

How do I create XSLT templates?

XSLT stands for Extensible Stylesheet Language. XSLT is a language for transforming XML documents into other documents such as text or html documents. XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

This tutorial will show you step by step how to transform the XEML (XML) model into business entity in C#.

Declare Style Sheet

The root element that declares the document to be an XSLT style sheet is `<xsl:stylesheet>`. The correct way to declare an XSLT style sheet according to the W3C Recommendation is:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0" >
```

The <xsl:template> Element

An XSLT document contains one or more templates, which are created with the `<xsl:template match="entity">` tag. The `match` attribute is used to associate a template with an XML element.

Let's start by looking at a simple XML document (.xml model) and an XSLT stylesheet, which is used to transform the XML to C# code. The following XSL tags are most frequently used:

XML Document:

```
<entity name="Customer">
  <properties>
    <property name="CustomerID" type="string" nullable="true" />
    <property name="FirstName" type="string" nullable="true" />
    <property name="LastName" type="string" nullable="true" />
  </properties>
</entity>
```

XSLT Document:

```
<?xml version='1.0'?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
>
<xsl:output method="text"/>
<xsl:template match="entity">
using System;
using System.Collections.Generic;

namespace LatticeFramework
{
```

```

}
</xsl:template>
</xsl:stylesheet>

```

Example Explained

Since XSLT is an XML document, it always begins with the XML declaration:

```
<?xml version='1.0'?>
```

The next element, **<xsl:stylesheet>**, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).

The third line (shown below) indicates that the resulting output will be text.

```
<xsl:output method="text"/>
```

The **<xsl:template>** element defines a template. The `match="entity"` attribute associates the template with the `entity` element of the XML source document.

The content inside the `<xsl:template>` element defines some C# code to write to the output.

The last two lines define the end of the template and the end of the style sheet.

If we use this XSLT to transform the above XML document, the result will be:

```

using System;
using System.Collections.Generic;

namespace LatticeFramework
{
}

```

The <xsl:value-of> Element

The `<xsl:value-of>` is used to extract the value of selected node:

```

<?xml version='1.0'?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
>
<xsl:output method="text"/>
<xsl:template match="entity">
using System;
using System.Collections.Generic;

namespace LatticeFramework
{
  public partial class <xsl:value-of select="@name"/>
  {

```

```

        public <xsl:value-of select="@name"/>() {}
    }
}

</xsl:template>
</xsl:stylesheet>

```

The value of the **select** attribute is an XPath expression. An XPath expression works like navigating a file system; where a forward slash (/) selects subdirectories.

The result of the transformation above will look like this:

```

using System;
using System.Collections.Generic;

namespace LatticeFramework
{
    public partial class Customer
    {
        public Customer() {}
    }
}

```

The <xsl:for-each> Element

The <xsl:for-each> element can be used to select every XML element of a specified node-set:

```

<xsl:for-each select="properties//property">
private _<xsl:value-of select="lattice:Trim(@name)"/>;
</xsl:for-each>

```

The result of the transformation above will look like this:

```

private _CustomerId;
private _FirstName;
private _LastName;

```

The <xsl:choose> Element

The <xsl:choose> element is used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

Syntax

```

<xsl:choose>
  <xsl:when test="expression">
    ... some output ...
  </xsl:when>
  <xsl:otherwise>
    ... some output ....
  </xsl:otherwise>

```

```
</xsl:choose>
```

```
<xsl:for-each select="properties//property">

private <xsl:choose>
<xsl:when test="@type="integer"'>int</xsl:when>
<xsl:when test="@type="text"'>string</xsl:when>
<xsl:when test="@type="boolean"'>bool</xsl:when>
<xsl:when test="@type="binary"'>byte[]</xsl:when>
<xsl:when test="@type="datetime"'>DateTime</xsl:when>
<xsl:when test="@type="timestamp"'>DateTime</xsl:when>
<xsl:when test="@type="guid"'>System.Guid</xsl:when>
<xsl:otherwise><xsl:value-of select="@type"/></xsl:otherwise>
</xsl:choose> _<xsl:value-of select="@name"/>;

</xsl:for-each>
```

The result of the transformation above will look like this:

```
private int _CustomerID;
private string _FirstName;
private string _LastName;
```

Put Everything Together

Let's put everything together. We want to transform the following XML document into Business Entities:

XML Document:

```
<entity name="Customer">
  <properties>
    <property name="CustomerID" type="string" nullable="true" />
    <property name="FirstName" type="string" nullable="true" />
    <property name="LastName" type="string" nullable="true" />
  </properties>
</entity>
```

XSLT Document:

```
<?xml version='1.0'?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
>
<xsl:output method="text"/>
<xsl:template match="entity">
using System;
using System.Collections.Generic;

namespace LatticeFramework
{
  public partial class <xsl:value-of select="@name"/>DTO
  {
```

```

#region private fields
<xsl:for-each select="properties//property">
private <xsl:choose>
<xsl:when test="@type="integer">int</xsl:when>
<xsl:when test="@type="text">string</xsl:when>
<xsl:when test="@type="boolean">bool</xsl:when>
<xsl:when test="@type="binary">byte[]</xsl:when>
<xsl:when test="@type="datetime">DateTime</xsl:when>
<xsl:when test="@type="timestamp">DateTime</xsl:when>
<xsl:when test="@type="guid">System.Guid</xsl:when>
<xsl:otherwise>
    <xsl:value-of select="@type"/>
</xsl:otherwise>
</xsl:choose>
</xsl:for-each>
_<xsl:value-of select="@name"/>;
</xsl:for-each>
#endregion

public <xsl:value-of select="@name"/>DTO() {}

#region public properties
<xsl:for-each select="properties//property">
public <xsl:choose>
<xsl:when test="@type="integer">int</xsl:when>
<xsl:when test="@type="text">string</xsl:when>
<xsl:when test="@type="boolean">bool</xsl:when>
<xsl:when test="@type="binary">byte[]</xsl:when>
<xsl:when test="@type="datetime">DateTime</xsl:when>
<xsl:when test="@type="timestamp">DateTime</xsl:when>
<xsl:when test="@type="guid">System.Guid</xsl:when>
<xsl:otherwise>
    <xsl:value-of select="@type"/>
</xsl:otherwise>
</xsl:choose><xsl:text> </xsl:text>
<xsl:value-of select="$property"/>
{
    get { return _<xsl:value-of select="@name"/>; }
    set { _<xsl:value-of select="@name"/> = value; }
}
</xsl:for-each>
#endregion
}
}
</xsl:template>
</xsl:stylesheet>

```

If we use this XSLT to transform the above XML document, the result will be:

```

using System;
using System.Collections.Generic;

namespace MySchema
{
    public partial class CustomerDTO
    {
        private string _CustomerID;

```

```
private string _FirstName;
private string _LastName;

public CustomerDTO(){}

public string CustomerID
{
    get { return _CustomerID; }
    set { _CustomerID = value; }
}

public string FirstName
{
    get { return _FirstName; }
    set { _FirstName = value; }
}

public string LastName
{
    get { return _LastName; }
    set { _LastName = value; }
}
}
```