XML in Visual Basic

Visual Studio 2015

Visual Basic provides integrated language support that enables it to interact with LINQ to XML.

In This Section

The topics in this section introduce using LINQ to XML with Visual Basic.

Topic	Description
Overview of LINQ to XML in Visual Basic	Describes how Visual Basic supports LINQ to XML.
Creating XML in Visual Basic	Describes how to create XML literal objects by using LINQ to XML.
Manipulating XML in Visual Basic	Describes how to load and parse XML by using Visual Basic.
Accessing XML in Visual Basic	Describes the XML axis properties and LINQ to XML methods for accessing XML elements and attributes.
XML IntelliSense in Visual Basic	Describes the IntelliSense capabilities provided with Visual Basic.

See Also

System.Xml.Linq
XML Literals (Visual Basic)
XML Axis Properties (Visual Basic)
LINQ to XML

Overview of LINQ to XML in Visual Basic

Visual Studio 2015

Visual Basic provides support for LINQ to XML through XML literals and XML axis properties. This enables you to use a familiar, convenient syntax for working with XML in your Visual Basic code. *XML literals* enable you to include XML directly in your code. *XML axis properties* enable you to access child nodes, descendant nodes, and attributes of an XML literal. For more information, see XML Literals Overview (Visual Basic) and Accessing XML in Visual Basic.

LINQ to XML is an in-memory XML programming API designed specifically to take advantage of Language-Integrated Query (LINQ). Although you can call the LINQ APIs directly, only Visual Basic enables you to declare XML literals and directly access XML axis properties.

Note

XML literals and XML axis properties are not supported in declarative code in an ASP.NET page. To use Visual Basic XML features, put your code in a code-behind page in your ASP.NET application.

For related video demonstrations, see How Do I Get Started with LINQ to XML? and How Do I Create Excel Spreadsheets using LINQ to XML?.

Creating XML

There are two ways to create XML trees in Visual Basic. You can declare an XML literal directly in code, or you can use the LINQ APIs to create the tree. Both processes enable the code to reflect the final structure of the XML tree. For example, the following code example creates an XML element:

VB

```
Dim contact1 As XElement =
    <contact>
      <name>Patrick Hines</name>
      <phone type="home">206-555-0144</phone>
      <phone type="work">425-555-0145</phone>
    </contact>
```

For more information, see Creating XML in Visual Basic.

Accessing and Navigating XML

Visual Basic provides XML axis properties for accessing and navigating XML structures. These properties enable you to access XML elements and attributes by specifying the XML child element names. Alternatively, you can explicitly call the

LINQ methods for navigating and locating elements and attributes. For example, the following code example uses XML axis properties to refer to the attributes and child elements of an XML element. The code example uses a LINQ query to retrieve child elements and output them as XML elements, effectively performing a transform.

VB

```
' Place Imports statements at the top of your program.
Imports <xmlns:ns="http://SomeNamespace">

Module Sample1

    Sub SampleTransform()

        ' Create test by using a global XML namespace prefix.

        Dim contact =
            <ns:contact>
                <ns:name>Patrick Hines</ns:name>
                <ns:phone ns:type="home">206-555-0144</ns:phone>
                <ns:phone ns:type="work">425-555-0145</ns:phone>
            </ns:contact>

        Dim phoneTypes =
          <phoneTypes>
              <%= From phone In contact.<ns:phone>
                  Select <type><%= phone.@ns:type %></type>
              %>
          </phoneTypes>

        Console.WriteLine(phoneTypes)
    End Sub

End Module
```

For more information, see Accessing XML in Visual Basic.

XML Namespaces

Visual Basic enables you to specify an alias to a global XML namespace by using the **Imports** statement. The following example shows how to use the **Imports** statement to import an XML namespace:

VB

```
Imports <xmlns:ns="http://someNamespace">
```

You can use an XML namespace alias when you access XML axis properties and declare XML literals for XML documents and elements.

You can retrieve an XNamespace object for a particular namespace prefix by using the GetXmlNamespace Operator (Visual Basic).

For more information, see Imports Statement (XML Namespace).

Using XML Namespaces in XML Literals

The following example shows how to create an XElement object that uses the global namespace `ns`:

VB

```
Dim contact1 As XElement =
    <ns:contact>
        <ns:name>Patrick Hines</ns:name>
        <ns:phone type="home">206-555-0144</ns:phone>
        <ns:phone type="work">425-555-0145</ns:phone>
    </ns:contact>

Console.WriteLine(contact1)
```

The Visual Basic compiler translates XML literals that contain XML namespace aliases into equivalent code that uses the XML notation for using XML namespaces, with the `xmlns` attribute. When compiled, the code in the previous section's example produces essentially the same executable code as the following example:

VB

```
Dim contact2 As XElement =
    <ns1:contact xmlns:ns1="http://someNamespace">
        <ns1:name>Patrick Hines</ns1:name>
        <ns1:phone type="home">206-555-0144</ns1:phone>
        <ns1:phone type="work">425-555-0145</ns1:phone>
    </ns1:contact>

Console.WriteLine(contact2)
```

Using XML Namespaces in XML Axis Properties

XML namespaces declared in XML literals are not available for use in XML axis properties. However, global namespaces can be used with the XML axis properties. Use a colon to separate the XML namespace prefix from the local element name. Following is an example:

VB

```
Console.WriteLine("Contact name is: " & contact1.<ns:name>.Value)
```

See Also

XML in Visual Basic

Creating XML in Visual Basic
Accessing XML in Visual Basic
Manipulating XML in Visual Basic

XML Axis Properties (Visual Basic)

Visual Studio 2015

The topics in this section document the syntax of XML axis properties in Visual Basic. The XML axis properties make it easy to access XML directly in your code.

In This Section

Topic	**Description**
XML Attribute Axis Property (Visual Basic)	Describes how to access the attributes of an XElement object.
XML Child Axis Property (Visual Basic)	Describes how to access the children of an XElement object.
XML Descendant Axis Property (Visual Basic)	Describes how to access the descendants of an XElement object.
Extension Indexer Property (Visual Basic)	Describes how to access individual elements in a collection of XElement or XAttribute objects.
XML Value Property (Visual Basic)	Describes how to access the value of the first element of a collection of XElement or XAttribute objects.

See Also

XML in Visual Basic

XML Attribute Axis Property (Visual Basic)

Visual Studio 2015

Provides access to the value of an attribute for an XElement object or to the first element in a collection of XElement objects.

Syntax

```
      object.@attribute
-or-
object.@<attribute>
```

Parts

object

Required. An XElement object or a collection of XElement objects.

.@

Required. Denotes the start of an attribute axis property.

<

Optional. Denotes the beginning of the name of the attribute when *attribute* is not a valid identifier in Visual Basic.

attribute

Required. Name of the attribute to access, of the form [*prefix*:]*name*.

Part	**Description**
prefix	Optional. XML namespace prefix for the attribute. Must be a global XML namespace defined with an **Imports** statement.
name	Required. Local attribute name. See Names of Declared XML Elements and Attributes (Visual Basic).

>

Optional. Denotes the end of the name of the attribute when *attribute* is not a valid identifier in Visual Basic.

Return Value

A string that contains the value of *attribute*. If the attribute name does not exist, **Nothing** is returned.

Remarks

You can use an XML attribute axis property to access the value of an attribute by name from an XElement object or from the first element in a collection of XElement objects. You can retrieve an attribute value by name, or add a new attribute to an element by specifying a new name preceded by the @ identifier.

When you refer to an XML attribute using the @ identifier, the attribute value is returned as a string and you do not need to explicitly specify the Value property.

The naming rules for XML attributes differ from the naming rules for Visual Basic identifiers. To access an XML attribute that has a name that is not a valid Visual Basic identifier, enclose the name in angle brackets (< and >).

XML Namespaces

The name in an attribute axis property can use only XML namespace prefixes declared globally by using the **Imports** statement. It cannot use XML namespace prefixes declared locally within XML element literals. For more information, see Imports Statement (XML Namespace).

Example

The following example shows how to get the values of the XML attributes named type from a collection of XML elements that are named phone.

VB

```
' Topic: XML Attribute Axis Property
Dim phones As XElement =
    <phones>
        <phone type="home">206-555-0144</phone>
        <phone type="work">425-555-0145</phone>
    </phones>

Dim phoneTypes As XElement =
  <phoneTypes>
      <%= From phone In phones.<phone>
          Select <type><%= phone.@type %></type>
      %>
  </phoneTypes>

Console.WriteLine(phoneTypes)
```

This code displays the following text:

```
<phoneTypes>

<type>home</type>

<type>work</type>

</phoneTypes>
```

Example

The following example shows how to create attributes for an XML element both declaratively, as part of the XML, and dynamically by adding an attribute to an instance of an XElement object. The `type` attribute is created declaratively and the `owner` attribute is created dynamically.

VB

```
Dim phone2 As XElement = <phone type="home">206-555-0144</phone>
phone2.@owner = "Harris, Phyllis"

Console.WriteLine(phone2)
```

This code displays the following text:

```
<phone type="home" owner="Harris, Phyllis">206-555-0144</phone>
```

Example

The following example uses the angle bracket syntax to get the value of the XML attribute named `number-type`, which is not a valid identifier in Visual Basic.

VB

```
Dim phone As XElement =
     <phone number-type=" work">425-555-0145</phone>

 Console.WriteLine("Phone type: " & phone.@<number-type>)
```

This code displays the following text:

```
Phone type: work
```

Example

The following example declares `ns` as an XML namespace prefix. It then uses the prefix of the namespace to create an XML literal and access the first child node with the qualified name "`ns:name`".

