XAML

eXtensible Application Markup Language

Agenda

- XML Essentials
- XAML Essentials

What Is XAML?

- **It is an XML-based language**
  - **for creating trees of .NET objects.**
- What is it used for?
  - XAML is it used to build **WPF** user interfaces.
- Although XAML is strongly associated with **WPF**, the two are separate.
  - You do not have to use XAML in order to write a **WPF** application,
  - and it is possible to use XAML for technologies other than **WPF**.
- What is the syntax of XAML?
  - Well – that's what this presentation is all about.

XML ESSENTIALS

What Is XML?

- **XML** (The **Extensible Markup Language**) is a general-purpose specification for creating custom markup languages.
  - It is classified as an extensible language because it allows its users to define their own elements.
  - Its primary purpose is to facilitate the sharing of structured data across different information systems, and it is used both to encode documents and to serialize data.
  - XML is a generic framework for storing any amount of text or any data whose structure can be represented as a tree.

Well-formed documents: XML syntax

- A XML document has exactly one **root element**
  - This means that the text must be enclosed between a root start-tag and a corresponding end-tag.
  - The root element can be preceded by an optional **XML declaration**.

- The following is a "well-formed" XML document:

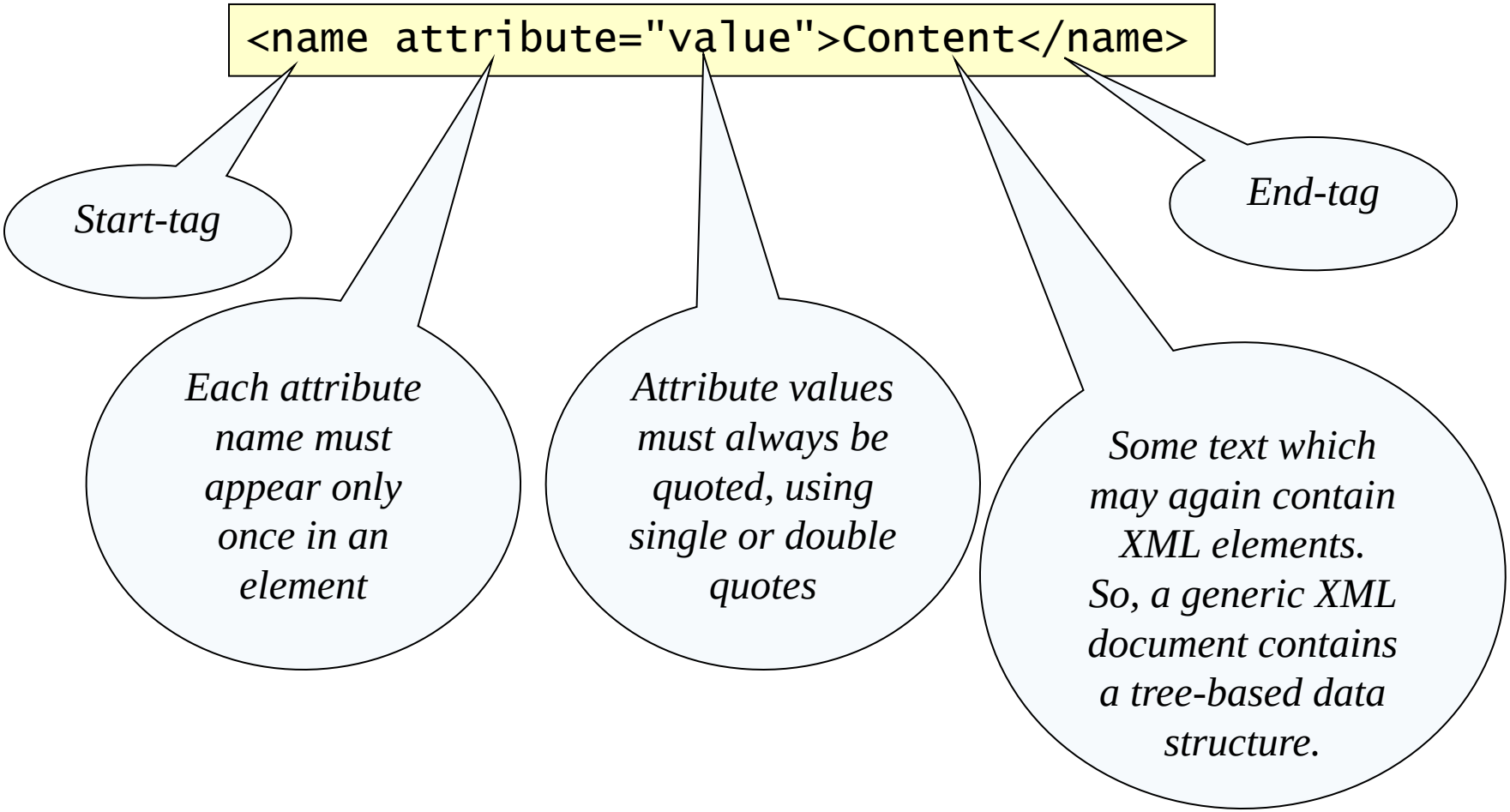
```
<book>This is a book.... </book>
```

