Getting started with the XCRICAP-NET base library

Please note that this library is not maintained by, endorsed by, or in any other way officially connected with, the xcri.org website or the people and organisations involved with XCRI in the UK.

Aims

The aim of this project is to give providers who may already have base data access code written in .NET an easy method to output a valid XCRI feed with minimal effort. The codebase currently does not support the delta update pattern mentioned within the XCRI blog.

Project structure

The project is contained within one single DLL named "xcri.dll"; the source of this is available via the Google Code website. The project itself aims to allow the output of the base XCRI specification with minimal work, whilst also allowing for the output of customised XML extensions where applicable.

Creating a new XCRI feed

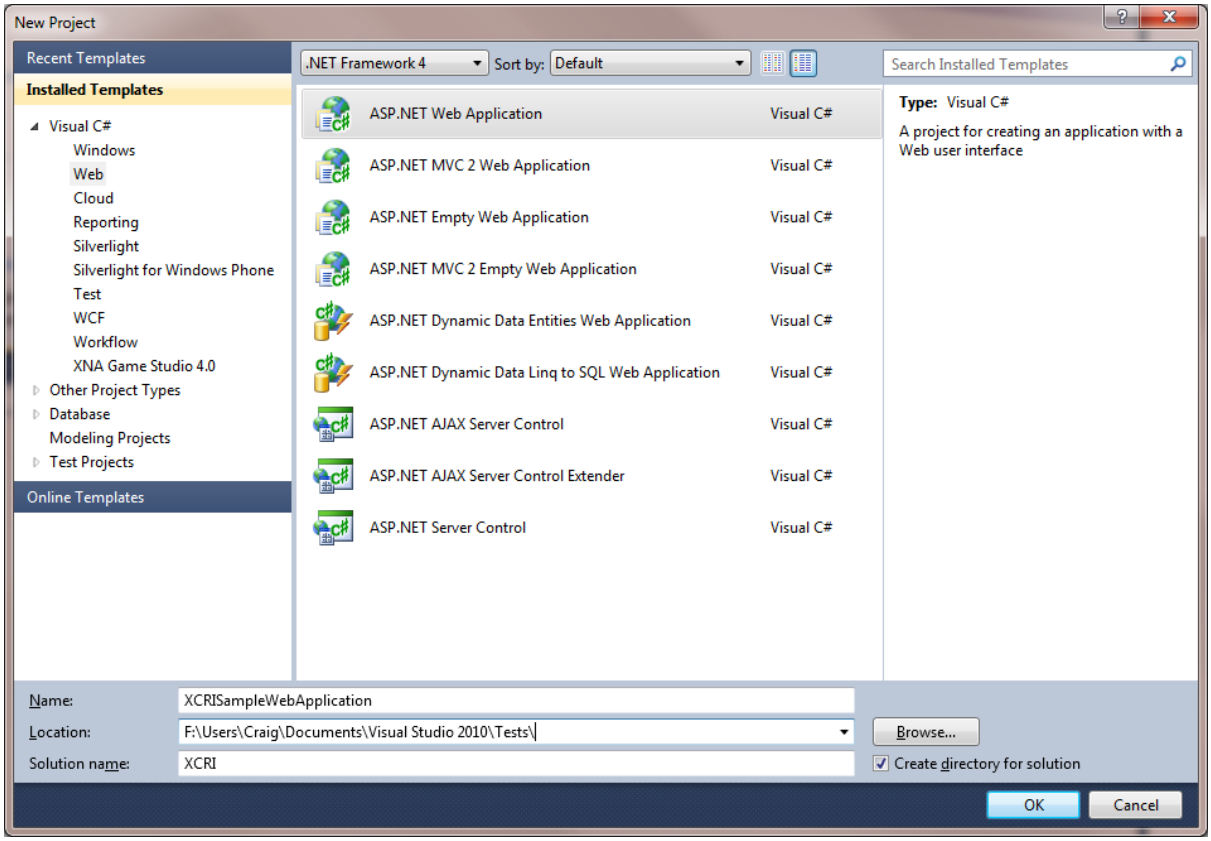
Please note that this documentation deals with using the DLL from within Visual Studio and concentrates upon using C# as the programming language.