VB

```
Imports <xmlns:ns = "http://SomeNamespace">
```

```
Class TestClass3

    Shared Sub TestPrefix()
        Dim phone =
            <ns:phone ns:type="home">206-555-0144</ns:phone>

        Console.WriteLine("Phone type: " & phone.@ns:type)
    End Sub

End Class
```

This code displays the following text:

```
Phone type: home
```

See Also

- XElement
- XML Axis Properties (Visual Basic)
- XML Literals (Visual Basic)
- Creating XML in Visual Basic
- Names of Declared XML Elements and Attributes (Visual Basic)

XML Child Axis Property (Visual Basic)

Visual Studio 2015

Provides access to the children of one of the following: an XElement object, an XDocument object, a collection of XElement objects, or a collection of XDocument objects.

Syntax

```
object.<child>
```

Parts

Term	Definition
object	Required. An XElement object, an XDocument object, a collection of XElement objects, or a collection of XDocument objects.
.<	Required. Denotes the start of a child axis property.
child	Required. Name of the child nodes to access, of the form [*prefix***:**]*name*. • *Prefix* - Optional. XML namespace prefix for the child node. Must be a global XML namespace defined with an **Imports** statement. • *Name* - Required. Local child node name. See Names of Declared XML Elements and Attributes (Visual Basic).
>	Required. Denotes the end of a child axis property.

Return Value

A collection of XElement objects.

Remarks

You can use an XML child axis property to access child nodes by name from an XElement or XDocument object, or from a collection of XElement or XDocument objects. Use the XML **Value** property to access the value of the first child node in the returned collection. For more information, see XML Value Property (Visual Basic).

The Visual Basic compiler converts child axis properties to calls to the Elements method.

XML Namespaces

The name in a child axis property can use only XML namespace prefixes declared globally with the **Imports** statement. It cannot use XML namespace prefixes declared locally within XML element literals. For more information, see Imports Statement (XML Namespace).

Example

The following example shows how to access the child nodes named phone from the contact object.

VB

```
Dim contact As XElement =
    <contact>
        <name>Patrick Hines</name>
        <phone type="home">206-555-0144</phone>
        <phone type="work">425-555-0145</phone>
    </contact>

Dim homePhone = From hp In contact.<phone>
                Where contact.<phone>.@type = "home"
                Select hp

Console.WriteLine("Home Phone = {0}", homePhone(0).Value)
```

This code displays the following text:

```
Home Phone = 206-555-0144
```

Example

The following example shows how to access the child nodes named phone from the collection returned by the contact child axis property of the contacts object.

VB

```
Dim contacts As XElement =
    <contacts>
        <contact>
            <name>Patrick Hines</name>
            <phone type="home">206-555-0144</phone>
        </contact>
        <contact>
            <name>Lance Tucker</name>
            <phone type="work">425-555-0145</phone>
        </contact>
    </contacts>

Dim homePhone = From contact In contacts.<contact>
                Where contact.<phone>.@type = "home"
                Select contact.<phone>

Console.WriteLine("Home Phone = {0}", homePhone(0).Value)
```

This code displays the following text:

```
Home Phone = 206-555-0144
```

Example

The following example declares `ns` as an XML namespace prefix. It then uses the prefix of the namespace to create an XML literal and access the first child node with the qualified name `ns:name`.

VB

```
Imports <xmlns:ns = "http://SomeNamespace">

Class TestClass4

    Shared Sub TestPrefix()
        Dim contact = <ns:contact>
                          <ns:name>Patrick Hines</ns:name>
                      </ns:contact>
        Console.WriteLine(contact.<ns:name>.Value)
    End Sub

End Class
```

This code displays the following text:

```
Patrick Hines
```

See Also

XElement
XML Axis Properties (Visual Basic)
XML Literals (Visual Basic)
Creating XML in Visual Basic
Names of Declared XML Elements and Attributes (Visual Basic)

XML Descendant Axis Property (Visual Basic)

Visual Studio 2015

Provides access to the descendants of the following: an XElement object, an XDocument object, a collection of XElement objects, or a collection of XDocument objects.

Syntax

```
object...<descendant>
```

Parts

object
Required. An XElement object, an XDocument object, a collection of XElement objects, or a collection of XDocument objects.

...<
Required. Denotes the start of a descendant axis property.

descendant
Required. Name of the descendant nodes to access, of the form [*prefix*:]*name*.

Part	**Description**
prefix	Optional. XML namespace prefix for the descendant node. Must be a global XML namespace that is defined by using an **Imports** statement.
name	Required. Local name of the descendant node. See Names of Declared XML Elements and Attributes (Visual Basic).

>
Required. Denotes the end of a descendant axis property.

Return Value

A collection of XElement objects.

Remarks

You can use an XML descendant axis property to access descendant nodes by name from an XElement or XDocument object, or from a collection of XElement or XDocument objects. Use the XML **Value** property to access the value of the first descendant node in the returned collection. For more information, see XML Value Property (Visual Basic).

The Visual Basic compiler converts descendant axis properties into calls to the Descendants method.

XML Namespaces

The name in a descendant axis property can use only XML namespaces declared globally with the **Imports** statement. It cannot use XML namespaces declared locally within XML element literals. For more information, see Imports Statement (XML Namespace).

Example

The following example shows how to access the value of the first descendant node named `name` and the values of all descendant nodes named `phone` from the `contacts` object.

VB

```
Dim contacts As XElement =
    <contacts>
        <contact>
            <name>Patrick Hines</name>
            <phone type="home">206-555-0144</phone>
            <phone type="work">425-555-0145</phone>
        </contact>
    </contacts>

Console.WriteLine("Name: " & contacts...<name>.Value)

Dim homePhone = From phone In contacts...<phone>
                Select phone.Value

Console.WriteLine("Home Phone = {0}", homePhone(0))
```

This code displays the following text:

```
Name: Patrick Hines

Home Phone = 206-555-0144
```

Example

The following example declares `ns` as an XML namespace prefix. It then uses the prefix of the namespace to create an XML literal and access the value of the first child node with the qualified name `ns:name`.

VB

```vb
Imports <xmlns:ns = "http://SomeNamespace">

Class TestClass2

    Shared Sub TestPrefix()
        Dim contacts =
            <ns:contacts>
                <ns:contact>
                    <ns:name>Patrick Hines</ns:name>
                </ns:contact>
            </ns:contacts>

        Console.WriteLine("Name: " & contacts...<ns:name>.Value)
    End Sub

End Class
```

This code displays the following text:

```
Name: Patrick Hines
```

See Also

XElement
XML Axis Properties (Visual Basic)
XML Literals (Visual Basic)
Creating XML in Visual Basic
Names of Declared XML Elements and Attributes (Visual Basic)

Extension Indexer Property (Visual Basic)

Visual Studio 2015

Provides access to individual elements in a collection.

Syntax

```
object(index)
```

Parts

Term	Definition
object	Required. A queryable collection. That is, a collection that implements IEnumerable(Of T) or IQueryable(Of T).
(	Required. Denotes the start of the indexer property.
index	Required. An integer expression that specifies the zero-based position of an element of the collection.
)	Required. Denotes the end of the indexer property.

Return Value

The object from the specified location in the collection, or **Nothing** if the index is out of range.

Remarks

You can use the extension indexer property to access individual elements in a collection. This indexer property is typically used on the output of XML axis properties. The XML child and XML descendent axis properties return collections of XElement objects or an attribute value.

The Visual Basic compiler converts extension indexer properties to calls to the**ElementAtOrDefault** method. Unlike an array indexer, the**ElementAtOrDefault** method returns **Nothing** if the index is out of range. This behavior is useful when you cannot easily determine the number of elements in a collection.

This indexer property is like an extension property for collections that implement IEnumerable(Of T) or IQueryable(Of T): it is used only if the collection does not have an indexer or a default property.

To access the value of the first element in a collection of XElement or XAttribute objects, you can use the XML **Value** property. For more information, see XML Value Property (Visual Basic).

Example

The following example shows how to use the extension indexer to access the second child node in a collection of XElement objects. The collection is accessed by using the child axis property, which gets all child elements named phone in the `contact` object.

VB

```
Dim contact As XElement =
    <contact>
        <name>Patrick Hines</name>
        <phone type="home">206-555-0144</phone>
        <phone type="work">425-555-0145</phone>
    </contact>

Console.WriteLine("Second phone number: " & contact.<phone>(1).Value)
```

This code displays the following text:

```
Second phone number: 425-555-0145
```

See Also

XElement
XML Axis Properties (Visual Basic)
XML Literals (Visual Basic)
Creating XML in Visual Basic
XML Value Property (Visual Basic)

XML Value Property (Visual Basic)

Visual Studio 2015

Provides access to the value of the first element of a collection of XElement objects.

Syntax

```
object.Value
```

Parts

Term	Definition
object	Required. Collection of XElement objects.

Return Value

A **String** that contains the value of the first element of the collection, or **Nothing** if the collection is empty.

Remarks

The Value property makes it easy to access the value of the first element in a collection of XElement objects. This property first checks whether the collection contains at least one object. If the collection is empty, this property returns **Nothing**. Otherwise, this property returns the value of the Value property of the first element in the collection.

Note

When you access the value of an XML attribute using the '@' identifier, the attribute value is returned as a **String** and you do not need to explicitly specify the Value property.

To access other elements in a collection, you can use the XML extension indexer property. For more information, see

Extension Indexer Property (Visual Basic).

Inheritance

Most users will not have to implement IEnumerable(Of T), and can therefore ignore this section.

