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)

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

XML Literals (Visual Basic)

Visual Studio 2015

The topics in this section document the syntax of XML literals in Visual Basic. The XML literal syntax enables you to incorporate XML directly in your code.

In This Section

Topic	Description
XML Element Literal (Visual Basic)	Describes the syntax for literals that represent XElement objects.
XML Document Literal (Visual Basic)	Describes the syntax for literals that represent XDocument objects.
XML CDATA Literal (Visual Basic)	Describes the syntax for literals that represent XCData objects.
XML Comment Literal (Visual Basic)	Describes the syntax for literals that represent XComment objects.
XML Processing Instruction Literal (Visual Basic)	Describes the syntax for literals that represent XProcessingInstruction objects.

See Also

XML in Visual Basic

XML Element Literal (Visual Basic)

Visual Studio 2015

A literal that represents an XElement object.

Syntax

```
<name [ attributeList ] />
-or-
<name [ attributeList ] > [ elementContents ] </[ name ]>
```

Parts

- **<**

  Required. Opens the starting element tag.

- *name*

  Required. Name of the element. The format is one of the following:

  - Literal text for the element name, of the form [*ePrefix***:**]*eName*, where:

Part	Description
ePrefix	Optional. XML namespace prefix for the element. Must be a global XML namespace that is defined with an **Imports** statement in the file or at the project level, or a local XML namespace that is defined in this element or a parent element.
eName	Required. Name of the element. The format is one of the following: ■ Literal text. See Names of Declared XML Elements and Attributes (Visual Basic). ■ Embedded expression of the form **<%=** e*NameExp* **%>**. The type of *eNameExp* must be **String** or a type that is implicitly convertible to XName.

  - Embedded expression of the form **<%=** *nameExp* **%>**. The type of *nameExp* must be **String** or a type implicitly convertible to XName. An embedded expression is not allowed in a closing tag of an element.

- *attributeList*

Optional. List of attributes declared in the literal.

```
attribute [ attribute ... ]
```

Each *attribute* has one of the following syntaxes:

- Attribute assignment, of the form [*aPrefix*:]*aName*=*aValue*, where:

Part	Description
aPrefix	Optional. XML namespace prefix for the attribute. Must be a global XML namespace that is defined with an **Imports** statement, or a local XML namespace that is defined in this element or a parent element.
aName	Required. Name of the attribute. The format is one of the following: ■ Literal text. See Names of Declared XML Elements and Attributes (Visual Basic). ■ Embedded expression of the form **<%=** *aNameExp* **%>**. The type of *aNameExp* must be **String** or a type that is implicitly convertible to XName.
aValue	Optional. Value of the attribute. The format is one of the following: ■ Literal text, enclosed in quotation marks. ■ Embedded expression of the form **<%=** *aValueExp* **%>**. Any type is allowed.

- Embedded expression of the form **<%=** *aExp* **%>**.

- **/>**

  Optional. Indicates that the element is an empty element, without content.

- **>**

  Required. Ends the beginning or empty element tag.

- *elementContents*

  Optional. Content of the element.

  ```
  content [ content ... ]
  ```

  Each *content* can be one of the following:

  - Literal text. All the white space in *elementContents* becomes significant if there is any literal text.
  - Embedded expression of the form **<%=** *contentExp* **%>**.
  - XML element literal.
  - XML comment literal. See XML Comment Literal (Visual Basic).
  - XML processing instruction literal. See XML Processing Instruction Literal (Visual Basic).

    - XML CDATA literal. See XML CDATA Literal (Visual Basic).

- **</**[*name*]**>**

  Optional. Represents the closing tag for the element. The optional *name* parameter is not allowed when it is the result of an embedded expression.

Return Value

An XElement object.

Remarks

You can use the XML element literal syntax to create XElement objects in your code.

> **Note**
>
> An XML literal can span multiple lines without using line continuation characters. This feature enables you to copy content from an XML document and paste it directly into a Visual Basic program.

Embedded expressions of the form **<%=** *exp* **%>** enable you to add dynamic information to an XML element literal. For more information, see Embedded Expressions in XML (Visual Basic).

The Visual Basic compiler converts the XML element literal into calls to the XElement constructor and, if it is required, the XAttribute constructor.

XML Namespaces

XML namespace prefixes are useful when you have to create XML literals with elements from the same namespace many times in code. You can use global XML namespace prefixes, which you define by using the **Imports** statement, or local prefixes, which you define by using the **xmlns:***xmlPrefix*="*xmlNamespace*" attribute syntax. For more information, see Imports Statement (XML Namespace).

