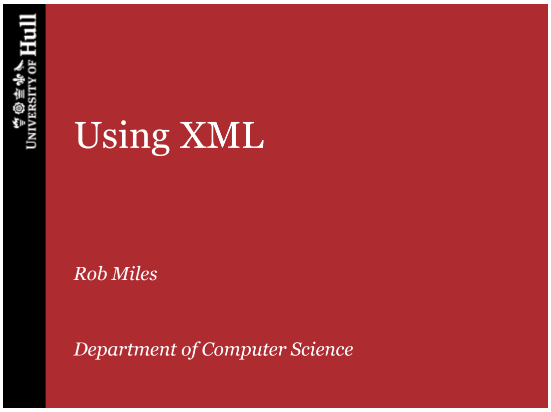
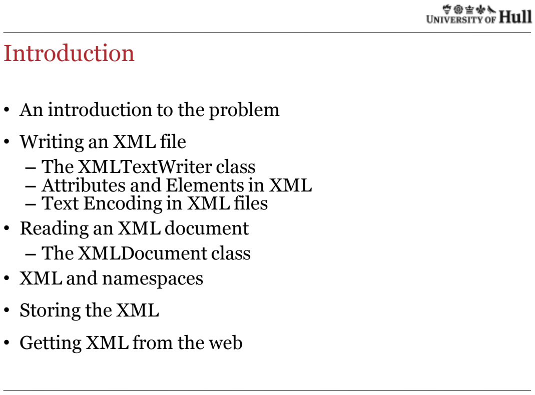


Using XML

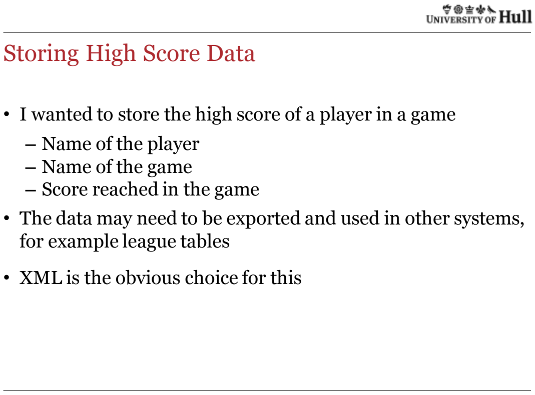
Rob Miles

Department of Computer Science



Introduction

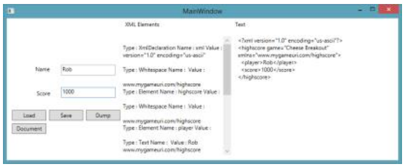
- An introduction to the problem
- Writing an XML file
  - The XMLTextWriter class
  - Attributes and Elements in XML
  - Text Encoding in XML files
- Reading an XML document
  - The XMLDocument class
- XML and namespaces
- Storing the XML
- Getting XML from the web



Storing High Score Data

- I wanted to store the high score of a player in a game
  - Name of the player
  - Name of the game
  - Score reached in the game
- The data may need to be exported and used in other systems, for example league tables
- XML is the obvious choice for this

Writing a Demo Version



Name: Rob

Score: 1000

```
Type : XmlDeclaration Name : xml Value :
version="1.0" encoding="us-ascii"

Type : Whitespace Name :  Value :

www.mygameuri.com/highscore
Type : Element Name : highscore Value :

Type : Whitespace Name :  Value :

www.mygameuri.com/highscore
Type : Element Name : player Value :

Type : Text Name :  Value : Rob
www.mygameuri.com/highscore
```

```
<?xml version="1.0" encoding="us-ascii"?>
<highscore game="Cheese Breakout"
xmlns="www.mygameuri.com/highscore">
  <player>Rob</player>
  <score>1000</score>
</highscore>
```

- Robert’s Rule 1:
- “Make a nice place to work”
- I’ve made a program that lets me play with XML storage
- It has a simple WPF interface

XML Namespaces

```
using System.Xml;
```

- To get direct access to the XML methods and classes I have to use the XML namespace:
- Once I have these I can write a method to save the values in an XML document

Writing an XML document

```
public void SaveXML ( string filename )
{
  XmlTextWriter writer ;
  writer = new XmlTextWriter( filename, Encoding.ASCII);
  writer.Formatting = Formatting.Indented ;
  writer.WriteStartDocument();
  writer.WriteStartElement("highscore");
  writer.WriteEndElement();
  writer.WriteEndDocument();
  writer.Close();
}
```

- This method will create a document with an empty highscore element
- The document is placed in the filename supplied to the call

Empty XML Document

```
<?xml version="1.0" encoding="us-ascii"?>
<highscore />
```

- The header of the document simply describes the version of xml and the encoding
- The score element is shown as empty
- This is a completely legal XML document
  - but it does not contain any data.

