TIPO DE DATOS XMLWRITER

Introducción

El objetivo del tipo de datos XMLWriter es proveer la posibilidad de grabar archivos XML de un modo más sencillo que las funciones incluidas en versiones anteriores. El documento XML contiene una introducción a dicho lenguaje.
Se recomienda leer también el documento XMLReader.

Alcance

Objetos: Transacciones, Work Panels, Web Panels, Reportes, Procedimientos.
Lenguajes: Java – Visual Basic – Visual Fox – C/SQL – C#.
Interfaces: Web Form, Win Form.

Descripción

Para poder crear un archivo XML desde un objeto ***GeneXus***, se debe definir una variable de un nuevo tipo de datos denominado XmlWriter (xmlwrt) y luego invocar a los métodos necesarios para crear los nodos que lo componen.

Métodos:

MÉTODOS BÁSICOS:

Open([FileName]**)**

Permite crear un documento XML.
File Name es el nombre de archivo con el cual se va a crear el documento, si se omite, el mismo se genera por "standard output". En caso de ser un objeto web se genera en el response. (Funciona con CGI, ISAPI WebClasses y Servlets).

FileName – Character

OpenToString()

Permite crear un documento XML a un buffer inerno en lugar de a un archivo.
El contenido del buffer puede ser consultado por medio de la propiedad ResultingString

WriteStartElement(<ElementName>**)**

Comienza un elemento compuesto. Este elemento puede tener subelementos, por lo que es valido anidar llamadas a WriteStarElement/WriteElement.

<ElementName> - Character

WriteElement(<ValueName>, <ElementValue>**)**

Almacena un valor dentro del tag de nombre ValueName. En el caso de que

ElementValue sea de tipo Character, se sustituyen los caracteres especiales por secuencias de caracteres (el '<' se sustituye por '<', el '>' por '>', etc.).
A diferencia del caso anterior este elemento no contiene subelementos.

<ValueName> - Character
<ElementValue> - Character

WriteText(<Value>**)**

Genera character data con el string <Value>. Se sustituyen los caracteres especiales por secuencias de caracteres (el '<' se sustituye por '<', el '>' por '>', etc.).

<Value> - Character

WriteRawText(<Value>)

Genera cualquier texto sin sustituir los caracteres especiales por secuencias de caracteres.

<Value> - Character

WriteAttribute(<AttriName>, <AttriValue>**)**

Asigna un atributo al último elemento. Es decir, al elemento que se creó con la última invocación a WriteStartElement(<ElementName>) o WriteElement(<ValueName> , <ElementValue>).

<AttriName> - Character
<AttriValue>- Character

WriteEndElement()

Cierra el último elemento abierto.

Close()

Cierra el documento.

OTROS MÉTODOS:

WriteComment(<Comment>**)**

Escribe el comentario indicado en <Comment>

Ejemplo:
WriteComment('Comentario') genera lo siguiente:
<!— Comentario -->

<Comment> - Character

WriteEntityReference(<Entity>**)**

Escribe una referencia a la entidad <Entity>

Ejemplo:
WriteEntityReference('Entidad') genera lo siguiente:
&Entidad;

<Entity> - Character

WriteCData(<DataValue>**)**

Escribe una sección Cdata con el valor <DataValue>.
Si el contenido especificado para la sección Cdata contiene la secuencia de caracteres]]>, el mismo es fraccionado y se generan tantas secciones Cdata como sean necesarias para obtener un documento XML bien formado.

Ejemplo:
WriteCData('Value') genera lo siguiente:
<![CDATA[value]]>

<DataValue> - Character

WriteProcessingInstruction(<Action>, <Value>**)**

Escribe el registro del tipo processing instruction indicado por <Action> y <Value>

Ejemplo:
WriteProcessingInstruction('play', 'sound = "Hello.wav"') genera lo siguiente:
<? Play sound="Hello.wav" ?>

<Action> - Character
<Value > - Character

WriteStartDocument([Encoding, [StandAlone]]**)**

Escribe la declaración de XML usando la versión 1.0 y codificación ISO-8859-1. Opcionalmente se puede indicar un entero (0/1) como valor booleano para usar en la declaración standalone y un encoding.
Este método debe ser llamado al comienzo de la generación del documento (luego del open() y antes de cualquier otro método).

Ejemplo:
WriteStartDocument() genera lo siguiente:
<?xml version="1.0" enconding="ISO-8859-1" ?>

Enconding - Character
StandAlone – Integer (0/1)

WriteDocType(<DocName> [,SubSet]**)**

Escribe la declaración DOCTYPE del documento XML. Si se incluye un SubSet se ingresa entre corchetes al final de la declaración.