Setting up a project within Visual Studio

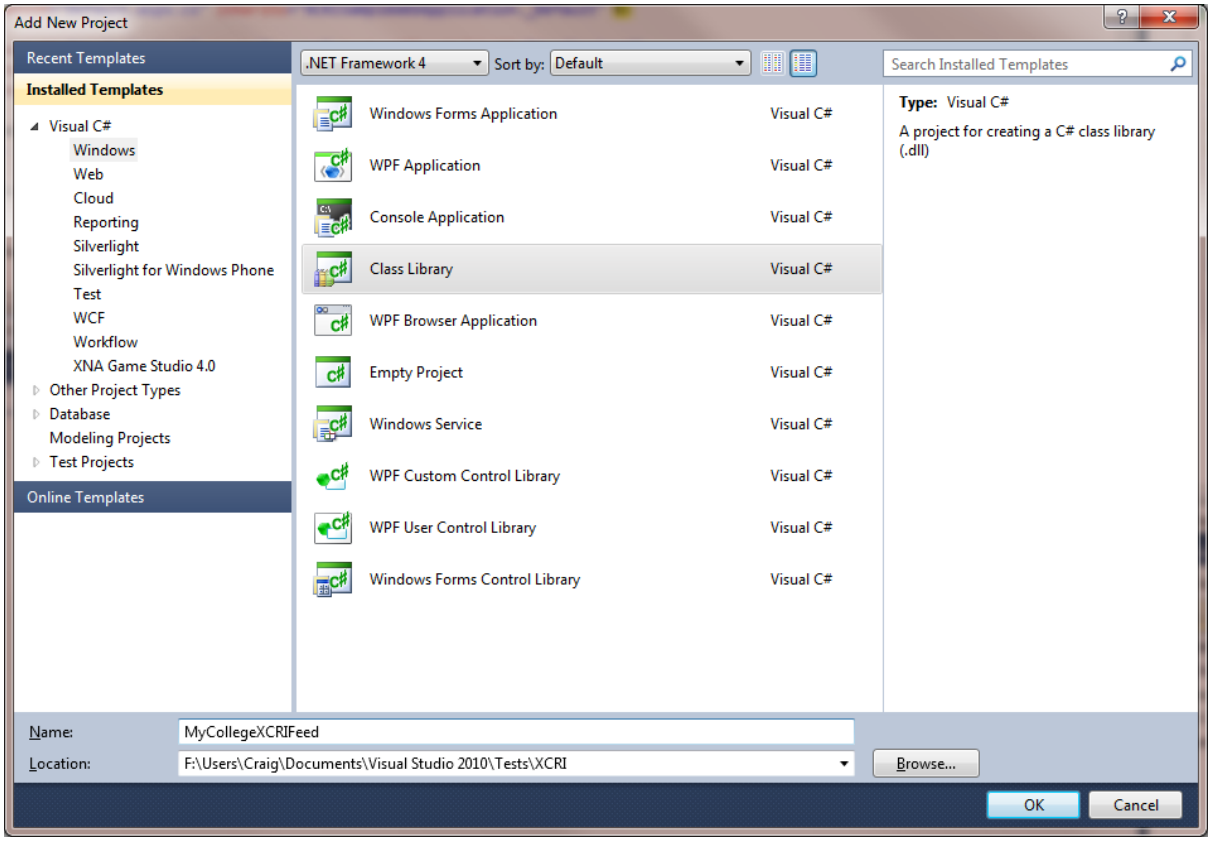
This will concentrate upon creating a sample Class Library and ASP.NET Web Application to use it from.

1. Create a new ASP.NET Web Application project within Visual Studio. Locate this wherever makes sense and choose an applicable name for the solution and project such as

