUICommonFunctions.SelectPath(DEFS.KEY\_STORAGE\_ROOT, "Select МXML storage path");

if ((ROOT == VSUILib.VSUICommonFunctions.CANCELLED) | (ROOT == ""))

{

ROOT = "";

return;

}

}

else

ROOT = path;

try

{

cont = new VXmlCatalog(ROOT);

}

catch (VXmlException e)

{

MessageBox.Show(e.Message, "VXML Error", MessageBoxButtons.OK);

cont.Close();

return;

}

cont.Commit();

1. CheckOut node

if (CONTEXT\_XML\_NODE.CheckedOut)

{

MessageBox.Show("Node is already checked out", "Error", MessageBoxButtons.OK);

return;

}

// Prompt directory

string st = VSUILib.VSUICommonFunctions.SelectPath(DEFX.KEY\_SNAP, "Select target directory");

if (st != "")

{

bool state = true;

if (lck)

tsNode.Text = "Checking out " + CONTEXT\_XML\_NODE.Name + " ...";

else

tsNode.Text = "Snapping " + CONTEXT\_XML\_NODE.Name + " ...";

Application.DoEvents();

cont.Begin();

try

{

CONTEXT\_XML\_NODE.CheckOut(st, lck);

}

catch (VXmlException e1)

{

MessageBox.Show(e1.Message, "Error", MessageBoxButtons.OK);

state = false;

}

cont.Commit();

if (state)

{

SELECT\_Node();

string op = lck ? "Check Out" : "Snap";

MessageBox.Show(op + " completed, GUID=" + CONTEXT\_XML\_NODE.GUID, "Successful", MessageBoxButtons.OK);

}

1. Display child nodes in the ‘TreeView’ control

VXmlNode node;

TreeNodeCollection tc;

Cursor.Current = Cursors.WaitCursor;

tvCat.BeginUpdate();

if (CONTEXT\_TREE\_NODE == null)

{

node = cont;

CONTEXT\_XML\_NODE = cont;

CONTEXT\_TREE\_NODE = tvCat.Nodes.Add(node.ID, UTIL\_PrepareName(node));

CONTEXT\_TREE\_NODE.ImageKey = node.NodeType;

ACTION\_SetTreeNodeName(CONTEXT\_TREE\_NODE, node);

tc = CONTEXT\_TREE\_NODE.Nodes;

tvCat.SelectedNode = CONTEXT\_TREE\_NODE;

}

else

{

tc = CONTEXT\_TREE\_NODE.Nodes;

node = CONTEXT\_XML\_NODE;

}

tc.Clear();

if (node.NodeTypeCode != DEFX.NODE\_TYPE\_ATTRIBUTE)

{

VXmlNode c = node.FirstAttribute;

while (c != null)

{

TreeNode tn = tc.Add(c.ID, UTIL\_PrepareName(c));

tn.ImageKey = c.NodeType;

ACTION\_SetTreeNodeName(tn, c);

c = c.Next;

}

c = node.First;

while (c != null)

{

TreeNode tn = tc.Add(c.ID, UTIL\_PrepareName(c));

tn.ImageKey = (c.NodeTypeCode == DEFX.NODE\_TYPE\_REFERENCE)? ((VXmlReference)c).ReferenceNode.NodeType : c.NodeType;

ACTION\_SetTreeNodeName(tn, c);

c = c.Next;

}

}

tvCat.EndUpdate();

Cursor.Current = Cursors.Default;

1. Search nodes using XQL expression

VXmlNode n = null;

string st = mnuSearchTextBox.Text.Trim();

if (st.Length > 0)

{

decimal d = 0;

if (CONTEXT\_LIST\_VIEW\_ITEM == null)

n = CONTEXT\_XML\_NODE;

else

n = cont.GetNode(VSLib.ConvertStringToLong(CONTEXT\_LIST\_VIEW\_ITEM.SubItems[0].Text));

if (n.NodeTypeCode == DEFX.NODE\_TYPE\_REFERENCE)

n = ((VXmlReference)n).ReferenceNode;

VXmlNodeCollection l = null;

try

{

lbResult.Items.Clear();

pnQuery.Visible = true;

tsNodeCount.Text = "Running query...";

Application.DoEvents();

Cursor.Current = Cursors.WaitCursor;

long l\_start = DateTime.Now.Ticks;

l = n.SelectNodes(st);

long l\_end = DateTime.Now.Ticks;

d = l\_end - l\_start;

}

catch (VXmlException e1)

{

MessageBox.Show(e1.Message, "Error", MessageBoxButtons.OK);

}

if (l != null)

{

tsNodeCount.Text = "Selected " + l.Count.ToString() + " nodes in " + (d / 10000000).ToString("F") + " seconds";

Application.DoEvents();

Cursor.Current = Cursors.WaitCursor;

lbResult.BeginUpdate();

for (int i = 0; i < l.Count; i++)

{

lbResult.Items.Add("<Id:" + l[i].Id.ToString("D8") + "><Name:" + l[i].Name + "><Type:" + l[i].NodeType + ">");

}

lbResult.EndUpdate();

}

else

tsNodeCount.Text = "";

Cursor.Current = Cursors.Default;

pnQuery.Visible = true;

mnuHide.Visible = true;

SHOW\_Buttons();

}