Ejemplo:
WriteDocType('<book>', '<!ENTITY ge "entity">') genera lo siguiente:
<!DOCTYPE <book> [<!ENTITY ge "entity">]>

<DocName> - Character
SubSet – Character

WriteDocTypeSystem(<DocName>,<uri> [,SunSet]**)**

Escribe la declaración DOCTYPE del documento XML con una declaración de tipo SYSTEM. Si se incluye un SubSet se ingresa entre corchetes al final de la declaración.

Ejemplo:
WriteDocTypeSystem('<book>', 'file.dtd') genera lo siguiente:<!DOCTYPE <book> SYSTEM "file.dtd">

<DocName> - Character
<uri> - Character
SubSet – Character

WriteDocTypePublic (<DocName>,<PubId>, <uri> [,SunSet]**)**

Escribe la declaración DOCTYPE del documento XML con una declaración de tipo PUBLIC. Si se incluye un SubSet se ingresa entre corchetes al final de la declaración.

Ejemplo:
WriteDocTypePublic('<book>','Id', 'http://www.ser.com/dtd') genera lo siguiente:
<!DOCTYPE <book> PUBLIC "Id" "http://www.ser.com/dtd">

<DocName> - Character
<PubId> - Character
<uri> - Character
Subset - Character

Propiedades:

Indentation
Define cuantos caracteres se utilizan para indentar el código generado. Por defecto es 2.

IndentChar
Define el carácter que se utiliza para indentar el código generado. Por defecto se utiliza un espacio en blanco.

ErrCode
Retorna un valor mayor que cero en caso de haber ocurrido un error durante la generación del documento XML.

ErrDescription
Contiene la descripción del error ocurrido cuando ErrCode tiene un valor distinto de cero.

ResultingString
Permite consultar el valor del documento XML que se encuentra en un buffer interno cuando el documento se creó a partir del método OpenToString().
El documento retornado puede ser incompleto si se invoca a la propiedad antes de la ejecución del método close()

Soporte para Name Spaces

En esta sección se describen las funcionalidades disponibles para generar documentos XML que utilizan name spaces.

MÉTODOS:

WriteNSStartElement(<LocalName> [,Prefix, NameSpaceURI]**)**

Este método es análogo al WriteStartElement.
Si se indica un NameSpaceURI que no está en el ámbito de definición del elemento, o este tiene un prefijo diferente del parámetro Prefrix, se crea automáticamente el atributo xmlns:prefix=URI

<LocalName> - Character
Prefix - Character
NameSpaceURI - Character

WriteNSElement(<LocalName> [,NameSpaceURI [,Value]]**)**

Este método es análogo al WriteElement.
El prefijo se determina automáticamente en base a los prefijos definidos en el ámbito del elemento. Si ningún prefijo estuviera asociado al URI especificado, se define junto con el elemento un name space por defecto.

Ejemplo:
WriteNSElement('Precio', '20.5', 'http://www.genexus.com') genera lo siguiente:
<prefijo:Precio>20.5</prefijo:Precio>
o
<Precio xmlns=" http://www.genexus.com">20.5</Precio>

<LocalName> - Character
NameSpaceURI - Character
<Value > - Character

WriteAttribute(<LocalName>, <Value>**)**

Cuando se trabaja con documentos con name sapace, el método WriteAttribute tiene un comportamiento especial para algunos atributos. Si el nombre del atributo es "xmlns" o comienza con "xmlns:" se registra la definición del atributo como la definición de un name space.
De esta forma, en futuras invocaciones de un método WriteNSStartElement o WriteNSElement puede determinarse el prefijo correspondiente a una determinada URI.

EJEMPLO DE SOPORTE DE NAME SPACE:

El siguiente procedimiento genera el archivo ejemplo.xml

```
&writer.Open('ejemplo.xml')
&writer.WriteNSStartElement('a')
    &writer.WriteAttribute('xmlns:p1', 'http://www.artech.com')
    &writer.WriteAttribute('xmlns', 'http://defecto.com')
    &writer.WriteAttribute('att', 'a1')
```

```
    &writer.WriteNSElement('b', 'http://www.artech.com')
    &writer.WriteAttribute('att', 'a1')
    &writer.WriteAttribute('p1:att2', 'a2')
    &writer.WriteNSElement('b', 'http://www.genexus.com')
    &writer.WriteAttribute('xmlns:p1','http://www.genexus.com')
    &writer.WriteAttribute('xmlns:p2','http://www.artech.com/segundo')
    &writer.WriteAttribute('p2:att1', 'a1')
  &writer.WriteEndElement()
  &writer.Close()
```

El archivo ejemplo.xml queda de la siguiente forma:

```
<a
  xmlns:p1="http://www.artech.com"
  xmlns="http://defecto.com"
  att="a1">
  <p1:b
        att1="a1"
        p1:att2="a2"
  />
  <p1:b xmlns:p1="http://www.genexus.com"
       xmlns:p2="http://www.artech.com/segundo"
        p2:att1="a1"
  />
</a>
```