XML Attributes and Elements

- There are two types of data in an XML file
  - Element: a lump of data about something; may contain other elements
  - Attribute: used to further describe a particular element.
- The document being created presently has one element, called highscore.
- I can add an attribute to the highscore element which identifies the game that was being played

Adding an Attribute

```
public void SaveXML ( string filename )
{
    XmlTextWriter writer ;
    writer = new XmlTextWriter (filename,Encoding.ASCII);
    writer.Formatting = Formatting.Indented;
    writer.WriteStartDocument();
    writer.WriteStartElement("highscore");
    writer.WriteAttributeString( "game", "Breakout");
    writer.WriteEndElement();
    writer.WriteEndDocument();
    writer.Close();
}
```

Elements and Attribute Output

```
<?xml version="1.0" encoding="us-ascii"?>
<highscore game="Breakout" />
```

- The game attribute identifies the name of the game for which the high score was reached
- This attribute is attributed to a given highscore element

Adding the Player and Score

- Now we need to add the data about the player and the score reached
- There are two ways to do this:
  - add two more attributes to the highscore element. These would be called player and score and would hold the required values.
  - add two new elements, player and score inside the highscore element

Elements vs. Attributes

- I have decided that player and score should be elements rather than attributes
- It is easier for me to extend the player and score storage;
  - I could add the address of the player and the date and time the score was achieved
  - Those attributes should bind to the player and score items, not the highscore itself
- Information directly about the high score data, such as the game it applies to, should be an attribute
- Another use for an attribute would be as an id tag of an element, or perhaps a version number (which you can see in the header of the XML file itself)

Writing the High Score

```
XmlTextWriter writer;
writer = new XmlTextWriter( filename,Encoding.ASCII) ;
writer.Formatting = Formatting.Indented;
writer.WriteStartDocument();
 writer.WriteStartElement("highscore");
  writer.WriteAttributeString( "game", "Breakout");
  writer.WriteElementString("playername",playerName);
  writer.WriteElementString("score",score.ToString());
 writer.WriteEndElement();
writer.WriteEndDocument();
writer.Close();
```

- This code builds the document
- I've used indenting to make it a bit clearer

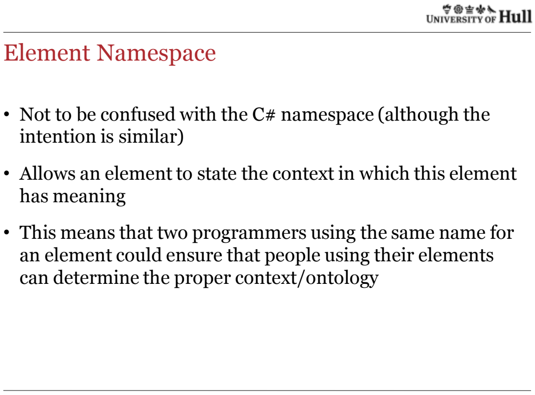
High Score XML

```
<?xml version="1.0" encoding="us-ascii" ?>
<highscore game="Breakout">
  <playername>Rob Miles</playername>
  <score>1500</score>
</highscore>
```

- This is the XML produced by the previous code
- This can be read by any program that understands XML
- It is also quite easy for humans to understand

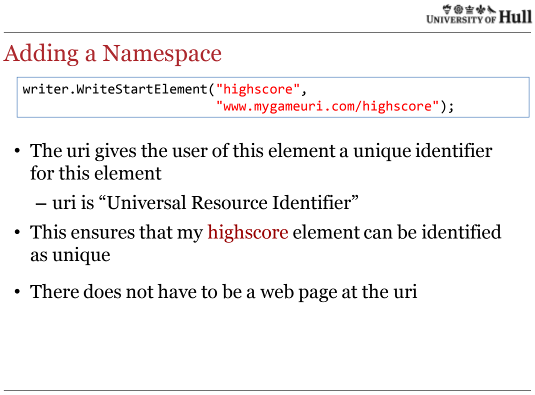
XML & Meanings

- Before we read the XML it is important to have discussion about the meaning of things
- The program that we write will ascribe meaning to the elements it gets:
  - A score which is a big value is "good"
  - In golf this would not be true.....
- There is nothing in the XML which gives the meaning of the data itself