The Value property is an extension property for types that implement **IEnumerable(Of XElement)**. The binding of this extension property is like the binding of extension methods: if a type implements one of the interfaces and defines a property that has the name "Value", that property has precedence over the extension property. In other words, this Value property can be overridden by defining a new property in a class that implements **IEnumerable(Of XElement)**.

Example

The following example shows how to use the Value property to access the first node in a collection of XElement objects. The example uses the child axis property to get the collection of all child nodes named phone that are in the contact object.

VB

```
Dim contact As XElement =
    <contact>
        <name>Patrick Hines</name>
        <phone type="home">206-555-0144</phone>
        <phone type="work">425-555-0145</phone>
    </contact>

Console.WriteLine("Phone number: " & contact.<phone>.Value)
```

This code displays the following text:

```
Phone number: 206-555-0144
```

Example

The following example shows how to get the value of an XML attribute from a collection of XAttribute objects. The example uses the attribute axis property to display the value of the type attribute for all of the the phone elements.

VB

```
Dim contact As XElement =
    <contact>
      <name>Patrick Hines</name>
      <phone type="home">206-555-0144</phone>
      <phone type="work">425-555-0145</phone>
    </contact>


Dim types = contact.<phone>.Attributes("type")

For Each attr In types
  Console.WriteLine(attr.Value)
```

```
    Next
```

This code displays the following text:

```
home

work
```

See Also

- XElement
- IEnumerable(Of T)
- XML Axis Properties (Visual Basic)
- XML Literals (Visual Basic)
- Creating XML in Visual Basic
- Extension Methods (Visual Basic)
- Extension Indexer Property (Visual Basic)
- XML Child Axis Property (Visual Basic)
- XML Attribute Axis Property (Visual Basic)

Creating XML in Visual Basic

Visual Studio 2015

Visual Basic enables you to use *XML literals* directly in your code. The XML literal syntax represents LINQ to XML objects, and it is similar to the XML 1.0 syntax. This makes it easier to create XML elements, documents, and fragments programmatically because your code has the same structure as the final XML.

In This Section

Term	Definition
XML Literals Overview (Visual Basic)	Introduction to XML literals and how they relate to LINQ to XML.
Embedded Expressions in XML (Visual Basic)	Describes how to use embedded expressions in XML literals.
How to: Create XML Literals (Visual Basic)	Describes how to create an XML element in code by using an XML literal.
White Space in XML Literals (Visual Basic)	Describes how Visual Basic treats white space in XML literals.
XML Literals and the XML 1.0 Specification (Visual Basic)	Describes how the XML literal syntax in Visual Basic relates to the XML 1.0 specification.
How to: Embed Expressions in XML Literals (Visual Basic)	Describes how to use embedded expressions in XML literals to create content at run time.
Names of Declared XML Elements and Attributes (Visual Basic)	Describes guidelines for naming XML elements and attributes.

See Also

XML in Visual Basic

XML Literals Overview (Visual Basic)

Visual Studio 2015

An *XML literal* allows you to incorporate XML directly into your Visual Basic code. The XML literal syntax represents LINQ to XML objects, and it is the similar to the XML 1.0 syntax. This makes it easier to create XML elements and documents programmatically because your code has the same structure as the final XML.

Visual Basic compiles XML literals into LINQ to XML objects. LINQ to XML provides a simple object model for creating and manipulating XML, and this model integrates well with Language-Integrated Query (LINQ). For more information, see XElement.

You can embed a Visual Basic expression in an XML literal. At run time, your application creates a LINQ to XML object for each literal, incorporating the values of the embedded expressions. This lets you specify dynamic content inside an XML literal. For more information, see Embedded Expressions in XML (Visual Basic).

For more information about the differences between the XML literal syntax and the XML 1.0 syntax, see XML Literals and the XML 1.0 Specification (Visual Basic).

Simple Literals

You can create a LINQ to XML object in your Visual Basic code by typing or pasting in valid XML. An XML element literal returns an XElement object. For more information, see XML Element Literal (Visual Basic) and XML Literals and the XML 1.0 Specification (Visual Basic). The following example creates an XML element that has several child elements.

VB

```
Dim contact1 As XElement = 
    <contact>
      <name>Patrick Hines</name>
      <phone type="home">206-555-0144</phone>
      <phone type="work">425-555-0145</phone>
    </contact>
```

You can create an XML document by starting an XML literal with `<?xml version="1.0"?>`, as shown in the following example. An XML document literal returns an XDocument object. For more information, see XML Document Literal (Visual Basic).

VB

```
Dim contactDoc As XDocument = 
    <?xml version="1.0"?>
    <contact>
      <name>Patrick Hines</name>
      <phone type="home">206-555-0144</phone>
      <phone type="work">425-555-0145</phone>
    </contact>
```

Note

The XML literal syntax in Visual Basic is not identical to the syntax in the XML 1.0 specification. For more information, see XML Literals and the XML 1.0 Specification (Visual Basic).

Line Continuation

An XML literal can span multiple lines without using line continuation characters (the space-underscore-enter sequence). This makes it easier to compare XML literals in code with XML documents.

The compiler treats line continuation characters as part of an XML literal. Therefore, you should use the space-underscore-enter sequence only when it belongs in the LINQ to XML object.

However, you do need line continuation characters if you have a multiline expression in an embedded expression. For more information, see Embedded Expressions in XML (Visual Basic).

Embedding Queries in XML Literals

You can use a query in an embedded expression. When you do this, the elements returned by the query are added to the XML element. This lets you add dynamic content, such as the result of a user's query, to an XML literal.

For example, the following code uses an embedded query to create XML elements from the members of the `phoneNumbers2` array and then add those elements as children of `contact2`.

VB

```
Public Class XmlSamples

  Public Sub Main()
    ' Initialize the objects.

    Dim phoneNumbers2 As Phone() = {
        New Phone("home", "206-555-0144"),
        New Phone("work", "425-555-0145")}

    ' Convert the data contained in phoneNumbers2 to XML.

    Dim contact2 =
        <contact>
          <name>Patrick Hines</name>
          <%= From p In phoneNumbers2
            Select <phone type=<%= p.Type %>><%= p.Number %></phone>
          %>
        </contact>

    Console.WriteLine(contact2)
  End Sub
End Class

Class Phone
  Public Type As String
  Public Number As String
  Public Sub New(ByVal t As String, ByVal n As String)
    Type = t
    Number = n
  End Sub
End Class
```

How the Compiler Creates Objects from XML Literals

The Visual Basic compiler translates XML literals into calls to the equivalent LINQ to XML constructors to build up the LINQ to XML object. For example, the Visual Basic compiler will translate the following code example into a call to the XProcessingInstruction constructor for the XML version instruction, calls to the XElement constructor for the `<contact>`, `<name>`, and `<phone>` elements, and calls to the XAttribute constructor for the `type` attribute. Specifically, given the attributes in the following sample, the Visual Basic compiler will call the XAttribute(XName, Object) constructor twice. The first will pass the value `type` for the *name* parameter and the value `home` for the *value* parameter. The second will also pass the value `type` for the *name* parameter, but the value `work` for the *value* parameter.

VB

```
Dim contactDoc As XDocument =
    <?xml version="1.0"?>
    <contact>
      <name>Patrick Hines</name>
      <phone type="home">206-555-0144</phone>
      <phone type="work">425-555-0145</phone>
    </contact>
```

See Also

XElement
Creating XML in Visual Basic
Embedded Expressions in XML (Visual Basic)
XML Document Literal (Visual Basic)
XML Element Literal (Visual Basic)
XML Literals (Visual Basic)

Embedded Expressions in XML (Visual Basic)

Visual Studio 2015

Embedded expressions enable you to create XML literals that contain expressions that are evaluated at run time. The syntax for an embedded expression is **<%=** *expression* **%>**, which is the same as the syntax used in ASP.NET.

For example, you can create an XML element literal, combining embedded expressions with literal text content.

VB

```
Dim isbnNumber As String = "12345"
Dim modifiedDate As String = "3/5/2006"
Dim book As XElement =
    <book category="fiction" isbn=<%= isbnNumber %>>
        <modifiedDate><%= modifiedDate %></modifiedDate>
    </book>
```

If `isbnNumber` contains the integer 12345 and `modifiedDate` contains the date 3/5/2006, when this code executes, the value of `book` is:

```
<book category="fiction" isbn="12345">
  <modifiedDate>3/5/2006</modifiedDate>
</book>
```

Embedded Expression Location and Validation

Embedded expressions can appear only at certain locations within XML literal expressions. The expression location controls which types the expression can return and how **Nothing** is handled. The following table describes the allowed locations and types of embedded expressions.

Location in literal	Type of expression	Handling of **Nothing**
XML element name	XName	Error
XML element content	**Object** or array of **Object**	Ignored
XML element attribute name	XName	Error, unless the attribute value is also **Nothing**

XML element attribute value	**Object**	Attribute declaration ignored
XML element attribute	XAttribute or a collection of XAttribute	Ignored
XML document root element	XElement or a collection of one XElement object and an arbitrary number of XProcessingInstruction and XComment objects	Ignored

- Example of an embedded expression in an XML element name:

VB

```
Dim elementName As String = "contact"
Dim contact1 As XElement = <<%= elementName %>/>
```

- Example of an embedded expression in the content of an XML element:

VB

```
Dim contactName As String = "Patrick Hines"
Dim contact2 As XElement =
  <contact><%= contactName %></contact>
```

- Example of an embedded expression in an XML element attribute name:

VB

```
Dim phoneType As String = "home"
Dim contact3 As XElement =
  <contact <%= phoneType %>="206-555-0144"/>
```

- Example of an embedded expression in an XML element attribute value:

VB

```
Dim phoneNumber As String = "206-555-0144"
Dim contact4 As XElement =
  <contact home=<%= phoneNumber %>/>
```

- Example of an embedded expression in an XML element attribute:

VB

```
Dim phoneAttribute As XAttribute =
  New XAttribute(XName.Get(phoneType), phoneNumber)
Dim contact5 As XElement =
  <contact <%= phoneAttribute %>/>
```

- Example of an embedded expression in an XML document root element:

VB

```
Dim document As XDocument =
  <?xml version="1.0"?><%= contact1 %>
```

If you enable **Option Strict**, the compiler checks that the type of each embedded expression widens to the required type. The only exception is for the root element of an XML document, which is verified when the code runs. If you compile without **Option Strict**, you can embed expressions of type **Object** and their type is verified at run time.

