QUICKSTART GUIDE TO XML

XML (Extensible Markup Language) is a markup language for documents containing structured information. XML is all about metadata and the idea that certain groups of people have similar needs for describing and organizing the data they use. Like HTML, XML is a set of tags and declarations -- but rather than being concerned with formatting information on a page, XML focuses on providing information about the data itself and how it relates to other data.

Some data types are pretty much universal (<First Name>, <Address>, <City>, and so forth). Others are industry or even company-specific (<price>, <manufacturer>, <component>). Healthcare organizations, for example, have a whole set of data types and acronyms understandable (some would say penetrable) only to claims processors. XML allows each of these data types to be easily recognized and, for site developers, used to create sites optimized around both the data and the people using it.

If you're designing data-hungry sites, especially for intranets, you should be getting excited about XML, because in XML, you'll be able to create and respond to much richer set of data elements. That will in turn let you build more individualized dynamic sites and pages. For example, your site's users could access information across databases and types of data without having to rely on a search engine.

The structured XML information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure.
A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

Step 1: Create a XML document.

The creation of an XML document can be done in two ways:

1. Create a XML document from an existing file

```
Do(
  var([docID], vbValue, XMLCreateDocument),
  XMLOpenFile(docID, [C:\\project01.xml])
)
```

2. Create a XML document from an existing string

```
Do(
  var(
    [strXML],
    vbString,
    [<root><atom>Source</atom><atom>Queue</atom></root>]
  ),
var([docID], vbValue, XMLCreateDocumentFromXMLString(strXML))
)
```

3. Create an empty XML document
```
var([docID], vbValue, XMLCreateDocument)
```

For the following examples this XML document is considered:

```
<root>
  <atoms>
    <atom id='1'>
      <name>Source</name>
      <x_position>1</x_position>
      <y_position>3</y_position>
    </atom>
    <atom id='2'>
      <name>Server</name>
      <x_position>2</x_position>
      <y_position>4</y_position>
    </atom>
  </atoms>
</root>
```

Step 2: Get data from the XML document

There are several ways to obtain information out of a XML document, these ways depend on the structure of the XML document.

- Obtain the name of the root node of the XML document:

```
Do(
 var([docID], vbValue, XMLCreateDocument),
 XMLOpenFile(docID, [C:\\example.xml]),

 var([rootNodeID], vbValue, XMLGetRootNode(docID)),
 XMLNodeName(docID,rootNodeID)
)
```

- Obtain the name of the atoms in the XML document:

```
Do(
 var([docID], vbValue, XMLCreateDocument),
 XMLOpenFile(docID, [C:\\example.xml]),

 var([rootNodeID], vbValue, XMLGetRootNode(docID)),
 var(
  [nodeID],
  vbValue,
  XMLNodeGetChildByName(docID, rootNodeID, [atoms])
 ),

 var(
  [valNumberAtoms],
  vbValue,
  XMLNodeGetChildCount(docID, nodeID)
 ),

 Repeat(
  valNumberAtoms,
  Do(
   {Get the node "atom" in the atoms tree}
   var(
    [nodeChildID],
    vbValue,
    XMLNodeGetChild(docID, nodeID, Count)
   ),

   {Get the "name" node in the atom tree}
   var(
    [nodeAtomChildID],
    vbValue,
    XMLNodeGetChildByName(docID, nodeChildID,[name])
   ),

   {Print the node value}
   trace(string(XMLNodeGetText(docID, nodeAtomChildID)))
  )
 )
)
```

- Obtain the atom-id's from the XML document:

```
Do(
 var([docID], vbValue, XMLCreateDocument),
 XMLOpenFile(docID, [C:\\example.xml]),

 var([rootNodeID], vbValue, XMLGetRootNode(docID)),
 var(
  [nodeID],
  vbValue,
  XMLNodeGetChildByName(docID, rootNodeID, [atoms])
 ),

 var(
  [valNumberAtoms],
  vbValue,
  XMLNodeGetChildCount(docID, nodeID)
 ),

 Repeat(
  valNumberAtoms,
  Do(
   var(
    [nodeChildID],
    vbValue,
    XMLNodeGetChild(docID, nodeID, Count)
    ),
    trace(
     string(XMLNodeGetAttribute(docID, nodeChildID,[id]))
    )
   )
  )
 )
```

Step 3: Manipulate the XML document

There are also several function to manipulate the XML document:

```
Do(
 var([docID], vbValue, XMLCreateDocument),

 var([rootNodeID], vbValue, XMLSetRootNode(docID, [root])),
 var(
  [childNodeID],
  vbValue,
  XMLNodeAddChild(docID, rootNodeID, [atoms])
 ),

 var(
  [serverNodeID],
  vbValue,
  XMLNodeAddChild(docID, childNodeID, [atom])
 ),

 var(
  [queueNodeID],
  vbValue,
  XMLNodeAddChild(docID, childNodeID, [atom])
 ),

 XMLNodeSetText(docID, serverNodeID, [Server]),
 XMLNodeSetText(docID, queueNodeID, [Queue]),

 var([strXML], vbString, XMLGetXMLString(docID))
)
```

The resulting string looks like:

Resulting string:

```
<root>
  <atoms>
    <atom>Server</atom>
    <atom>Queue</atom>
  </atoms>
</root>
```