C# XmlReader

XmlReader opens and parses XML files. It handles attribute values, text nodes and multiple tags names. It provides a lower-level abstraction over the XML file structure. This is more complex than other solutions but benefits performance.

Example

Conceptually, the XmlReader provides a forward-only parsing object for the underlying XML files. In other words, you must manage the position in the file logically in your code as the parser can only go forward.

You can use certain methods, such as IsStartElement and the Name property, to detect the location in the file and then execute conditional logic based on this location. This increases complexity.

However:
XmlReader reduces the memory space required for the parser.
It is efficient.

Program that uses XmlReader type: C#

```
using System;
using System.Xml;

class Program
{
    static void Main()
    {
        // Create an XML reader for this file.
        using (XmlReader reader = XmlReader.Create("perls.xml"))
        {
            while (reader.Read())
            {
                // Only detect start elements.
                if (reader.IsStartElement())
                {
                    // Get element name and switch on it.
                    switch (reader.Name)
                    {
                        case "perls":
                            // Detect this element.
                            Console.WriteLine("Start <perls> element.");
```

```
                            break;
                        case "article":
                            // Detect this article element.
                            Console.WriteLine("Start <article> element.");
                            // Search for the attribute name on this current node.
                            string attribute = reader["name"];
                            if (attribute != null)
                            {
                                Console.WriteLine("  Has attribute name: " + attribute);
                            }
                            // Next read will contain text.
                            if (reader.Read())
                            {
                                Console.WriteLine("  Text node: " + reader.Value.Trim());
                            }
                            break;
                    }
                }
            }
        }
    }
}
```

Input text: perls.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<perls>
    <article name="backgroundworker">
        Example text.
    </article>
    <article name="threadpool">
        More text.
    </article>
    <article></article>
    <article>Final text.</article>
</perls>
```

Output

```
Start <perls> element.
Start <article> element.
  Has attribute name: backgroundworker
  Text node: Example text.
Start <article> element.
  Has attribute name: threadpool
  Text node: More text.
Start <article> element.
  Text node:
Start <article> element.
  Text node: Final text.
```

Create method. One of the simplest ways to instantiate an XmlReader instance is to assign an XmlReader reference to the result of the XmlReader.Create static method. This throws an exception if the file is not found.

Next:
You can read forward through the nodes in the file, which deals with the physical file data itself.

While reader Read(). The program next shows a useful

pattern of reading XML files, which is a while-loop construct that evaluates the result of the Read instance method in its expression body.

Note:
The loop will always terminate if there is nothing more to read, so you will not have to handle the end-of-file (EOF) manually.

Tip:
This pattern is also used with the StreamReader type in C# programs. More information on StreamReader is available.
While
StreamReader

IsStartElement

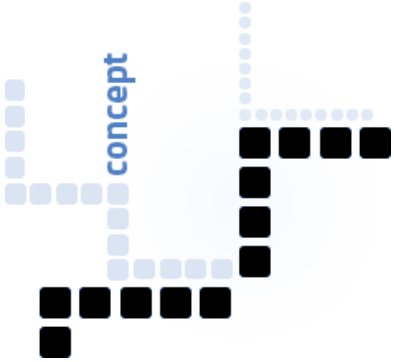
The program shows IsStartElement. This returns true if the element is not an end tag. This means it will return true if you encounter "<article>" but false if you encounter "</article>".

When the article start tag is detected, we proceed with further logic to parse that part of the file. The attribute "name" is accessed with the "name" argument in an indexer. This is null when not found.

And:
The enclosed text node in the element <article> is parsed as a separate node, so we call Read() a second time to get it.

Benefits

Let's review the benefits of XmlReader as opposed to some solutions such as the XElement type. Unfortunately, the XmlReader will introduce more complexity into the parsing code in your program, due to its nature as a forward-only parser.



And:

XmlReader will not parse an entire file into an object model automatically.

Types such as XElement can do this, as with the XElement.Load method. But they can greatly expand memory usage and reduce performance by forcing unnecessary disk IO and allocations.

XElement
Optimizations

Summary

We looked at a simple example of the XmlReader type in the C# language. This class can be used to implement higher-level parsing code, while retaining top performance and low memory usage.

Thus:
The XmlReader retains more complexity. With it you can manipulate the XML at a level more suited to many programs.