In locations where content is optional, embedded expressions that contain **Nothing** are ignored. This means you do not have to check that element content, attribute values, and array elements are not **Nothing** before you use an XML literal. Required values, such as element and attribute names, cannot be **Nothing**.

For more information about using an embedded expression in a particular type of literal, see XML Document Literal (Visual Basic), XML Element Literal (Visual Basic).

Scoping Rules

The compiler converts each XML literal into a constructor call for the appropriate literal type. The literal content and embedded expressions in an XML literal are passed as arguments to the constructor. This means that all Visual Basic programming elements available to an XML literal are also available to its embedded expressions.

Within an XML literal, you can access the XML namespace prefixes declared with the **Imports** statement. You can declare a new XML namespace prefix, or shadow an existing XML namespace prefix, in an element by using the **xmlns** attribute. The new namespace is available to the child nodes of that element, but not to XML literals in embedded expressions.

Note

When you declare an XML namespace prefix by using the **xmlns** namespace attribute, the attribute value must be a constant string. In this regard, using the **xmlns** attribute is like using the **Imports** statement to declare an XML namespace. You cannot use an embedded expression to specify the XML namespace value.

See Also

Creating XML in Visual Basic
XML Document Literal (Visual Basic)
XML Element Literal (Visual Basic)
Option Strict Statement
Imports Statement (.NET Namespace and Type)
XML Literals Overview (Visual Basic)

Names of Declared XML Elements and Attributes (Visual Basic)

Visual Studio 2015

This topic provides Visual Basic guidelines for naming XML elements and attributes in XML literals. In an XML literal, you can specify a local name or a qualified name. A qualified name consists of an XML namespace prefix, a colon, and a local name. For more information about XML namespace prefixes, see XML Element Literal (Visual Basic).

Rules

A local name of an element or attribute in Visual Basic must adhere to the following rules.

- It can begin with a namespace. It must begin with an alphabetical character or an underscore (_).
- It must contain only alphabetical characters, decimal digits, underscores, periods (.), and hyphens (-).
- It must not be more than 1,024 characters long.
- Colons that appear in names indicate namespace demarcation. Therefore, you can use colons only to specify an XML namespace for a particular name.

In addition, you should adhere to the following guideline.

- The XML 1.0 specification reserves all names starting with the string "xml", of any capitalization variation. Therefore, do not use those names for your element and attribute names.

Name Length Guidelines

As a practical matter, a name should be as short as possible while still clearly identifying the nature of the element. This improves the readability of your code and reduces line length and source-file size.

However, your name should not be so short that it does not adequately describe the element or how your code uses it. This is important for the readability of your code. If somebody else is trying to understand it, or if you yourself are looking at it a long time after you wrote it, appropriate element names can save time.

Case Sensitivity in Names

XML element names are case sensitive. This means that when the Visual Basic compiler compares two names that differ in alphabetical case only, it interprets them as different names. For example, it interprets `ABC` and `abc` as referring to separate elements.

XML Namespaces

When creating an XML element literal, you can specify the XML namespace prefix for the element name. For more information, see XML Element Literal (Visual Basic).

See Also

Creating XML in Visual Basic
XML Element Literal (Visual Basic)

XML Literals and the XML 1.0 Specification (Visual Basic)

Visual Studio 2015

The XML literal syntax in Visual Basic supports most of the Extensible Markup Language (XML) 1.0 specification. For details about the XML 1.0 specification, see Extensible Markup Language (XML) 1.0 on the W3C Web site.

What Visual Basic Does Not Support

- An XML literal cannot contain a document type definition (DTD).
- An XML document literal must start with an XML document declaration.
- An XML literal cannot contain more than 65,535 characters on one line.
- XML namespace prefixes, element names, and attribute names cannot contain more than 1,024 characters.

Extra Features That Visual Basic Supports

- The embedded expression syntax allowed in document and element literals is not valid XML.

See Also

Creating XML in Visual Basic
XML Document Literal (Visual Basic)
XML Element Literal (Visual Basic)

White Space in XML Literals (Visual Basic)

Visual Studio 2015

The Visual Basic compiler incorporates only the significant white space characters from an XML literal when it creates a LINQ to XML object. The insignificant white space characters are not incorporated.

Significant and Insignificant White Space

White space characters in XML literals are significant in only three areas:

- When they are in an attribute value.
- When they are part of an element's text content and the text also contains other characters.
- When they are in an embedded expression for an element's text content.

Otherwise, the compiler treats white space characters as insignificant and does not include then in the LINQ to XML object for the literal.

To include insignificant white space in an XML literal, use an embedded expression that contains a string literal with the white space.

Note

If the **xml:space** attribute appears in an XML element literal, the Visual Basic compiler includes the attribute in the XElement object, but adding this attribute does not change how the compiler treats white space.

Examples

The following example contains two XML elements, outer and inner. Both elements contain white space in their text content. The white space in the outer element is insignificant because it contains only white space and an XML element. The white space in the inner element is significant because it contains white space and text.

VB

```
Dim example As XElement = <outer>
                              <inner>
                                  Inner text
                              </inner>
                          </outer>

Console.WriteLine(example)
```

When run, this code displays the following text.

```
<outer>
  <inner>
                                          Inner text
                                      </inner>
</outer>
```

See Also

Creating XML in Visual Basic

How to: Create XML Literals (Visual Basic)

Visual Studio 2015

You can create an XML document, fragment, or element directly in code by using an XML literal. The examples in this topic demonstrate how to create an XML element that has three child elements, and how to create an XML document.

You can also use the LINQ to XML APIs to create LINQ to XML objects. For more information, see XElement.

To create an XML element

- Create the XML inline by using the XML literal syntax, which is the same as the actual XML syntax.

VB

```
Dim contact1 As XElement =
    <contact>
      <name>Patrick Hines</name>
      <phone type="home">206-555-0144</phone>
      <phone type="work">425-555-0145</phone>
    </contact>
```

Run the code. The output of this code is:

```
<contact>

<name>Patrick Hines</name>

<phone type="home">206-555-0144</phone>

<phone type="work">425-555-0145</phone>

</contact>
```

To create an XML document

- Create the XML document inline. The following code creates an XML document that has literal syntax, an XML declaration, a processing instruction, a comment, and an element that contains another element.

VB

```
Dim libraryRequest As XDocument =
    <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <?xml-stylesheet type="text/xsl" href="show_book.xsl"?>
    <!-- Tests that the application works. -->
    <books>
        <book/>
    </books>
Console.WriteLine(libraryRequest)
```

Run the code. The output of this code is:

```
<?xml-stylesheet type="text/xsl" href="show_book.xsl"?>

<!-- Tests that the application works. -->

<books>

<book/>

</books>
```

See Also

XML in Visual Basic
Creating XML in Visual Basic
XML Element Literal (Visual Basic)
XML Document Literal (Visual Basic)

How to: Embed Expressions in XML Literals (Visual Basic)

Visual Studio 2015

You can combine XML literals with embedded expressions to create an XML document, fragment, or element that contains content created at run time. The following examples demonstrate how to use embedded expressions to populate element content, attributes, and element names at run time.

The syntax for an embedded expression is **<%=** *exp* **%>**, which is the same syntax that ASP.NET uses. For more information, see Embedded Expressions in XML (Visual Basic).

You can also use the LINQ to XML APIs to create LINQ to XML objects. For more information, see XElement.

Procedures

To insert text as element content

- The following example shows how to insert the text that is contained in the `contactName` variable between the opening and closing name elements.

VB

```
Dim contactName As String = "Patrick Hines"
Dim contact As XElement =
  <contact>
    <name><%= contactName %></name>
  </contact>
Console.WriteLine(contact)
```

This example produces the following output:

```
<contact>
  <name>Patrick Hines</name>
</contact>
```

To insert text as an attribute value

- The following example shows how to insert the text that is contained in the `phoneType` variable as the value of the `type` attribute.

VB

```
Dim phoneType As String = "home"
Dim contact2 As XElement =
  <contact>
    <phone type=<%= phoneType %>>206-555-0144</phone>
  </contact>
Console.WriteLine(contact2)
```

This example produces the following output:

```
<contact>
  <phone type="home">206-555-0144</phone>
</contact>
```

To insert text for an element name

- The following example shows how to insert the text that is contained in the `elementName` variable as the name of an element.

  When creating elements by using this technique, you must close them with the `</>` tag.