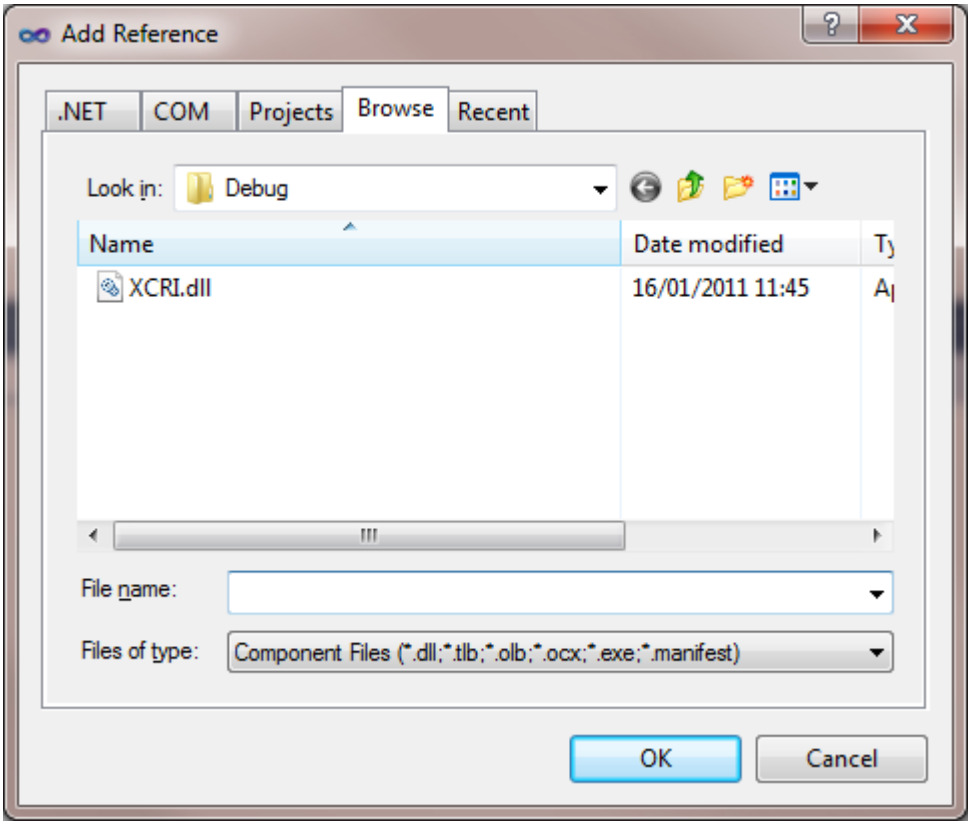
"XCRISampleWebApplication".



2. Right-click on the solution node and select "Add | New project" to add a new Class Library project. Locate this in a similar vein to the project above and choose an applicable name such as "MyCollegeXCRIFeed".



3. Add a reference within the Class Library project to the release XCRI.dll file which can be downloaded from http://xcricap-net.googlecode.com. You may also need to add a reference to your base data access layer, if you already have one.