In accordance with the scoping rules for XML namespaces, local prefixes take precedence over global prefixes. However, if an XML literal defines an XML namespace, that namespace is not available to expressions that appear in an embedded expression. The embedded expression can access only the global XML namespace.

The Visual Basic compiler converts each global XML namespace that is used by an XML literal into a one local namespace definition in the generated code. Global XML namespaces that are not used do not appear in the generated code.

Example

The following example shows how to create a simple XML element that has two nested empty elements.

VB

```
Dim test1 As XElement =
<outer>
    <inner1></inner1>
    <inner2/>
</outer>

Console.WriteLine(test1)
```

The example displays the following text. Notice that the literal preserves the structure of the empty elements.

```
<outer>
  <inner1></inner1>
  <inner2 />
</outer>
```

Example

The following example shows how to use embedded expressions to name an element and create attributes.

VB

```
Dim elementType = "book"
Dim authorName = "My Author"
Dim attributeName1 = "year"
Dim attributeValue1 = 1999
Dim attributeName2 = "title"
Dim attributeValue2 = "My Book"

Dim book As XElement =
<<%= elementType %>
    isbn="1234"
    author=<%= authorName %>
    <%= attributeName1 %>=<%= attributeValue1 %>
    <%= New XAttribute(attributeName2, attributeValue2) %>
/>

Console.WriteLine(book)
```

This code displays the following text:

```
<book isbn="1234" author="My Author" year="1999" title="My Book" />
```

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 displays the element's final form.

VB

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

Class TestClass1

    Shared Sub TestPrefix()
        ' Create test using a global XML namespace prefix.
        Dim inner2 = <ns:inner2/>

        Dim test =
        <ns:outer>
            <ns:middle xmlns:ns="http://NewNamespace">
                <ns:inner1/>
                <%= inner2 %>
            </ns:middle>
        </ns:outer>

        ' Display test to see its final form.
        Console.WriteLine(test)
    End Sub

End Class
```

This code displays the following text:

```
<ns:outer xmlns:ns="http://SomeNamespace">
  <ns:middle xmlns:ns="http://NewNamespace">
    <ns:inner1 />
    <inner2 xmlns="http://SomeNamespace" />
  </ns:middle>
</ns:outer>
```

Notice that the compiler converted the prefix of the global XML namespace into a prefix definition for the XML namespace. The <ns:middle> element redefines the XML namespace prefix for the <ns:inner1> element. However, the <ns:inner2> element uses the namespace defined by the **Imports** statement.

See Also

XElement
Names of Declared XML Elements and Attributes (Visual Basic)
XML Comment Literal (Visual Basic)

XML CDATA Literal (Visual Basic)
XML Literals (Visual Basic)
Creating XML in Visual Basic
Embedded Expressions in XML (Visual Basic)
Imports Statement (XML Namespace)

XML Document Literal (Visual Basic)

Visual Studio 2015

A literal representing an XDocument object.

Syntax

```
<?xml version="1.0" [encoding="encoding"] [standalone="standalone"] ?>
[ piCommentList ]
rootElement
[ piCommentList ]
```

Parts

Term	Definition
encoding	Optional. Literal text declaring which encoding the document uses.
standalone	Optional. Literal text. Must be "yes" or "no".
piCommentList	Optional. List of XML processing instructions and XML comments. Takes the following format: `piComment [ piComment ... ]` Each *piComment* can be one of the following: • XML Processing Instruction Literal (Visual Basic). • XML Comment Literal (Visual Basic).

rootElement	Required. Root element of the document. The format is one of the following:
	• XML Element Literal (Visual Basic).
	• Embedded expression of the form **<%=** *elementExp* **%>**. The *elementExp* returns one of the following:
	  ○ An XElement object.
	  ○ A collection that contains one XElement object and any number of XProcessingInstruction and XComment objects.

	For more information, see Embedded Expressions in XML (Visual Basic).

Return Value

An XDocument object.

Remarks

An XML document literal is identified by the XML declaration at the start of the literal. Although each XML document literal must have exactly one root XML element, it can have any number of XML processing instructions and XML comments.

An XML document literal cannot appear in an XML element.

Note

An XML literal can span multiple lines without using line continuation characters. This enables you to copy content from an XML document and paste it directly into a Visual Basic program.

The Visual Basic compiler converts the XML document literal into calls to the XDocument and XDeclaration constructors.

Example

The following example creates an XML document that has 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)
```

See Also

XElement
XProcessingInstruction
XComment

- XDocument
- XML Processing Instruction Literal (Visual Basic)
- XML Comment Literal (Visual Basic)
- XML Element Literal (Visual Basic)
- XML Literals (Visual Basic)
- Creating XML in Visual Basic
- Embedded Expressions in XML (Visual Basic)

XML CDATA Literal (Visual Basic)

Visual Studio 2015

A literal representing an XCData object.