VB

```
Dim elementName As String = "contact"
Dim contact3 As XElement =
    <<%= elementName %>>
        <name>Patrick Hines</name>
    </>
Console.WriteLine(contact3)
```

This example produces the following output:

```
<contact>
  <name>Patrick Hines</name>
</contact>
```

See Also

How to: Create XML Literals (Visual Basic)
Embedded Expressions in XML (Visual Basic)
Creating XML in Visual Basic
XML in Visual Basic

Manipulating XML in Visual Basic

Visual Studio 2015

You can use *XML literals* to load XML from an external source such as a string, file, or stream. You can then use LINQ to XML to manipulate the XML and use Language-Integrated Query (LINQ) to query the XML.

In This Section

How to: Load XML from a File, String, or Stream (Visual Basic)
    Demonstrates how to load XML into an XDocument or XElement object from a text file, string, or stream.

How to: Transform XML by Using LINQ (Visual Basic)
    Demonstrates how to transform the contents of an XDocument object into a new XML document.

How to: Modify XML Literals (Visual Basic)
    Demonstrates how to modify the elements, attributes, and values in an XML literal.

Related Sections

XML Axis Properties (Visual Basic)
    Provides links to sections that describe the various XML access properties.

Overview of LINQ to XML in Visual Basic
    Provides an introduction to using LINQ to XML in Visual Basic.

Creating XML in Visual Basic
    Provides an introduction to using XML literals in Visual Basic.

Accessing XML in Visual Basic
    Demonstrates how to access parts of an XML element or document in Visual Basic.

XML in Visual Basic
    Provides links to sections that describe how to use LINQ to XML in Visual Basic.

See Also

XML in Visual Basic
LINQ in Visual Basic
Introduction to LINQ in Visual Basic

How to: Load XML from a File, String, or Stream (Visual Basic)

Visual Studio 2015

You can create XML Literals (Visual Basic) and populate them with the contents from an external source such as a file, a string, or a stream by using several methods. These methods are shown in the following examples.

Note

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see Personalizing the Visual Studio IDE.

To load XML from a file

- To populate an XML literal such as an XElement or XDocument object from a file, use the **Load** method. This method can take a file path, text stream, or XML stream as input.

  The following code example shows the use of the Load(String) method to populate an XDocument object with XML from a text file.

  VB

```
Dim books =
    XDocument.Load(My.Application.Info.DirectoryPath &
                   "\..\..\Data\books.xml")
Console.WriteLine(books)
```

To load XML from a string

- To populate an XML literal such as an XElement or XDocument object from a string, you can use the **Parse** method.

  The following code example shows the use of the XDocument.Parse(String) method to populate an XDocument object with XML from a string.

  VB

```
Dim xmlString = "<Book id=""bk102"">" & vbCrLf &
                "  <Author>Garcia, Debra</Author>" & vbCrLf &
                "  <Title>Writing Code</Title>" & vbCrLf &
                "  <Price>5.95</Price>" & vbCrLf &
                "</Book>"
```

```
Dim xmlElem = XElement.Parse(xmlString)
Console.WriteLine(xmlElem)
```

To load XML from a stream

- To populate an XML literal such as an XElement or XDocument object from a stream, you can use the **Load** method or the XNode.ReadFrom method.

The following code example shows the use of the ReadFrom method to populate an XDocument object with XML from an XML stream.

VB

```
Dim reader =
  System.Xml.XmlReader.Create(My.Application.Info.DirectoryPath &
                              "\..\..\Data\books.xml")
reader.MoveToContent()
Dim inputXml = XDocument.ReadFrom(reader)
Console.WriteLine(inputXml)
```

See Also

XDocument.Load
XElement.Load
XElement.Parse
XDocument.Parse
XNode.ReadFrom
XML Literals (Visual Basic)
XML in Visual Basic
Manipulating XML in Visual Basic

How to: Transform XML by Using LINQ (Visual Basic)

Visual Studio 2015

XML Literals (Visual Basic) make it easy to read XML from one source and transform it to a new XML format. You can take advantage of LINQ queries to retrieve the content to transform, or change content in an existing document to a new XML format.

The example in this topic transforms content from an XML source document to HTML to be viewed in a browser.

Note

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see Personalizing the Visual Studio IDE.

To transform an XML document

1. In Visual Studio, create a new Visual Basic project in the **Console Application** project template.

2. Double-click the Module1.vb file created in the project to modify the Visual Basic code. Add the following code to the `Sub Main` of the `Module1` module. This code creates the source XML document as an XDocument object.

VB

```
Dim catalog =
  <?xml version="1.0"?>
    <Catalog>
      <Book id="bk101">
        <Author>Garghentini, Davide</Author>
        <Title>XML Developer's Guide</Title>
        <Price>44.95</Price>
        <Description>
          An in-depth look at creating applications
          with <technology>XML</technology>. For
          <audience>beginners</audience> or
          <audience>advanced</audience> developers.
        </Description>
      </Book>
      <Book id="bk331">
        <Author>Spencer, Phil</Author>
        <Title>Developing Applications with Visual Basic .NET</Title>
        <Price>45.95</Price>
        <Description>
```

```
          Get the expert insights, practical code samples,
          and best practices you need
          to advance your expertise with <technology>Visual
          Basic .NET</technology>.
          Learn how to create faster, more reliable applications
          based on professional,
          pragmatic guidance by today's top <audience>developers</audience>.
        </Description>
      </Book>
    </Catalog>
```

How to: Load XML from a File, String, or Stream (Visual Basic).

3. After the code to create the source XML document, add the following code to retrieve all the <Book> elements from the object and transform them into an HTML document. The list of <Book> elements is created by using a LINQ query that returns a collection of XElement objects that contain the transformed HTML. You can use embedded expressions to put the values from the source document in the new XML format.

   The resulting HTML document is written to a file by using the Save method.

   VB

```
Dim htmlOutput =
  <html>
    <body>
      <%= From book In catalog.<Catalog>.<Book>
          Select <div>
                   <h1><%= book.<Title>.Value %></h1>
                   <h3><%= "By " & book.<Author>.Value %></h3>
                    <h3><%= "Price = " & book.<Price>.Value %></h3>
                    <h2>Description</h2>
                    <%= TransformDescription(book.<Description>(0)) %>
                    <hr/>
                  </div> %>
    </body>
  </html>

htmlOutput.Save("BookDescription.html")
```

4. After `Sub Main` of `Module1`, add a new method (**Sub**) to transform a <Description> node into the specified HTML format. This method is called by the code in the previous step and is used to preserve the format of the <Description> elements.

   This method replaces sub-elements of the <Description> element with HTML. The **ReplaceWith** method is used to preserve the location of the sub-elements. The transformed content of the <Description> element is included in an HTML paragraph (<p>) element. The Nodes property is used to retrieve the transformed content of the <Description> element. This ensures that sub-elements are included in the transformed content.

   Add the following code after `Sub Main` of `Module1`.

   VB

```
Public Function TransformDescription(ByVal desc As XElement) As XElement
```

```
    ' Replace <technology> elements with <b>.
    Dim content = (From element In desc...<technology>).ToList()

    If content.Count > 0 Then
      For i = 0 To content.Count - 1
        content(i).ReplaceWith(<b><%= content(i).Value %></b>)
      Next
    End If

    ' Replace <audience> elements with <i>.
    content = (From element In desc...<audience>).ToList()

    If content.Count > 0 Then
      For i = 0 To content.Count - 1
        content(i).ReplaceWith(<i><%= content(i).Value %></i>)
      Next
    End If

    ' Return the updated contents of the <Description> element.
    Return <p><%= desc.Nodes %></p>
  End Function
```

5. Save your changes.

6. Press F5 to run the code. The resulting saved document will resemble the following:

```
<?xml version="1.0"?>
<html>
  <body>
    <div>
      <h1>XML Developer's Guide</h1>
      <h3>By Garghentini, Davide</h3>
      <h3>Price = 44.95</h3>
      <h2>Description</h2>
      <p>
        An in-depth look at creating applications
        with <b>XML</b>. For
        <i>beginners</i> or
        <i>advanced</i> developers.
      </p>
      <hr />
    </div>
    <div>
      <h1>Developing Applications with Visual Basic .NET</h1>
      <h3>By Spencer, Phil</h3>
      <h3>Price = 45.95</h3>
      <h2>Description</h2>
      <p>
        Get the expert insights, practical code
        samples, and best practices you need
```

```
          to advance your expertise with <b>Visual
          Basic .NET</b>. Learn how to create faster,
          more reliable applications based on
          professional, pragmatic guidance by today's
          top <i>developers</i>.
        </p>
        <hr />
      </div>
    </body>
  </html>
```

See Also

XML Literals (Visual Basic)
Manipulating XML in Visual Basic
XML in Visual Basic
How to: Load XML from a File, String, or Stream (Visual Basic)
LINQ in Visual Basic
Introduction to LINQ in Visual Basic

How to: Modify XML Literals (Visual Basic)

Visual Studio 2015

Visual Basic provides convenient ways to modify XML literals. You can add or delete elements and attributes, and you can also replace an existing element with a new XML element. This topic provides several examples of how to modify an existing XML literal.

To modify the value of an XML literal

1. To modify the value of an XML literal, obtain a reference to the XML literal and set the **Value** property to the desired value.

   The following code example updates the value of all the <Price> elements in an XML document.

   VB

```
For Each book In From element In catalog.<Catalog>.<Book>
  book.<Price>.Value = (book.<Price>.Value * 1.05).ToString("#.00")
Next
```

   The following shows sample source XML and modified XML from this code example.

```
Source XML:
<?xml version="1.0"?>
<Catalog>
  <Book id="bk101">
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Price>44.95</Price>
  </Book>
  <Book id="bk331">
    <Author>Spencer, Phil</Author>
    <Title>Developing Applications with Visual Basic .NET</Title>
    <Price>45.95</Price>
  </Book>
</Catalog>

Modified XML:
<?xml version="1.0"?>
<Catalog>
  <Book id="bk101">
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Price>47.20</Price>
  </Book>
  <Book id="bk331">
    <Author>Spencer, Phil</Author>
    <Title>Developing Applications with Visual Basic .NET</Title>
    <Price>48.25</Price>
  </Book>
</Catalog>
```

Note

The **Value** property refers to the first XML element in a collection. If there is more than one element that has the same name in a collection, setting the **Value** property affects only the first element in the collection.