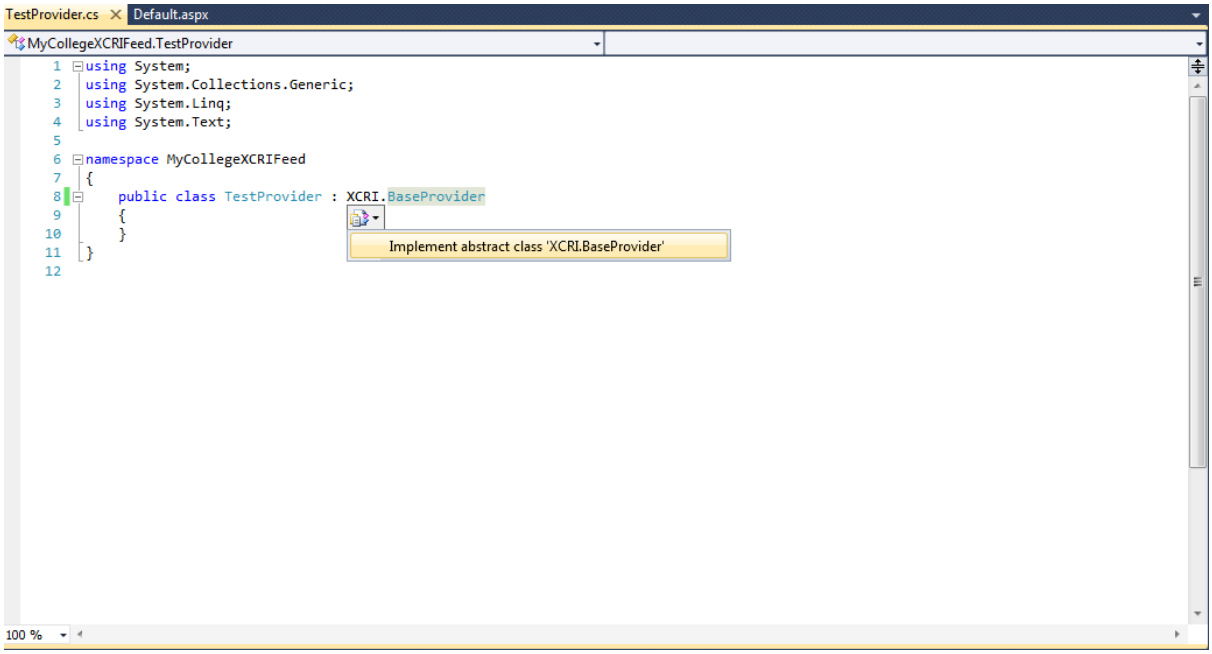


4. Add a reference within the Web Application to both the XCRI.dll file and your Class Library project.

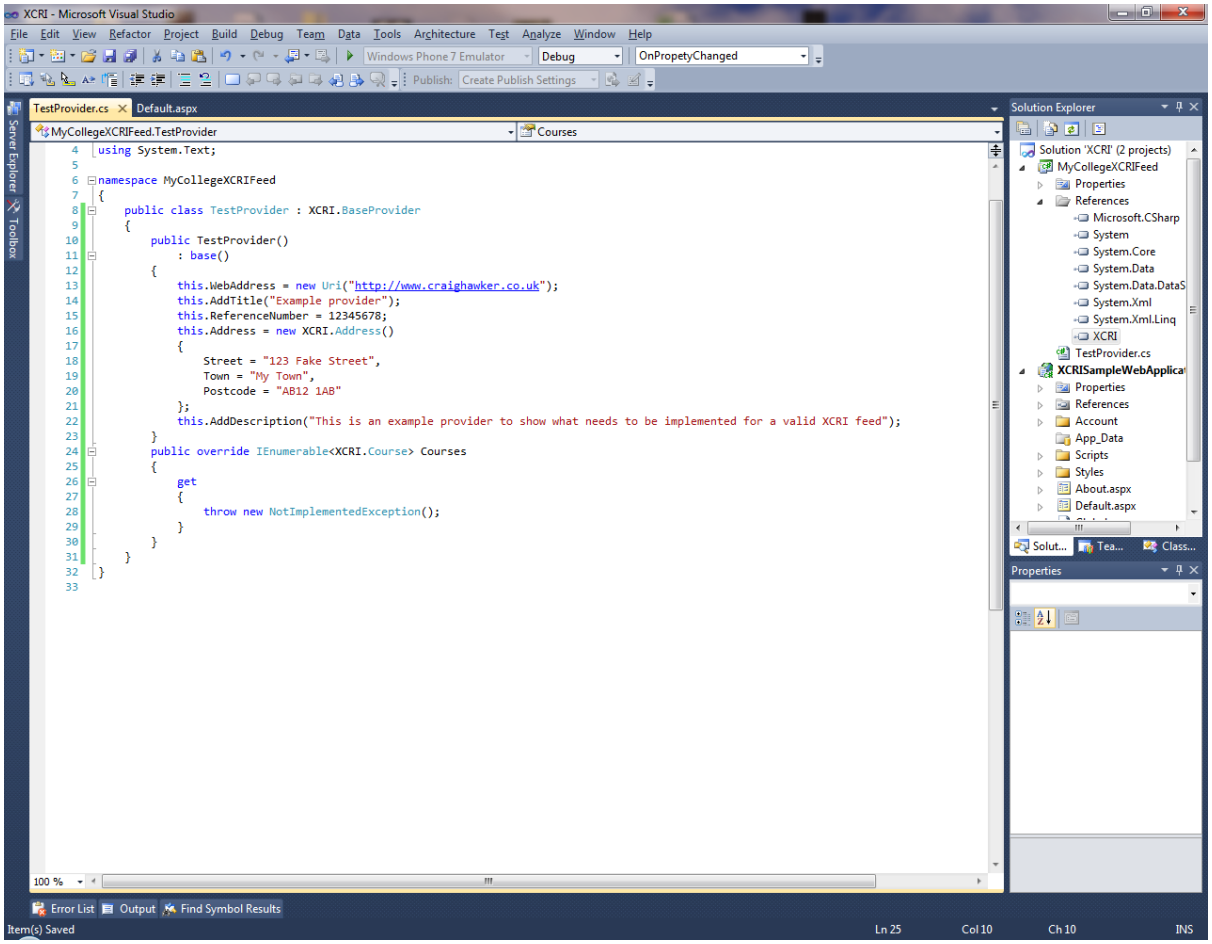
At this point we're ready to start creating a class hierarchy that extends the one provided and allows you to quickly produce an XCRI feed. For the sake of this document we're not going to investigate integrating with any existing data access layer, we're going to just create sample objects.