Ejemplo

El siguiente procedimiento genera un archivo llamado Reunion.xml que contiene datos de una reunión, indicando que integrantes participaron de la misma, y cuales son las tareas de cada uno.

```
&filexml.open('Reunion.xml')
&filexml.WriteStartDocument()

&filexml.WriteStartElement('REUNION')
   &filexml.WriteAttribute('Fecha', dtoc(ReuFch) )
   &filexml.WriteElement('FECHA', dtoc(ReuFch) )
   &filexml.WriteComment('Descrpción de la reunión')
   &filexml.WriteCData(ReuDsc )
   &filexml.WriteStartElement('INTEGRANTES')
      for each
         &filexml.WriteElement('INTEGRANTE', ReuPerNom  )
      endfor
   &filexml.WriteEndElement()
   &filexml.WriteStartElement('TAREAS')
      for each
         &filexml.WriteStartElement('TAREA')
            &filexml.WriteElement('RESPONSABLE', ReuTarPerNom)
            &filexml.WriteCData(ReuTarDsc  )
         &filexml.WriteEndElement()
      endfor
   &filexml.WriteEndElement()
&filexml.WriteEndElement()
&filexml.Close()
```

El archivo reunion.xml contiene:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<REUNION Fecha="06/03/01">
    <FECHA>06/03/01</FECHA>
    <!-- Descrpción de la reunión-->
       <![CDATA[ Reunion del equipo de desarrollo de la aplicación.
       La reunión fue realizada el Viernes a las 9:30 horas.]]>
  <INTEGRANTES>
      <INTEGRANTE>Juan Pedro</INTEGRANTE>
      <INTEGRANTE>Laura</INTEGRANTE>
      <INTEGRANTE>Diego</INTEGRANTE>
      <INTEGRANTE>Florinda</INTEGRANTE>
    </INTEGRANTES>
  <TAREAS>
    <TAREA>
        <RESPONSABLE>Juan Pedro</RESPONSABLE>
        <![CDATA[ Realizar la documentción de la aplicación]]>
      </TAREA>
    <TAREA>
        <RESPONSABLE>Florinda</RESPONSABLE>
        <![CDATA[ Reunion con los clientes.]]>
      </TAREA>
    <TAREA>
        <RESPONSABLE>Laura</RESPONSABLE>
        <![CDATA[ Realizar el maunal de usuario]]>
      </TAREA>
    <TAREA>
        <RESPONSABLE>Diego</RESPONSABLE>
        <![CDATA[ Documentar las especificaciones.]]>
      </TAREA>
    </TAREAS>
  </REUNION>
```

Glosario

Las siguientes definiciones fueron extraídas de "Extensible Markup Language (XML) 1.0 (Second Edition)"; por más información referirse a http://www.w3.org/TR/2000/REC-xml-20001006 ; y para el caso de los Namespaces referirse a http://www.w3.org/TR/1999/REC-xml-names-19990114/

- **Character data:** Markup takes the form of start-tags, end-tags, empty-element tags, entity references, character references, comments, CDATA section delimiters, document type declarations, processing instructions, XML declarations, text declarations, and any white space that is at the top level of the document entity (that is, outside the document element and not inside any other markup).
  All text that is not **markup** constitutes the **character data** of the document.

- **Entities:** An XML document may consist of one or many storage units. These are called entities; they all have content and are all (except for the document entity and the external DTD subset) identified by entity name.
  Entities may be either parsed or unparsed.
  A **parsed entity's** contents are referred to as its replacement text; this text is considered an integral part of the document.
  An **unparsed entity** is a resource whose contents may or may not be text, and if text, may be other than XML. Each unparsed entity has an associated notation, identified by name. Beyond a requirement that an XML processor make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.
  Parsed entities are invoked by name using entity references; unparsed entities by name, given in the value of ENTITY or ENTITIES attributes.

- **Entity reference:** Refers to the content of a named entity.

- **CDATA sections** may occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup. CDATA sections begin with the string "<![CDATA[" and end with the string "]]>".

- **Processing instructions** (PIs) allow documents to contain instructions for applications.

- **Standalone document declaration:** Markup declarations can affect the content of the document, as passed from an XML processor to an application; examples are attribute defaults and entity declarations. The standalone document declaration, which may appear as a component of the XML declaration, signals whether or not there are such declarations which appear external to the document entity or in parameter entities.

- **Document type declaration:** XML provides a mechanism, the document type declaration, to define constraints on the logical structure and to support the use of predefined storage units. An XML document is **valid** if it has an associated document type declaration and if the document complies with the constraints expressed in it.
  The XML **document type declaration** contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or **DTD**. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. The DTD for a document consists of both subsets taken together.

- **XML namespaces** provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.