To add an attribute to an XML literal

1. To add an attribute to an XML literal, first obtain a reference to the XML literal. You can then add an attribute by adding a new XML attribute axis property. You can also add a new XAttribute object to the XML literal by using the Add method. The following example shows both options.

VB

```
Dim newAttribute = "editorEmail"
Dim editorID = "someone@example.com"
For Each book In From element In catalog.<Catalog>.<Book>
  ' Add an attribute by using an XML attribute axis property.
  book.@genre = "Computer"

  ' Add an attribute to the Attributes collection.
  book.Add(New XAttribute(newAttribute, editorID))
Next
```

The following shows sample source XML and modified XML from this code example.

```
Source XML:
<?xml version="1.0"?>
<Catalog>
  <Book id="bk101" >
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Price>44.95</Price>
  </Book>
  <Book id="bk331">
    <Author>Spencer, Phil</Author>
    <Title>Developing Applications with Visual Basic .NET</Title>
    <Price>45.95</Price>
  </Book>
</Catalog>
```

```
Modified XML:
<?xml version="1.0"?>
<Catalog>
  <Book id="bk101" genre="Computer" editorEmail="someone@example.com">
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Price>44.95</Price>
  </Book>
  <Book id="bk331" genre="Computer" editorEmail="someone@example.com">
    <Author>Spencer, Phil</Author>
    <Title>Developing Applications with Visual Basic .NET</Title>
    <Price>45.95</Price>
  </Book>
</Catalog>
```

For more information about XML attribute axis properties, see XML Attribute Axis Property (Visual Basic).

To add an element to an XML literal

1. To add an element to an XML literal, first obtain a reference to the XML literal. You can then add a new XElement object as the last sub-element of the element by using the Add method. You can add a new XElement object as the first sub-element by using the AddFirst method.

   To add a new element in a specific location relative to other sub-elements, first obtain a reference to an adjacent sub-element. You can then add the new XElement object before the adjacent sub-element by using the AddBeforeSelf method. You can also add the new XElement object after the adjacent sub-element by using the AddAfterSelf method.

   The following example shows examples of each of these techniques.

VB

```
Dim vbBook = From book In catalog.<Catalog>.<Book>
             Where book.<Title>.Value =
               "Developing Applications with Visual Basic .NET"

vbBook(0).AddFirst(<Publisher>Microsoft Press</Publisher>)

vbBook(0).Add(<PublishDate>2005-2-14</PublishDate>)

vbBook(0).AddAfterSelf(<Book id="bk999"></Book>)

vbBook(0).AddBeforeSelf(<Book id="bk000"></Book>)
```

The following shows sample source XML and modified XML from this code example.

```
Source XML:
<?xml version="1.0"?>
<Catalog>
```

```
  <Book id="bk101" >
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Price>44.95</Price>
  </Book>
  <Book id="bk331">
    <Author>Spencer, Phil</Author>
    <Title>Developing Applications with Visual Basic .NET</Title>
    <Price>45.95</Price>
  </Book>
</Catalog>

Modified XML:
<?xml version="1.0"?>
<Catalog>
  <Book id="bk101" >
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Price>44.95</Price>
  </Book>
  <Book id="bk000"></Book>
  <Book id="bk331">
    <Publisher>Microsoft Press</Publisher>
    <Author>Spencer, Phil</Author>
    <Title>Developing Applications with Visual Basic .NET</Title>
    <Price>45.95</Price>
    <PublishDate>2005-2-14</PublishDate>
  </Book>
  <Book id="bk999"></Book>
</Catalog>
```

To remove an element or attribute from an XML literal

1. To remove an element or an attribute from an XML literal, obtain a reference to the element or attribute and call the **Remove** method, as shown in the following example.

VB

```
For Each book In From element In catalog.<Catalog>.<Book>
  book.Attributes("genre").Remove()
Next

For Each book In From element In catalog.<Catalog>.<Book>
                 Where element.@id = "bk999"
  book.Remove()
Next
```

The following shows sample source XML and modified XML from this code example.

```
Source XML:
```

```
<?xml version="1.0"?>
<Catalog>
  <Book id="bk101" genre="Computer" editorEmail="someone@example.com">
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Price>44.95</Price>
  </Book>
  <Book id="bk000"></Book>
  <Book id="bk331" genre="Computer" editorEmail="someone@example.com">
    <Author>Spencer, Phil</Author>
    <Title>Developing Applications with Visual Basic .NET</Title>
    <Price>45.95</Price>
  </Book>
  <Book id="bk999"></Book>
</Catalog>

Modified XML:
<?xml version="1.0"?>
<Catalog>
  <Book id="bk101" editorEmail="someone@example.com">
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Price>44.95</Price>
  </Book>
  <Book id="bk000"></Book>
  <Book id="bk331" editorEmail="someone@example.com">
    <Author>Spencer, Phil</Author>
    <Title>Developing Applications with Visual Basic .NET</Title>
    <Price>45.95</Price>
  </Book>
</Catalog>
```

To remove all elements or attributes from an XML literal, obtain a reference to the XML literal and call the RemoveAll method.

To modify an XML literal

1. To change the name of an XML element, first obtain a reference to the element. You can then create a new XElement object that has a new name and pass the new XElement object to the ReplaceWith method of the existing XElement object.

   If the element that you are replacing has sub-elements that must be preserved, set the value of the new XElement object to the Nodes property of the existing element. This will set the value of the new element to the inner XML of the existing element. Otherwise, you can set the value of the new element to the **Value** property of the existing element.

   The following code example replaces all <Description> elements with an <Abstract> element. The content of the <Description> element is preserved in the new <Abstract> element by using the Nodes property of the <Description> XElement object.

   VB

```
For Each desc In From element In catalog.<Catalog>.<Book>.<Description>
  ' Replace and preserve inner XML.
  desc.ReplaceWith(<Abstract><%= desc.Nodes %></Abstract>)
Next

For Each price In From element In catalog.<Catalog>.<Book>.<Price>
  ' Replace with text value.
  price.ReplaceWith(<MSRP><%= price.Value %></MSRP>)
Next
```

The following shows sample source XML and modified XML from this code example.

```
Source XML:
<?xml version="1.0"?>
<Catalog>
  <Book id="bk101">
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <Price>44.95</Price>
    <Description>
      An in-depth look at creating applications
      with <technology>XML</technology>. For
      <audience>beginners</audience> or
      <audience>advanced</audience> developers.
    </Description>
  </Book>
  <Book id="bk331">
    <Author>Spencer, Phil</Author>
    <Title>Developing Applications with Visual Basic .NET</Title>
    <Price>45.95</Price>
    <Description>
      Get the expert insights, practical code samples, and best
      practices you need to advance your expertise with
      <technology>Visual Basic .NET</technology>.
      Learn how to create faster, more reliable applications
      based on professional, pragmatic guidance by today's top
      <audience>developers</audience>.
    </Description>
  </Book>
</Catalog>

Modified XML:
<?xml version="1.0"?>
<Catalog>
  <Book id="bk101">
    <Author>Garghentini, Davide</Author>
    <Title>XML Developer's Guide</Title>
    <MSRP>44.95</MSRP>
    <Abstract>
      An in-depth look at creating applications
```

```
        with <technology>XML</technology>. For
        <audience>beginners</audience> or
        <audience>advanced</audience> developers.
      </Abstract>
    </Book>
    <Book id="bk331">
      <Author>Spencer, Phil</Author>
      <Title>Developing Applications with Visual Basic .NET</Title>
      <MSRP>45.95</MSRP>
      <Abstract>
        Get the expert insights, practical code samples, and best
        practices you need to advance your expertise with
        <technology>Visual Basic .NET</technology>.
        Learn how to create faster, more reliable applications
        based on professional, pragmatic guidance by today's top
        <audience>developers</audience>.
      </Abstract>
    </Book>
  </Catalog>
```

See Also

Manipulating XML in Visual Basic
XML in Visual Basic
How to: Load XML from a File, String, or Stream (Visual Basic)
LINQ in Visual Basic
Introduction to LINQ in Visual Basic

Accessing XML in Visual Basic

Visual Studio 2015

Visual Basic provides XML axis properties for accessing and navigating LINQ to XML structures. These properties use a special syntax to enable you to access elements and attributes by specifying the XML names.

The following table lists the language features that enable you to access XML elements and attributes in Visual Basic.

XML Axis Properties

Property description	**Example**	**Description**
child axis	`contact.<phone>`	Gets all `phone` elements that are child elements of the `contact` element.
attribute axis	`phone.@type`	Gets all `type` attributes of the `phone` element.
descendant axis	`contacts...<name>`	Gets all `name` elements of the `contacts` element, regardless of how deep in the hierarchy they occur.
extension indexer	`contacts...<name>(0)`	Gets the first `name` element from the sequence.
value	`contacts...<name>.Value`	Gets the string representation of the first object in the sequence, or **Nothing** if the sequence is empty.

In This Section

How to: Access XML Descendant Elements (Visual Basic)
: Shows how to use a descendant axis property to access all XML elements that have a specified name and that are contained under a specified XML element.