- **Comments** can be placed anywhere outside of a tag

```
<!-- This is a comment. -->
```

The basic syntax

- The basic syntax for one **element** is:



Special Syntax For Empty Content

- XML provides special syntax for representing an element with empty content.
- The following three examples are equivalent in XML:

```
<foo></foo>
<foo />
<foo/>
```

- An empty-element may contain attributes:

```
<info author="John Smith"
      genre="science-fiction"
      date="2009-Jan-01" />
```

Not Well-formed XML

- XML requires that elements are properly nested
  - **elements may never overlap!**

```
<!-- WRONG! NOT WELL-FORMED XML! -->
<title>Book on Logic<author>Aristotle<title>Another Book on Logic
<author>Boole</title></author></title></author>

<!-- Correct: Well-formed XML. -->
<title>Book on Logic</title> <author>Aristotle</author>
<title>Another Book on Logic <author>Boole</author> </title>
```

XAML ESSENTIALS

```
<Window
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/p
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xam
   x:Class="XamlProj.Window1"
   Title="Main Window">
   <Grid>
      <Ellipse Fill="LightBlue" />
      <TextBlock> Name:
         <TextBlock Text="{Binding Name}" />
      </TextBlock>
   </Grid>
</Window>
```

Namespaces

```
<Window
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   ...
```

- XAML relies on XML namespaces to determine the meaning of elements.
- Many class names are ambiguous.
  - There are three different classes called Control in the .NET class library.
  - The .NET Framework has a namespace system that is used for disambiguation.
  - Standard XML also has a namespace system that is used for the same purpose.
  - XAML uses XML namespaces to represent .NET namespaces.
  - **But there is not a one-to-one correspondence between XML namespaces in XAML and .NET namespaces!**

Mapping Namespaces

- One XML namespace can encompass several .NET namespaces.
  - **`XmlnsDefinitionAttribute`** is used to define the relationship between a XML namespace and one or more .NET namespaces.
  - you apply the **`XmlnsDefinitionAttribute`** to the assembly that contains the types you would like to make accessible in XAML.
  - you can apply the attribute several times in order to add multiple .NET namespaces to a single XML namespace,
    - or to define multiple XML namespaces.

```
[assembly:XmlnsDefinition("http://example.com/mywpftypes",
                          "MyNamespace")]
[assembly:XmlnsDefinition("http://example.com/mywpftypes",
                          "MyNamespace.NestedNamespace")]
[assembly:XmlnsDefinition("http://example.com/otherwpftypes",
                          "MyOtherNamespace ")]
   ...
```

XmlnsDefinitionAttribute

```
[assembly:XmlnsDefinition("http://example.com/mywpftypes",
                          "MyNamespace")]
[assembly:XmlnsDefinition("http://example.com/mywpftypes",
                          "MyNamespace.NestedNamespace")]
[assembly:XmlnsDefinition("http://example.com/otherwpftypes",
                          "MyOtherNamespace ")]
   ...
```

