XML files

Data for three products

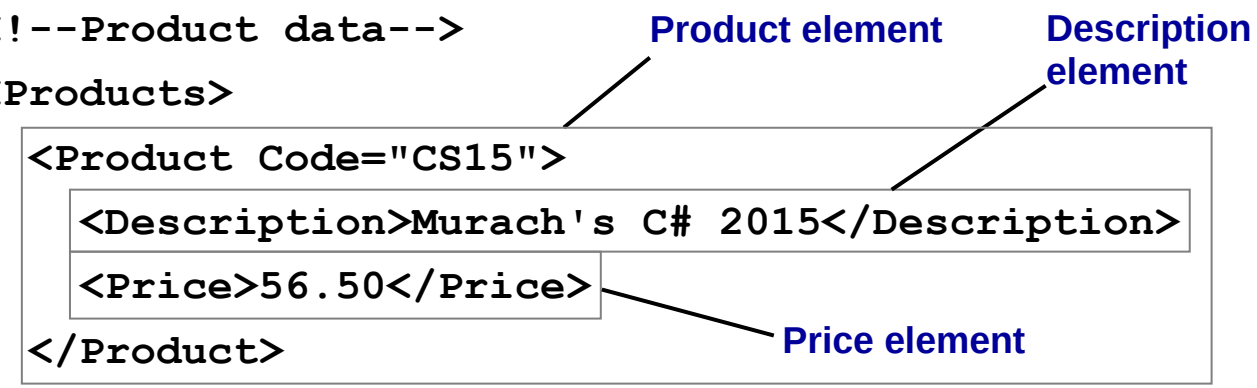
Code	Description	Price
A5CS	Murach's ASP.NET 4.5 with C# 2012	57.50
BJWN	Murach's Beginning Java with NetBeans	57.50
CS15	Murach's C# 2015	56.50

An XML document that contains the data for the three products

```
<?xml version="1.0" encoding="utf-8"?>
<!--Product data-->
<Products>
  <Product Code="A5CS">
    <Description>Murach's ASP.NET 4.5 with C# 2012
    </Description>
    <Price>57.50</Price>
  </Product>
  <Product Code="BJWN">
    <Description>Murach's Beginning Java with NetBeans
    </Description>
    <Price>57.50</Price>
  </Product>
  <Product Code="CS15">
    <Description>Murach's C# 2015</Description>
    <Price>56.50</Price>
  </Product>
</Products>
```

An XML document

```
<?xml version="1.0" encoding="utf-8"?>
<!--Product data-->
<Products>
  <Product Code="CS15">
    <Description>Murach's C# 2015</Description>
    <Price>56.50</Price>
  </Product>
</Products>
```



Tags, XML declarations, and comments

- Each XML tag begins with < and ends with >.
- The first line in an XML document is an *XML declaration* that indicates which version of the XML standard is being used for the document.
- In addition, the declaration usually identifies the standard character set that's being used. For documents in English-speaking countries, UTF-8 is the character set that's commonly used.
- You can use the <!-- and --> tags to include comments in an XML document.

Elements

- An *element* is a unit of XML data that begins with a *start tag* and ends with an *end tag*.
  - The start tag provides the name of the element and contains any attributes assigned to the element.
  - The end tag repeats the name, prefixed with a slash (/).
  - The text between an element's start and end tags is called the element's *content*.
  - You can use any name you want for an XML element.
- Elements can contain other elements. An element that's contained within another element is known as a *child element*. The element that contains a child element is known as the child's *parent element*.
- Child elements can repeat within a parent element.
- The highest-level parent element in an XML document is known as the *root element*. An XML document can have only one root element.

An XML document

```
<?xml version="1.0" encoding="utf-8"?>
<!--Product data-->
<Products>
  <Product Code="CS15">
    <Description>Murach's C# 2015</Description>
    <Price>56.50</Price>
  </Product>
</Products>
```