How to: Access XML Child Elements (Visual Basic)
: Shows how to use a child axis property to access all XML child elements that have a specified name in an XML element.

How to: Access XML Attributes (Visual Basic)
: Shows how to use an attribute axis property to access all XML attributes that have a specified name in an XML element.

How to: Declare and Use XML Namespace Prefixes (Visual Basic)
: Shows how to declare an XML namespace prefix and use it to create and access XML elements.

Related Sections

XML Axis Properties (Visual Basic)
Provides links to sections describing the various XML access properties.

Overview of LINQ to XML in Visual Basic
Provides an introduction to using LINQ to XML in Visual Basic.

Creating XML in Visual Basic
Provides an introduction to using XML literals in Visual Basic.

Manipulating XML in Visual Basic
Provides links to sections about loading and modifying XML in Visual Basic.

XML in Visual Basic
Provides links to sections describing how to use LINQ to XML in Visual Basic.

How to: Access XML Descendant Elements (Visual Basic)

Visual Studio 2015

This example shows how to use a descendant axis property to access all XML elements that have a specified name and that are contained under an XML element. In particular, it uses the **Value** property to get the value of the first element in the collection that the name descendant axis property returns. The name descendant axis property gets all elements named name that are contained in the contacts object. This example also uses the phone descendant axis property to access all descendants named phone that are contained in the contacts object.

Example

VB

```
Dim contacts As XElement =
<contacts>
    <contact>
        <name>Patrick Hines</name>
        <phone type="home">206-555-0144</phone>
        <phone type="work">425-555-0145</phone>
    </contact>
</contacts>

Console.WriteLine("Name: " & contacts...<name>.Value)

Dim phoneTypes As XElement =
  <phoneTypes>
      <%= From phone In contacts...<phone>
          Select <type><%= phone.@type %></type>
      %>
  </phoneTypes>

Console.WriteLine(phoneTypes)
```

Compiling the Code

This example requires:

- A reference to the System.Xml.Linq namespace.

See Also

XContainer.Descendants

XML Descendant Axis Property (Visual Basic)
XML Value Property (Visual Basic)
Accessing XML in Visual Basic
XML in Visual Basic

How to: Access XML Child Elements (Visual Basic)

Visual Studio 2015

This example shows how to use a child axis property to access all XML child elements that have a specified name in an XML element. In particular, it uses the Value property to get the value of the first element in the collection that the name child axis property returns. The name child axis property gets all child elements named phone in the contact object. This example also uses the phone child axis property to access all child elements named phone that are contained in the contact object.

Example

VB

```
Dim contact As XElement =
<contact>
    <name>Patrick Hines</name>
    <phone type="home">206-555-0144</phone>
    <phone type="work">425-555-0145</phone>
</contact>

Console.WriteLine("Contact name: " & contact.<name>.Value)

Dim phoneTypes As XElement =
  <phoneTypes>
      <%= From phone In contact.<phone>
          Select <type><%= phone.@type %></type>
      %>
  </phoneTypes>

Console.WriteLine(phoneTypes)
```

Compiling the Code

This example requires:

- A reference to the System.Xml.Linq namespace.

See Also

XContainer.Elements
XML Child Axis Property (Visual Basic)
XML Value Property (Visual Basic)
Accessing XML in Visual Basic

XML in Visual Basic

How to: Access XML Attributes (Visual Basic)

Visual Studio 2015

This example shows how to use an attribute axis property to access XML attributes in an XML element by name. In particular, it uses the `type` attribute axis property to access the attribute named `type` in the `phone` object.

Example

VB

```
Dim phone As XElement = <phone type="home">206-555-0144</phone>

Console.WriteLine("Type: " & phone.@type)
```

See Also

XElement.Attributes
XML Attribute Axis Property (Visual Basic)
Accessing XML in Visual Basic
XML in Visual Basic

How to: Declare and Use XML Namespace Prefixes (Visual Basic)

Visual Studio 2015

This example shows how to import the XML namespace prefix ns and use it in an XML literal and XML axis properties.

Example

VB

```
' Place Imports statements at the top of your program.
Imports <xmlns:ns="http://SomeNamespace">

Module Sample1

    Sub SampleTransform()

        ' Create test by using a global XML namespace prefix.

        Dim contact =
            <ns:contact>
                <ns:name>Patrick Hines</ns:name>
                <ns:phone ns:type="home">206-555-0144</ns:phone>
                <ns:phone ns:type="work">425-555-0145</ns:phone>
            </ns:contact>

        Dim phoneTypes =
          <phoneTypes>
              <%= From phone In contact.<ns:phone>
                  Select <type><%= phone.@ns:type %></type>
              %>
          </phoneTypes>

        Console.WriteLine(phoneTypes)
    End Sub

End Module
```

Compiling the Code

This example requires:

- A reference to the System.Xml.Linq namespace.

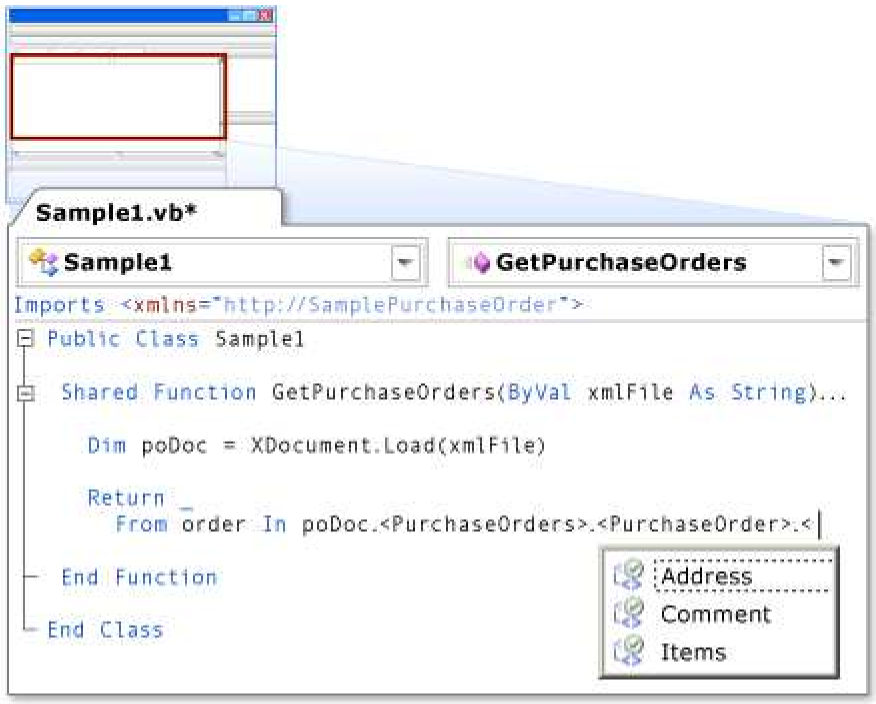
See Also

XElement.Attributes
Imports Statement (XML Namespace)
XML Attribute Axis Property (Visual Basic)
XML Value Property (Visual Basic)
Accessing XML in Visual Basic
XML in Visual Basic

XML IntelliSense in Visual Basic

Visual Studio 2015

The Visual Basic Code Editor includes IntelliSense features for XML that provide word completion for elements defined in an XML schema. If you include an XML Schema Definition (XSD) file in your project and import the target namespace of the schema by using the **Imports** statement, the Code Editor will include elements from the XSD schema in the IntelliSense list of valid member variables for XElement and XDocument objects. The following illustration shows the IntelliSense members list for an XElement object.



```
Imports <xmlns="http://SamplePurchaseOrder">
Public Class Sample1

  Shared Function GetPurchaseOrders(ByVal xmlFile As String)...

    Dim poDoc = XDocument.Load(xmlFile)

    Return _
      From order In poDoc.<PurchaseOrders>.<PurchaseOrder>.<

  End Function

End Class
```

XML IntelliSense

Enabling XML IntelliSense in Visual Basic

To enable XML IntelliSense in Visual Basic, you must include an XSD schema file in your Visual Basic project. You must also import the target namespace for the XSD schema into your code file by using the **Imports** statement. Alternatively, you can add the target namespace to the project-level namespace list by using the **References** page of the Visual Basic Project Designer. For examples, see How to: Enable XML IntelliSense in Visual Basic. For more information, see Imports Statement (XML Namespace) and References Page, Project Designer (Visual Basic).

Note that by default you cannot see XSD schema files in Visual Basic projects. You may have to click the **Show All Files** button to select an XSD file to include in your project.

Generating a Schema File (Schema Inference)

You can create an XSD schema for an existing XML file by inferring the XSD schema by using Visual Studio XML tools.

- Starting in SP1, you can use the XML to Schema Wizard to create an XML Schema set that is inferred from one or more XML documents and include it your project. You can use any combination of XML documents in the form of text files, XML from HTTP Internet addresses, or XML that is typed or pasted into the XML to Schema Wizard. To access the XML to Schema Wizard, click **Add New Item** on the **Project** menu and add an **XML to Schema** template from either the **Data** or **Common Items** template group. After you have included all the XML document sources to infer the XML Schema set from, click **OK** to create the inferred schema set. For more information, see XML to Schema Wizard (Visual Basic).

- You can also use the Visual Studio XML Editor to infer an XSD schema set from an XML file. To create an XML schema set by using the XML Editor, open an XML file in the Visual Studio XML Designer and then click **Create Schema** on the **XML** menu. After you create the XSD schema set, you can save the created schema set to one or more XSD files and include them in your project. For more information, seeHow to: Enable XML IntelliSense in Visual Basic.