Element Namespace

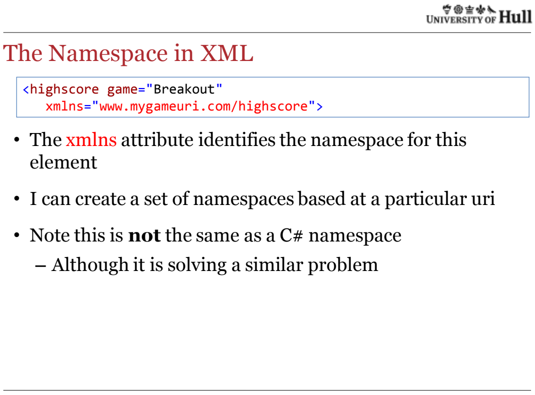
- Not to be confused with the C# namespace (although the intention is similar)
- Allows an element to state the context in which this element has meaning
- This means that two programmers using the same name for an element could ensure that people using their elements can determine the proper context/ontology



Adding a Namespace

```
writer.WriteStartElement("highscore",
                         "www.mygameuri.com/highscore");
```

- The uri gives the user of this element a unique identifier for this element
  - uri is "Universal Resource Identifier"
- This ensures that my highscore element can be identified as unique
- There does not have to be a web page at the uri



The Namespace in XML

```
<highscore game="Breakout"
   xmlns="www.mygameuri.com/highscore">
```

- The xmlns attribute identifies the namespace for this element
- I can create a set of namespaces based at a particular uri
- Note this is **not** the same as a C# namespace
  - Although it is solving a similar problem

Data in XML

```
writer.WriteElementString("player",playerName);
```

```
<player>Rob Miles</player>
```

- You write elements out by using the `WriteElementString` method
- This is given the name of the element and the data payload
- Data is always written as text

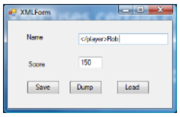
Writing Numbers

```
writer.WriteElementString("score",score.ToString());
```

```
<score>150</score>
```

- To write a number you need to convert it into a string
- When you read the number back you will have to parse it back into a value

Data and Escape Sequences



Name </player>Rob

Score 150

- XML uses certain characters to mark the start and end of items in the data file
  - These are called *delimiters*
- This could lead to problems if the user puts these characters into the data the program is storing

XML Character Escaping

```
<?xml version="1.0" encoding="us-ascii"?>
<highscore game="Cheese Breakout"
xmlns="www.mygameuri.com/highscore">
  <player>&lt;/player&gt;Rob</player>
  <score>150</score>
</highscore>
```

- When the XML writer saves a text element it will automatically convert dangerous characters into escape sequences
- This means that with XML Writer a user can't type a name that will upset the parser
  - If you create XML "by hand" you should remember this

Storing Raw Data

```
<?xml version="1.0" encoding="us-ascii"?>
<highscore game="Cheese Breakout" xmlns="www.mygameuri.com/highscore">
  <player><![CDATA[Very long and complicated name]]></player>
  <score>0</score>
</highscore>
```

- If you want so send large amounts of text which include lots of escape characters you can use the CDATA element in your XML
- This tells the XML parser not to look for XML content until it sees the sequence that marks the end of the CDATA element.

CDATA Danger

- If you store what users type in as CDATA this can lead to problems
  - A naughty user could type]]> into the data and then add other elements that they are not supposed to
  - This is a standard form of attack for web sites, particularly those powered by SQL
- http://xkcd.com/327/

Reading an XML document

```
public void DumpXml(string filename)
{
    XmlTextReader reader = new XmlTextReader(filename);

    while (reader.Read())
    {
         Console.WriteLine(
            "Type : " + reader.NodeType.ToString() +
            " Name : " + reader.Name +
            " Value : " + reader.Value);
    }
    reader.Close();
}
```

- You can create an XMLTextReader to read in nodes from an XML document
- The above method just dumps the document