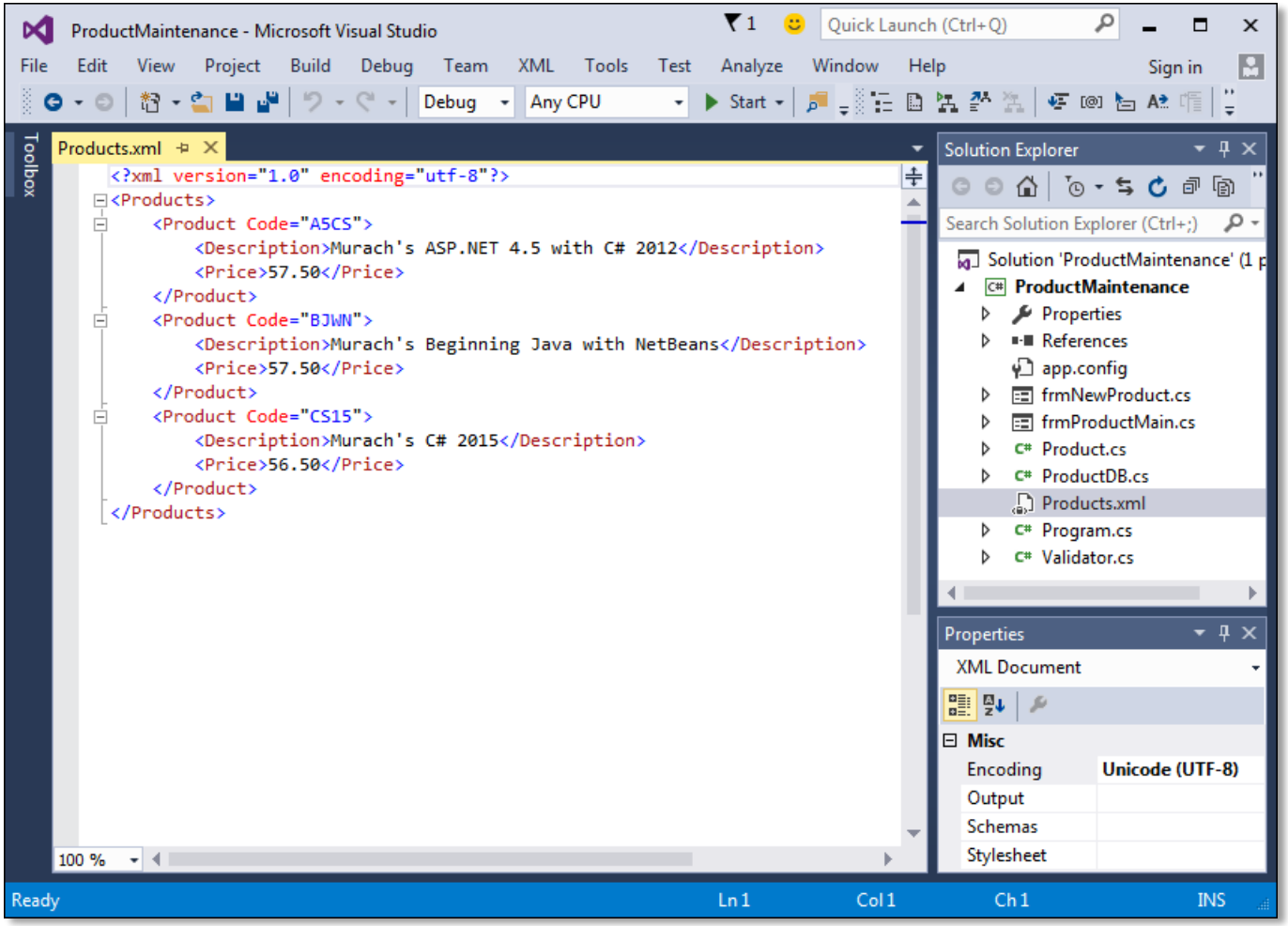
Attributes

- You can include one or more *attributes* in the start tag for an element.
- An attribute consists of an attribute name, an equals sign, and a string value in quotes.
- If an element has more than one attribute, the order in which the attributes appear doesn't matter, but the attributes must be separated by one or more spaces.

When to use attributes instead of child elements

- When you design an XML document, you can use either child elements or attributes to represent the data for an element. The choice of one over the other is often a matter of preference.
- Two advantages of attributes are that they can appear in any order and they are more concise because they do not require end tags.
- Two advantages of child elements are that they are easier for people to read and they are more convenient for long string values.

An XML document in the XML Editor



How to add a new XML file to a project

- Choose the Project→Add New Item command. In the Add New Item dialog box, select the XML File template; type a name for the XML file in the Name text box; and click Add.
- This adds an XML file to your project, adds the XML declaration for the document, and opens the document in the XML Editor.

How to open an existing XML file

- To open an existing XML file that's stored in the project, double-click on the file.
- To open an existing XML file without adding the file to your project, use the File→Open→File command.

How to edit an XML file

- When you type a start tag, the XML Editor automatically adds an end tag for the element and positions the insertion point between the start and end tags so you can type the element's content.
- When you type an attribute and an equals sign, the XML Editor automatically adds a pair of quotation marks and positions the insertion point between them so you can type the attribute value.

Common methods of the XmlWriter class

```
Create(path)
Create(path, settings)
WriteStartDocument()
WriteComment(comment)
WriteStartElement(elementName)
WriteAttributeString(attributeName, value)
WriteEndElement()
WriteElementString(elementName, content)
Close()
```

Common properties of the XmlWriterSettings class

```
Indent
IndentChars
```

Code that writes an XML document

```
// create the path variable
string path = @"C:\C# 2015\Files\Products.xml";

// create the XmlWriterSettings object
XmlWriterSettings settings = new XmlWriterSettings();
settings.Indent = true;
settings.IndentChars = ("    ");

// create the XmlWriter object
XmlWriter xmlOut = XmlWriter.Create(path, settings);

// write the start of the document
xmlOut.WriteStartDocument();
xmlOut.WriteStartElement("Products");
```