1. In the Class Library, remove the "Class1.cs" file which is automatically generated.
2. Add a class file and name it "MyCollegeProvider". For this sample, I've named it "TestProvider". This represents a logical provider of courses – for example, this may be a single college or university. If your college or university also has an associated entity that offers courses to businesses, as an example, we could create a second provider for their data.
3. Open the provider class up for testing. This provider class needs to extend the XCRI.BaseProvider class, so add that code in and get VS to auto-populate the required

members (Ctrl+.)

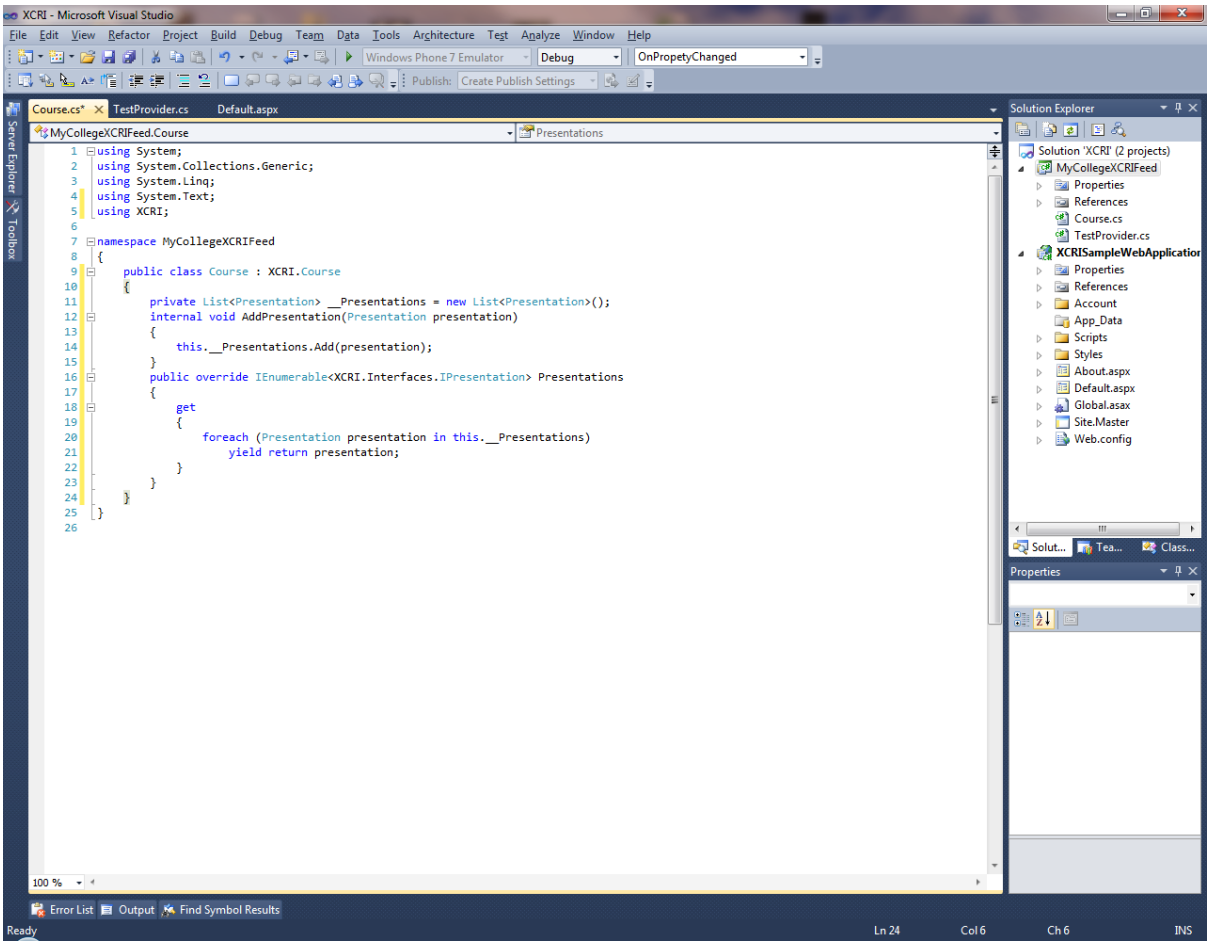


4. Add in a new default constructor and set the minimum required set of values to defaults. You may have another methodology for this, or you may want to populate these values programmatically at runtime – in which case, use your own judgement.