XML nodes

```
Type : XmlDeclaration Name : xml Value : version="1.0"
encoding="us-ascii"
Type : Whitespace Name :  Value :
Type : Element Name : highscore Value :
Type : Whitespace Name :  Value :
Type : Element Name : player Value :
Type : Text Name :  Value : Rob Miles
Type : EndElement Name : player Value :
Type : Whitespace Name :  Value :
Type : Element Name : score Value :
Type : Text Name :  Value : 1500
Type : EndElement Name : score Value :
Type : Whitespace Name :  Value :
Type : EndElement Name : highscore Value :
```

- You could write a read method that unpicks the nodes and pulls the data from the values of the appropriate ones
- But there is a better way to do this

The XMLDocument Class

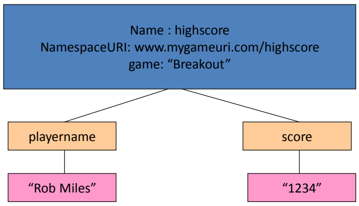
- You can create an instance of the XMLDocument class that holds all the information in our high score document
- You can then read the information you require from properties that the document exposes
- This is much easier than working through individual XML elements

Creating the XMLDocument value

```
XmlDocument document = null;
// get a new document
document = new XmlDocument();
// load it from a file
document.Load(filename);
```

- The above code creates a document instance which is based on the XML held in the given filename
- If it doesn't like the document format it will throw an exception

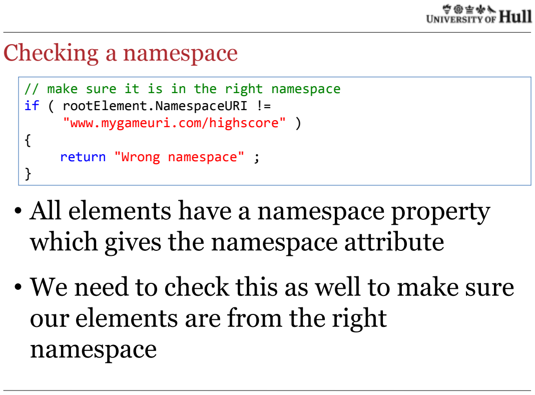
XmlDocument structure



Getting the Root element

```
 System.Xml.XmlElement rootElement =
                                  document.DocumentElement;
// make sure it is the right element
if ( rootElement.Name != "highscore" )
{
    return "Not highscore data";
}
```

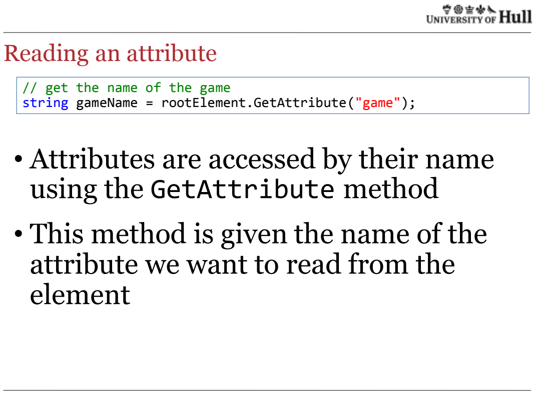
- This gets the root element for the document and makes sure it is the right one
- All elements expose a Name property that can be used to identify them



Checking a namespace

```
// make sure it is in the right namespace
if ( rootElement.NamespaceURI !=
     "www.mygameuri.com/highscore" )
{
    return "Wrong namespace" ;
}
```

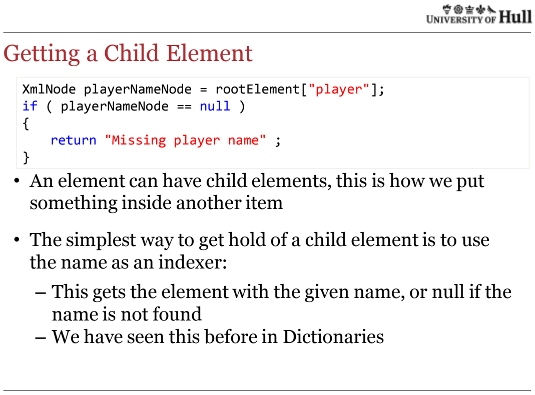
- All elements have a namespace property which gives the namespace attribute
- We need to check this as well to make sure our elements are from the right namespace



Reading an attribute

```
// get the name of the game
string gameName = rootElement.GetAttribute("game");
```

