```
<?xml version="1.0" encoding="utf-8"?>
<books>
  <book status="In Progress">
     <author>Jonathan Hartwell</author>
      <title>C# And XML: A Primer</title>
  </book>
</books>
```

C# And XML: A Primer

jon@dontbreakthebuild.com

This book is for sale at http://leanpub.com/candxmlaprimer

This version was published on 2013-10-10

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help jon@dontbreakthebuild.com by spreading the word about this book on Twitter!

The suggested tweet for this book is:

Check out C# And XML: A Primer

The suggested hashtag for this book is #csharpandxmlprimer.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#csharpandxmlprimer

Contents

XML Basics **1**
Introduction 1
XML Heading 1
Elements 2
Attributes 2
Namespaces 2

XML Basics

Introduction

To start, let's cover some of the basics of XML. XML, which stands for Extensible Markup Language, is a markup language that allows for flexibility because the tags are not hard coded by any standards committee but can be created by any user at any time. This allows developers to create an XML standard themselves for use in saving data and transporting data.

Since the tags are created by the individual that needs to create an XML document, it allows an XML document to be very descriptive. You may not be familiar with XML but I bet you could understand what the below XML snippet is describing

```
<?xml version="1.0" encoding="utf-8" ?>
<house>
  <room>
    <name>Master Bedroom</name>
    <length>10</length>
    <width>20</width>
    <units>feet</units>
  </room>
</house>
```

The fact that XML can be very self-descriptive makes it a great tool to use when creating settings files or other configuration files. In fact, that is what Microsoft does. If you take a look at the any of the .csproj files, you will notice that there is XML there.

XML Heading

When starting an XML file, it is always a good idea to use the standard XML heading. This is so that any XML parsers know exactly what they are parsing and can also do any validating if need be. The standard heading for an XML file looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

This is going to go on the top of each XML file and tells the parser (as well as anybody reading the file) that this is using XML version 1.0 (the only version as of now) and has an encoding of "ISO-8859-1". This encoding is very important and as we will see later on, the way the .NET platform handles parsing XML can be a bit sensitive if you don't have the correct encoding.

Elements

Elements are the bulk of XML. In the example above, there are six different nodes: `house`, `room`, `name`, `width`, `length` and `units`. Elements need to have an open and closing tag, however, if there is no data in an element you can accomplish this in one tag. In the House example above, you can see that each element has a corresponding element. But what happens if we have an element with no data? Well you can follow the pattern above and go

```
<emptyElement\></emptyElement>
```

or you can perform that same action in just one tag

```
<emptyElement/>
```

In practice, you will most likely see the latter method used as it is easier and saves a few extra keystrokes.

Attributes

XML is more than just elements; we can also have attributes. Attributes provide another way to describe an item and are in the form of

```
<item basicAttribute="attribute value"/>
```

For a more concrete example, we can revise the house XML snippet to look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<house>
  <room name="Master Bedroom">
    <length>10</length>
    <width>20</width>
    <units>feet</units>
  </room>
</house>
```

You'll notice that we removed a child of `room` and added the `name` attribute. Because of the nature of XML, this doesn't change anything, it is just another way to describe the data.

Namespaces

Imagine that there are 10 developers working on a product and each part of the product needs to have a specific section that converts to XML so that the data can be saved. There is a chance that two developers on different teams could use the same name for an element. If that happens and

their code is looking for that name, it could cause a lot of problems and they could corrupt all of the data by loading the wrong element of the same name. Luckily there is a fix for this issue: using namespaces.

Namespaces in XML are similar to namespaces in C#, they both are there to organize the code as well as prevent name clashes. If you are using a namespace, you must declare that namespace in the root element of the XML document. In the below example we have an element of the same name but in different namespaces, which allows us to easily manipulate the correct element when that time comes:

```
<?xml version="1.0" encoding="utf-8" ?>
<root xmlns:h="http://www.w3.org/TR/html4/"
    xmlns:t="http://www.dontbreakthebuild.com/time">

    <h:span>
        <h:br/>
        <h:b>Today</h:b>
        <h:br/>
    </h:span>

    <t:span>
        <t:start>10-1-2013</t:start>
        <t:end>10-4-2013</t:end>
    </t:span>
</root>
```

In this example we have two namespaces: the first is html4 from the W3 specification and the other is a time namespace. Notice that both namespaces have a span element but the `h:span` is related to the HTML `span` element which is a text formatting element while `t:span` is relating to a time span.