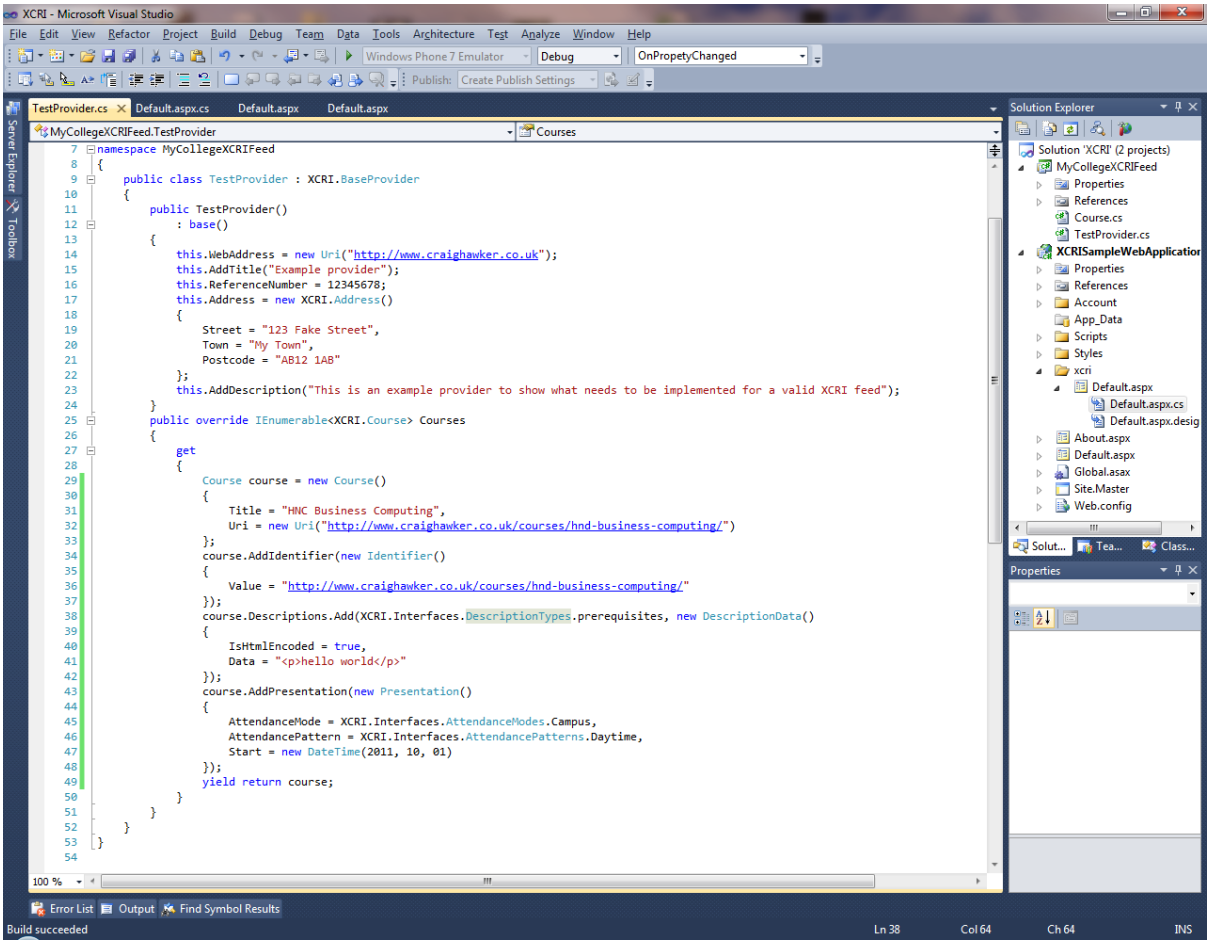
5. Note that we must implement a mechanism for returning course details, but the XCRI.Course class is marked as abstract. Create a new class and name it “Course”, inheriting

from the XCRI.Course object. Note that I've also implemented the abstract member "Presentations", although the implementation is very basic.