Note that different XSD schema sets might be inferred from multiple XML documents that are intended to have the same schema. This can occur when particular elements and attributes are found in one XML file and not in another, or when elements are included in different order, for example. You should review inferred XSD schema sets for completeness and accuracy when you use XSD schema inference.

Member List

After you type a period (.) to delimit an instance of an XElement or XDocument object (or an instance of **IEnumerable(Of XElement)** or **IEnumerable(Of XDocument)**), Visual Basic IntelliSense displays a list of possible object members. The initial list includes three options that represent XML axis properties, as described in the following list.

Option	Description
< >	Select this option to show a list of possible child elements. For more information, see XML Element Literal (Visual Basic) and the Elements method.
@	Select this option to show a list of possible attributes. For more information, see XML Axis Properties (Visual Basic).This option is available only for objects of type XElement.
...< >	Select this option to show a list of possible descendant elements. For more information, see How to: Access XML Descendant Elements (Visual Basic) and the Elements method.

Select or begin typing any of the XML options from the list. The member list will then display potential members from the XML schema that are specific to the selected option. If you have XML namespaces imported that are associated with a specific XML namespace prefix, a list of potential XML namespace prefixes is included in the member list.

For example, consider the following XSD schema.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
           elementFormDefault="qualified"
           targetNamespace="http://SamplePurchaseOrder"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="PurchaseOrders">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PurchaseOrder">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Address" />
              <xs:element name="Items" />
              <xs:element name="Comment" />
            </xs:sequence>
            <xs:attribute name="PurchaseOrderNumber" type="xs:unsignedShort"
use="required" />
            <xs:attribute name="OrderDate" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Valid XML for the XSD schema would resemble the following.

```
<?xml version="1.0"?>
<PurchaseOrders xmlns="http://SamplePurchaseOrder">
  <PurchaseOrder PurchaseOrderNumber="12345" OrderDate="2000-1-1">
    <Address />
    <Items />
    <Comment />
  </PurchaseOrder>
</PurchaseOrders>
```

If you include this XSD schema file in a project and import the target namespace from the XSD schema into your code file or project, Visual Basic IntelliSense displays members from the schema as you type your Visual Basic code. If the target namespace for the XSD schema is imported as the default namespace and you type the following, IntelliSense displays a list of possible child elements for the **PurchaseOrder** XML element.

```
Dim po = <PurchaseOrder />
po.<
```

The list consists of the Address, Comment, and Items elements.

Certainty Levels for IntelliSense List Items

Determining the XSD type to use for IntelliSense is not exact. As a result, XML IntelliSense will often show an expanded list of possible members. To aid you in selecting an item from the IntelliSense member list, items are displayed with an indication of the level of certainty that XML IntelliSense has for a particular member.

Sometimes XML IntelliSense can identify a specific type from the XSD schema. In these cases, it will display possible child elements, attributes, or descendant elements for that XSD type with a high degree of certainty. These items are identified with a check mark.

However, sometimes XML IntelliSense is not able to identify a specific type from the XSD schema. In these cases, it will display an expanded list of possible child elements, attributes, or descendant elements from the XSD schema for the project with a low degree of certainty. These items are identified with a question mark.

See Also

How to: Enable XML IntelliSense in Visual Basic
XML to Schema Wizard (Visual Basic)
Imports Statement (XML Namespace)
XML Element Literal (Visual Basic)
XML Attribute Axis Property (Visual Basic)
XML Descendant Axis Property (Visual Basic)
References Page, Project Designer (Visual Basic)

How to: Enable XML IntelliSense in Visual Basic

Visual Studio 2015

XML IntelliSense in Visual Basic provides word completion for elements that are defined in an XML schema. To enable XML IntelliSense in Visual Basic, you must do the following:

1. Obtain the XML schema (XSD) file or files for the XML files that your application will read from or write to.
2. Include the XML schema files in your project.
3. Import the target namespace or namespaces into your code file or project. A target namespace is identified by the **targetNamespace** or **tns** attribute of your XSD schema.

   To import a target namespace, use the **Imports** statement, or add a namespace for all code files in a project by using the **References** page of the Project Designer.

For more information on the capabilities of XML IntelliSense in Visual Basic, see XML IntelliSense in Visual Basic. For more information on importing XML namespaces, see Imports Statement (XML Namespace) or References Page, Project Designer (Visual Basic).

Note

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see Personalizing the Visual Studio IDE.

For a video version of this topic, see Video How to: Enable XML IntelliSense in Visual Basic. For a related video demonstration, see How Do I Enable XML IntelliSense and Use XML Namespaces?.

Enable XML IntelliSense in Visual Basic

If you have an XML file but you do not have an XSD schema file for it, in SP1 you can create an XSD schema file by using the XML to Schema Wizard. You can also use schema inference in the Visual Studio XML Editor.

To create an XSD schema file for an XML file by using the XML to Schema Wizard (requires SP1)

1. In your project, click **Add New Item** on the **Project** menu.
2. Select the **Xml to Schema** item template from either the **Data** or **Common Items** template categories.
3. Provide a file name for the XSD file or files that the inferred schema set will be stored in, and then click **Add**.

4. In the **Infer XML Schema set from XML documents** window, add one or more XML documents to infer the XML schema set from.
    - To add text files that contain XML documents by using File Explorer, click **Add from File**.
    - To add an XML document from an HTTP address, click **Add from Web**.
    - To copy or type the contents of an XML document into the wizard, click **Type or paste XML**.
5. When you have specified all the XML document sources from which you want to infer the XML schema set, click **OK** to infer the XML schema set. The schema set is saved in your project folder in one or more XSD files. (For each XML namespace in the schema, a file is created.)

To create an XSD schema file for an XML file by using schema inference in the Visual Studio XML Editor

1. Edit the XML file in the Visual Studio XML Designer.
2. When the cursor is somewhere in the XML file, the **XML** menu appears. Click **Create Schema** on the **XML** menu. An XSD file is created from XSD schema inferred from the XML file.
3. Save the XSD schema file.

Note

Different XSD schemas might be inferred from multiple XML documents that are intended to have the same schema. This can occur when particular elements and attributes are found in one XML file and not in another, or when elements are included in different order, for example. You should review inferred XSD schemas for completeness and accuracy when you use XSD schema inference.

To include an XSD schema file

- By default, you cannot see XSD files in Visual Basic projects. If your XSD file is already included in the folders for your project, click the **Show All Files** button in **Solution Explorer**. Locate the XSD file in **Solution Explorer**, right-click the file, and click **Include File in Project**.
- If your XSD file is not already part of your project, in **Solution Explorer**, right-click the folder in which you want to store the XSD file, point to **Add**, and then click **Existing Item**. Locate your XSD file and then click **Add**.

To import an XML namespace in a code file

1. Identify the target namespace from your XSD schema.
2. At the beginning of the code file, add an **Imports** statement for the target XML namespace, as shown in the following example.

VB

```
Imports <xmlns:ns="http://someNamespace">
```

To import an XML namespace as the default namespace, that is, the namespace that is applied to XML elements and attributes that do not have a namespace prefix, add an **Imports** statement for the target default XML namespace. Do not specify a namespace prefix. Following is an example of an **Imports** statement.

VB

```
Imports <xmlns="http://defaultNamespace">
```

To import an XML namespace for all files in a project

1. An XML namespace imported in a code file applies to that code file only. To import an XML namespace that applies to all code files in a project, open the Project Designer by double-clicking **My Project** in **Solution Explorer**.

2. On the **References** tab, in the **Imported namespaces** box, type the target XML namespace in the form of a full XML namespace declaration (for example, `<xmlns: ns="http://sampleNamespace">`). If the target XML namespace does not specify a namespace prefix, the namespace will be the default XML namespace for the project.

3. Click **Add User Import**.

See Also

Imports Statement (XML Namespace)
References Page, Project Designer (Visual Basic)
XML IntelliSense in Visual Basic

XML to Schema Wizard (Visual Basic)

Visual Studio 2015

Use the XML to Schema Wizard to create an XML schema set that is inferred from one or more XML documents and include it your project. You can use any combination of XML documents in the form of text files, XML from HTTP Internet addresses, or XML that is typed or pasted into the XML to Schema Wizard.

XML schemas are used to provide IntelliSense for XML properties in Visual Basic. For more information, see XML in Visual Basic and XML IntelliSense in Visual Basic.

Note

Before you run the XML to Schema Wizard, it is recommended that you remove any existing XSD files from the project that were previously generated by the wizard. If you infer an XML schema set that matches an existing schema set, a conflict can occur and Visual Basic will not be able to provide IntelliSense for XML properties.

The XML to Schema Wizard uses the XmlSchemaInference class to create the schema for the supplied XML. As a result, multiple schema files may be created for the schema set. For each XML namespace in the supplied XML, an Extensible Schema Definition (XSD) file is created. For more information, see the InferSchema method.

To access the XML to Schema Wizard, click **Add New Item** on the **Project** menu and add an **XML to Schema** template from either the **Data** or **Common Items** template group. After you have included all the XML document sources to infer the XML schema set from, click **OK** to create the inferred schema set.

Source Type
This column displays the type of the XML document source: **File**, **URL**, or **XML**.

XML Document Location
This column displays the path of the XML document. For typed or pasted XML documents, displays the contents of the XML document.

Add from File
Click this button to add XML document files by using File Explorer.

Add from Web
Click this button to supply the HTTP address of an XML document.

Type or paste XML
Click this button to type or paste an XML document into the dialog box.

See Also

XmlSchemaInference
XML IntelliSense in Visual Basic