Xml namespace

.Net (C#) namespace

3 .Net namespaces is mapped to

MyNamespace
MyNamespace.NestedNamespace

MyOtherNamespace

2 xml namespaces

xample.com/mywpftypes

example.com/otherwpftypes

Standard XAML Namespaces

```
<Window
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   ...
```

- The first indicates types that are part of the **WPF** framework.
  - This particular XML namespace encompasses several .NET namespaces, including System.Windows and many of its children.
  - There is no colon after the xmlns tag which make this namespace the default namespace for this element (everything between <Window> and </Window>.
- The second namespace represents various XAML utility features not specific to **WPF**,
  - such as the ability to represent type objects, or a null reference.
  - This is a special namespace, in that not everything in it corresponds to a type.
  - The second namespace is associated with the x prefix.

Namespace mapping URI syntax

- XAML also supports a way to refer to types in namespaces for which the XmlnsDefinitionAttribute has not been used.

```
<Grid xmlns:local="clr-namespace:MyProject"
      xmlns:mylib="clr-namespace:MyLibraryNS;assembly=MyLibrary">

  <!-- MyProject.MyLocalType in local assembly -->
  <local:MyLocalType />

  <!-- MyLibraryNamespace.MyLibraryType in MyLibrary assembly -->
  <mylib:MyLibraryType />
</Grid>
```

- If an XML namespace URI begins with clr-namespace: it will not be treated as a simple opaque identifier, as namespace URIs normally are
  - the XAML compiler will parse the URI to extract the .NET namespace, and optionally an assembly name.

Generating Classes

```
<Window
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentat
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   x:Class="XamlProj.Window1"
   Title="Main Window">
   ...
</Window>
```

- The x:Class attribute is a signal to the XAML compiler that it should generate a class definition based on this XAML file.
- The x:Class attribute determines the name of the generated class, and it will derive from the type of the root element.
- You do not have to specify an x:Class attribute.
  - If we were to omit the attribute from this example, the root object's type would be Window, rather than the generated Window1 class.

Properties

```
<Window
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentat
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   x:Class="XamlProj.Window1"
   Title="Main Window">
   ...
</Window>
```

- This attribute has no namespace qualifier. In XAML, unqualified attributes usually correspond to properties on the .NET object to which the element refers.
- The Title attribute indicates that when an instance of this generated XamlProj.Window1 class is constructed, it should set its own Title property to "Main Window".
- This is equivalent to the following code:

```
this.Title = "Main Window";
```

Children

```
<Window
    ... >
    <Grid Name="g">
        <Ellipse Fill="LightBlue" />
        <TextBlock> Name:
            <TextBlock Text="{Binding Name}" />
        </TextBlock>
    </Grid>
</Window>
```

- Whenever you attempt to provide nested content, the XAML compiler will require the parent type (Window, in this case) or its base class to be annotated with the ContentPropertyAttribute.
- This attribute tells the XAML compiler the name of the property that will contain the child content.
- In this example, the compiler will arrange for a Grid object to be created and assigned to the Content property of the Window.

```
Grid g = new Grid();
myWindow.Content = g;
```

Handling multiple children

- Elements that can contain multiple children, such as panels, simply designate a property with a collection type as the content property:

```
[ContentProperty("Children"), ...]
public class Panel : FrameworkElement {
    ...
    public UIElementCollection Children
{ get { ... } }
}
```

```
<Window
   ...
   <Grid Name="g">
      <Ellipse Fill="Li
      <TextBlock> Name:
         <TextBlock Tex
      </TextBlock>
   </Grid>
</Window>
```

- Panel is the base class of Grid, so this tells us how the XAML compiler will add the children inside the Grid in our example:

```
Ellipse e = new Ellipse(  );
TextBlock t = new TextBlock(  );
...
g.Children.Add(e);
g.Children.Add(t);
```

References

- XML overview:
  http://en.wikipedia.org/wiki/XML
- XML definition
  http://www.w3.org/XML/Core/#Publications
- XAML overview
  http://en.wikipedia.org/wiki/Extensible_Application_Markup_Language
- XAML definition
  http://msdn2.microsoft.com/en-us/library/ms747122.aspx