Syntax

```
<![CDATA[content]]>
```

Parts

<![CDATA[
Required. Denotes the start of the XML CDATA section.

content
Required. Text content to appear in the XML CDATA section.

]]>
Required. Denotes the end of the section.

Return Value

An XCData object.

Remarks

XML CDATA sections contain raw text that should be included, but not parsed, with the XML that contains it. A XML CDATA section can contain any text. This includes reserved XML characters. The XML CDATA section ends with the sequence "]]>". This implies the following points:

- You cannot use an embedded expression in an XML CDATA literal because the embedded expression delimiters are valid XML CDATA content.
- XML CDATA sections cannot be nested, because *content* cannot contain the value "]]>".

You can assign an XML CDATA literal to a variable, or include it in an XML element literal.

> **Note**
>
> An XML literal can span multiple lines but does not use line continuation characters. This enables you to copy content from an XML document and paste it directly into a Visual Basic program.

The Visual Basic compiler converts the XML CDATA literal to a call to the XCData constructor.

Example

The following example creates a CDATA section that contains the text "Can contain literal <XML> tags".

VB

```
Dim cdata As XCData = <![CDATA[Can contain literal <XML> tags]]>
```

See Also

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

XML Comment Literal (Visual Basic)

Visual Studio 2015

A literal representing an XComment object.

Syntax

```
<!-- content -->
```

Parts

Term	Definition
<!--	Required. Denotes the start of the XML comment.
content	Required. Text to appear in the XML comment. Cannot contain a series of two hyphens (--) or end with a hyphen adjacent to the closing tag.
-->	Required. Denotes the end of the XML comment.

Return Value

An XComment object.

Remarks

XML comment literals do not contain document content; they contain information about the document. The XML comment section ends with the sequence "-->". This implies the following points:

- You cannot use an embedded expression in an XML comment literal because the embedded expression delimiters are valid XML comment content.
- XML comment sections cannot be nested, because *content* cannot contain the value "-->".

You can assign an XML comment literal to a variable, or you can include it in an XML element literal.

> **Note**
>
> An XML literal can span multiple lines without using line continuation characters. This feature enables you to copy content from an XML document and paste it directly into a Visual Basic program.

The Visual Basic compiler converts the XML comment literal to a call to the XComment constructor.

Example

The following example creates an XML comment that contains the text "This is a comment".

VB

```
Dim com As XComment = <!-- This is a comment -->
```

See Also

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

XML Processing Instruction Literal (Visual Basic)

Visual Studio 2015

A literal representing an XProcessingInstruction object.

Syntax

```
<?piName [ = piData ] ?>
```

Parts

<?
Required. Denotes the start of the XML processing instruction literal.

piName
Required. Name indicating which application the processing instruction targets. Cannot begin with "xml" or "XML".

piData
Optional. String indicating how the application targeted by *piName* should process the XML document.

?>
Required. Denotes the end of the processing instruction.

Return Value

An XProcessingInstruction object.

Remarks

XML processing instruction literals indicate how applications should process an XML document. When an application loads an XML document, the application can check the XML processing instructions to determine how to process the document. The application interprets the meaning of *piName* and *piData*.

The XML document literal uses syntax that is similar to that of the XML processing instruction. For more information, see

XML Document Literal (Visual Basic).

> **Note**
>
> The *piName* element cannot begin with the strings "xml" or "XML", because the XML 1.0 specification reserves those identifiers.

You can assign an XML processing instruction literal to a variable or include it in an XML document literal.

> **Note**
>
> An XML literal can span multiple lines without needing line continuation characters. This enables you to copy content from an XML document and paste it directly into a Visual Basic program.

The Visual Basic compiler converts the XML processing instruction literal to a call to the XProcessingInstruction constructor.

Example

The following example creates a processing instruction identifying a style-sheet for an XML document.

VB

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

See Also

XProcessingInstruction
XML Document Literal (Visual Basic)
XML Literals (Visual Basic)
Creating 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

```
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)