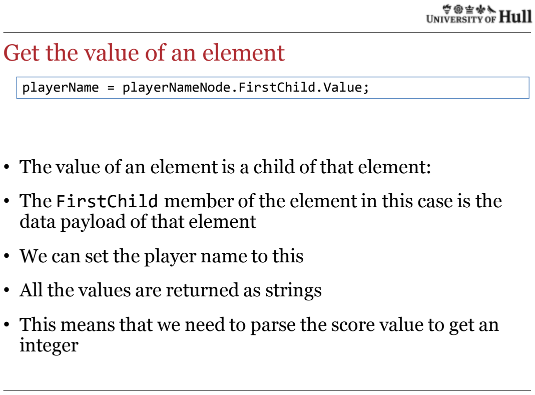
- Attributes are accessed by their name using the `GetAttribute` method
- This method is given the name of the attribute we want to read from the element



Getting a Child Element

```
XmlNode playerNameNode = rootElement["player"];
if ( playerNameNode == null )
{
    return "Missing player name" ;
}
```

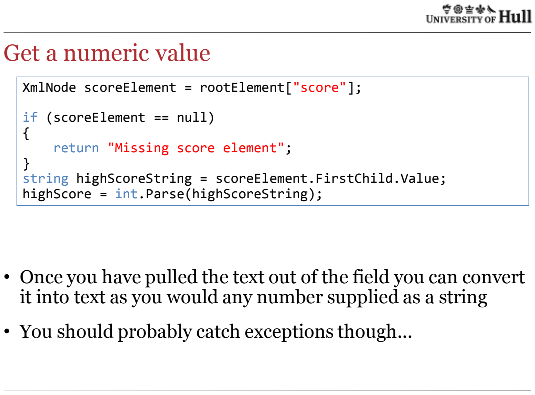
- An element can have child elements, this is how we put something inside another item
- The simplest way to get hold of a child element is to use the name as an indexer:
  - This gets the element with the given name, or null if the name is not found
  - We have seen this before in Dictionaries



Get the value of an element

```
playerName = playerNameNode.FirstChild.Value;
```

- The value of an element is a child of that element:
- The `FirstChild` member of the element in this case is the data payload of that element
- We can set the player name to this
- All the values are returned as strings
- This means that we need to parse the score value to get an integer

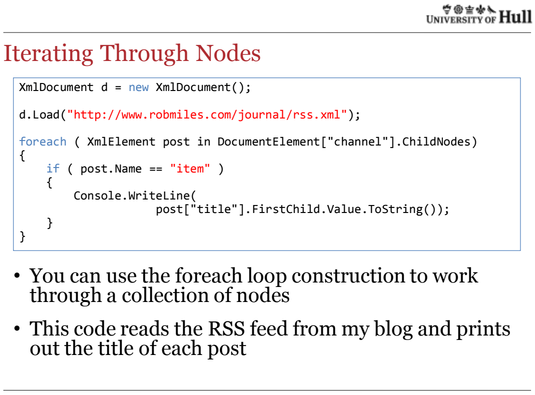


Get a numeric value

```
XmlNode scoreElement = rootElement["score"];

if (scoreElement == null)
{
    return "Missing score element";
}
string highScoreString = scoreElement.FirstChild.Value;
highScore = int.Parse(highScoreString);
```

- Once you have pulled the text out of the field you can convert it into text as you would any number supplied as a string
- You should probably catch exceptions though…



Iterating Through Nodes

```
XmlDocument d = new XmlDocument();

d.Load("http://www.robmiles.com/journal/rss.xml");

foreach ( XmlElement post in DocumentElement["channel"].ChildNodes)
{
    if ( post.Name == "item" )
    {
        Console.WriteLine(
                    post["title"].FirstChild.Value.ToString());
    }
}
```

- You can use the foreach loop construction to work through a collection of nodes
- This code reads the RSS feed from my blog and prints out the title of each post

Setting Values

- You can set values in an element as well
- There is also a method call which will save an element (and all of it's children)
- This can be used if you want to update values
- You can call the Save method on the document to save it to a file

XML is Fun!

- No, really…..
- It provides a great way to manage program data in a flexible and extensible manner
  - For very little effort on your part
- Whenever you are storing program data, and you aren't putting it in a database, you should put it in XML!
  - And it is very easy to write programs that consume XML formatted data from the internet