Code that writes an XML document (cont.)

```
// write each Product object to the xml file
foreach (Product product in products)
{
    xmlOut.WriteStartElement("Product");
    xmlOut.WriteAttributeString("Code",
        product.Code);
    xmlOut.WriteElementString("Description",
        product.Description);
    xmlOut.WriteElementString("Price",
        Convert.ToString(product.Price));
    xmlOut.WriteEndElement();
}

// write the end tag for the root element
xmlOut.WriteEndElement();

// close the XmlWriter object
xmlOut.Close();
```

Common indexer of the XmlReader class

`[name]`

Common properties of the XmlReader class

NodeType

Name

Value

EOF

Common methods of the XmlReader class

- `Create(path)`
- `Create(path, settings)`
- `Read()`
- `ReadStartElement(name)`
- `ReadEndElement()`
- `ReadToDescendant(name)`
- `ReadToNextSibling(name)`
- `ReadElementContentAsString()`
- `ReadElementContentAsDecimal()`
- `Close()`

Common properties of the XmlReaderSettings class

- `IgnoreWhitespace`
- `IgnoreComments`

An XML document

```
<?xml version="1.0" encoding="utf-8"?>
<!--Product data-->
<Products>
  <Product Code="CS15">
    <Description>Murach's C# 2015</Description>
    <Price>56.50</Price>
  </Product>
</Products>
```

The XML nodes in this document

NodeType	Name	Other properties
XmlDeclaration	xml	
Comment		Value = “Product data”
Element	Products	
Element	Product	[“Code”] = “CS15”
Element	Description	
Text		Value = “Murach’s C# 2015”
EndElement	Description	
Element	Price	
Text		Value = “56.50”
EndElement	Price	
EndElement	Product	
EndElement	Products	

Code that reads an XML document

```
// create the list
List<Product> products = new List<Product>();

// create the path variable
string path = @"C:\C# 2015\Files\Products.xml";

// create the XmlReaderSettings object
XmlReaderSettings settings = new XmlReaderSettings();
settings.IgnoreWhitespace = true;
settings.IgnoreComments = true;

// create the XmlReader object
XmlReader xmlIn = XmlReader.Create(path, settings);
```

Code that reads an XML document (cont.)

```
// read past all nodes to the first Product node
if (xmlIn.ReadToDescendant("Product"))
{
    // create one Product object for each Product node
    do
    {
        Product product = new Product();
        product.Code = xmlIn["Code"];
        xmlIn.ReadStartElement("Product");
        product.Description =
            xmlIn.ReadElementContentAsString();
        product.Price =
            xmlIn.ReadElementContentAsDecimal();
        products.Add(product);
    }
    while(xmlIn.ReadToNextSibling("Product"));
}

// close the XmlReader object
xmlIn.Close();
```

A class that works with an XML document

```
using System;
using System.Xml;
using System.Collections.Generic;

namespace ProductMaintenance
{
    public class ProductDB
    {
        private const string path =
            @"C:\C# 2015\Files\Products.xml";

        public static List<Product> GetProducts()
        {
            // create the list
            List<Product> products = new List<Product>();

            // create the XmlReaderSettings object
            XmlReaderSettings settings = new XmlReaderSettings();
            settings.IgnoreWhitespace = true;
            settings.IgnoreComments = true;
```

A class that works with an XML document (cont.)

```
// create the XmlReader object
XmlReader xmlIn = XmlReader.Create(path, settings);

// read past all nodes to the first Product node
if (xmlIn.ReadToDescendant("Product"))
{
    // create one Product object for each Product node
    do
    {
        Product product = new Product();
        product.Code = xmlIn["Code"];
        xmlIn.ReadStartElement("Product");
        product.Description =
            xmlIn.ReadElementContentAsString();
        product.Price =
            xmlIn.ReadElementContentAsDecimal();
        products.Add(product);
    }
    while(xmlIn.ReadToNextSibling("Product"));
}
```

A class that works with an XML document (cont.)

```
        // close the XmlReader object
        xmlIn.Close();

        return products;
    }
    public static void SaveProducts(List<Product> products)
    {
        // create the XmlWriterSettings object
        XmlWriterSettings settings = new XmlWriterSettings();
        settings.Indent = true;
        settings.IndentChars = ("    ");

        // create the XmlWriter object
        XmlWriter xmlOut = XmlWriter.Create(path, settings);

        // write the start of the document
        xmlOut.WriteStartDocument();
        xmlOut.WriteStartElement("Products");
```

A class that works with an XML document (cont.)

```
            // write each Product object to the xml file
            foreach (Product product in products)
            {
                xmlOut.WriteStartElement("Product");
                xmlOut.WriteAttributeString("Code",
                    product.Code);
                xmlOut.WriteElementString("Description",
                    product.Description);
                xmlOut.WriteElementString("Price",
                    Convert.ToString(product.Price));
                xmlOut.WriteEndElement();
            }

            // write the end tag for the root element
            xmlOut.WriteEndElement();

            // close the XmlWriter object
            xmlOut.Close();
        }
    }
}
```