The NET Xml Writer

Last Updated on Fri, 07 Jan 2022 | IronPython Introduction

The .NET XML support is enshrined in the System.Xml assembly. The System.Xml namespace has classes for reading and writing XML. There are further namespaces for working with XML schema, XSLT transformations, and XPath queries. Table 5.2 shows the major XML namespaces in .NET 2.0.

Table 5.2 The .NET XML namespaces

Namespace			Purpose
System	Xml		Provides support for standards-based XML support. Includes the XmlReader and XmlWriter classes.
System	Xml	Schema	Support for XML Schemas in the form of schema definition language (XSD) schemas.
System	Xml	Serialization	Classes to serialize objects in XML form.
System	Xml	XPath	Classes to work with the XQuery 1.0 and XPath 2.0 Data Model.
System	Xml	Xsl	Support for Extensible Stylesheet Language Transformation (XSLT) transforms.

The basic classes for reading and writing XML documents are the XmlReader and XmlWriter. Because you'll be modifying the SaveCommand, we start with the XmlWriter.

XmlWriter is designed for creating conformant documents. The documents it creates will be valid XML, capable of being read by any standards-based reader. Along with the XmlWriter class, you use XmlWriterSettings. This is a class used for configuring an XmlWriter instance; you set attributes on the XmlWriterSettings instance to configure how the XML is written out.

Table 5.3 shows the different settings (properties) on XmlWriterSettings. The defaults are largely sensible, but we do like to change a couple. We like the XML tags

Table 5.3 The properties of XmlWriterSettings and the default values

Property	Initial value
CheckCharacters	True
CloseOutput	False
ConformanceLevel	Document
Encoding	Encoding.UTF8 (Encoding lives in the System.Text namespace, and is a useful class.)
Indent	False
IndentChars	Two spaces

NewLineChars	\r\n (carriage return, new line)
NewLineHandling	Replace
NewLineOnAttributes	False
OmitXmlDeclaration	False

to be indented with each level of nesting. This gives you a visual indication of the structure of the document (and we all know that indentation to indicate structure is a brilliant idea). The following segment creates an XmlWriterSettings instance, and sets the two properties required for indentation with four spaces. Because you haven't yet used the System.Xml assembly, you first need to add a reference to it.

>>> from System.Xml import XmlWriter, XmlWriterSettings >>> settings = XmlWriterSettings() >>> settings . Indent = True

>>> settings.IndentChars = ' ' # four spaces

You don't instantiate a new XmlWriter instance directly; instead, you call the static method Create, which returns a new instance of the correct type of writer for the settings passed in. There are various call signatures for Create; for example, you could pass in a filename and your settings object, and the writer would create the file for you. If you don't want the writer to be responsible for creating the file, you can pass in an opened FileStream instead. In the segments that follow, you'll pass in a String-Builder instance. StringBuilder is in the System.Text namespace, and is a mutable string type—it allows strings to be built up incrementally.

An odd side effect of passing a StringBuilder to the XmlWriter is that it will refuse to write any encoding in the XML declaration other than UTF-16.2 Because

2 Because the writer is writing into a string, which is still Unicode and has no encoding yet, it's no wonder that it gets confused. The logic is possibly that the Windows internal UCS2 Unicode representation is most like UTF-16. Still, ignoring the explicit encoding on the XmlWriterSettings is a dubious practice.

you're likely to be happy with the default encoding of UTF-8, you set OmitXmlDeclara-tion to True.

>>> settings.OmitXmlDeclaration = True >>> from System.Text import StringBuilder >>> document = StringBuilder()

>>> writer = XmlWriter.Create(document, settings) >>> writer.WriteStartDocument() >>> writer.WriteStartElement("document") >>> writer.WriteStartElement("page")

>>> writer.WriteAttributeString("title", "A page title") >>> writer.WriteString("This is a page contents") >>> writer.WriteEndElement() >>> writer.WriteEndElement() >>> writer.WriteEndDocument() >>> writer.Flush() >>> writer.Close() >>> print document.ToString() <document>

<page title="A page title">This is a page contents</page> </document>

This is great because (by happy coincidence) it's exactly the kind of structure that we want for our documents. Having to remember to close all the elements in the right order is a nuisance, though. This is what happens if you get it wrong:

>>> writer = XmlWriter.Create(document, settings) >>> writer.WriteEndElement() Traceback (most recent call last): File System.Xml, line unknown, in WriteEndElement File System.Xml, line unknown, in AdvanceState

File System.Xml, line unknown, in ThrowInvalidStateTransition SystemError: Token EndElement in state Start would result in an invalid XML document. Make sure that the ConformanceLevel setting is set to ConformanceLevel.Fragment or ConformanceLevel.Auto if you want to write an XML fragment.

Oops. A sensible way to avoid this is to make sure that your XML document structure and your program structure are as similar as possible. Ideally, a top-level method should create (and close) the root node, calling down to submethods to write out its child elements. Each method should only be responsible for creating and closing a single element, again calling submethods for their child elements. This way you know that, as long as the code runs to completion, every element will be closed and the result will be valid XML. A nice side effect is that this is also a good way to write modular and readable code—which is important because you're writing Python. In the next section, we apply this strategy to MultiDoc.

Continue reading here: Document Writer Class

Related Posts

- This is the content of the element
- Iron Python and the CLR - IronPython Introduction
- WPF in action - IronPython Introduction
- Event handlers in Iron Python
- Ironpython Enum - IronPython Introduction
- XPS documents and flow content
- Improve Your Handwriting
- Guru Tools for Microsoft Access
- How To Create Your Own Programming Language
- Advanced Registry Cleaner PC Diagnosis and Repair
- Search Engine Traffic Guide