6. Now you need to populate the getter for "Courses" within your Provider. For the purposes of this document, I'm manually returning a course. In real life you would modify this code to talk to your database or data access layer and extract data from your systems to return a

populated Course object:

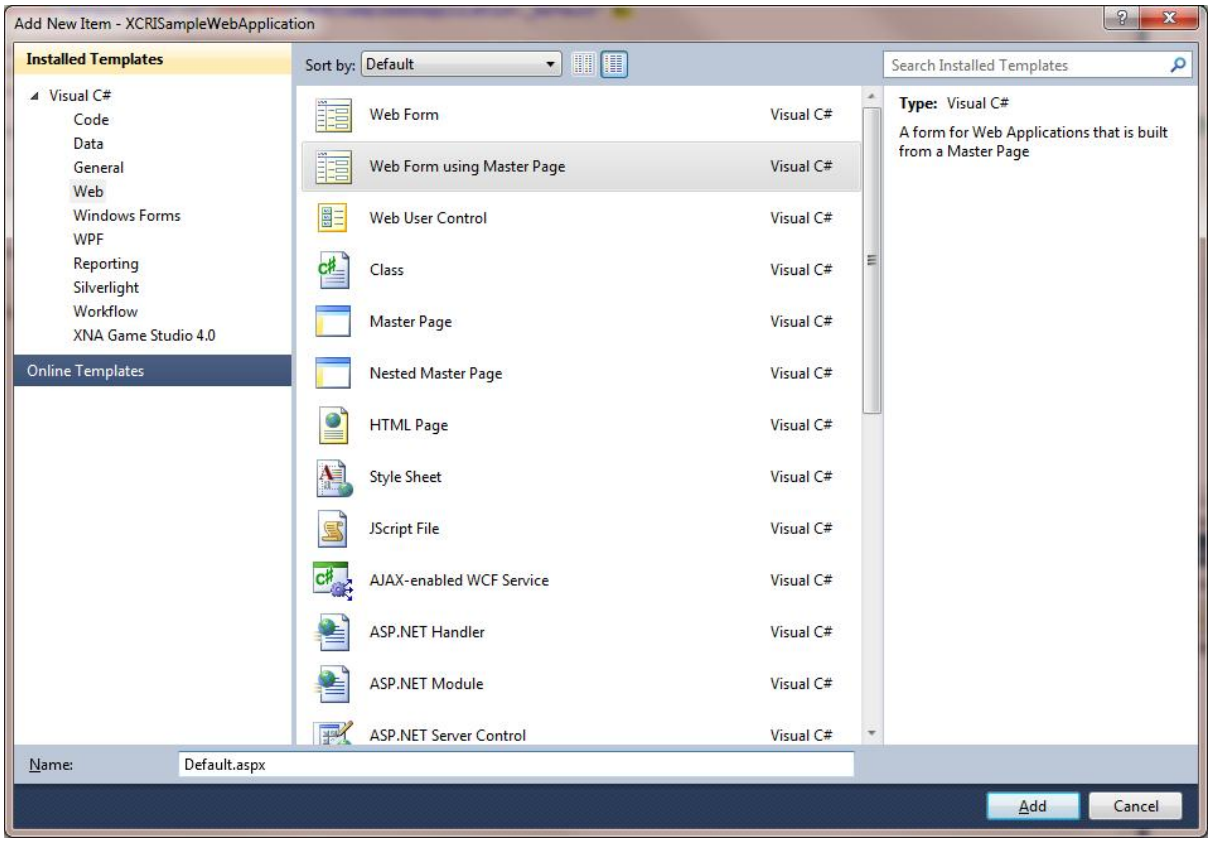



```
namespace MyCollegeXCRIFeed
{
    public class TestProvider : XCRI.BaseProvider
    {
        public TestProvider()
            : base()
        {
            this.WebAddress = new Uri("http://www.craighawker.co.uk");
            this.AddTitle("Example provider");
            this.ReferenceNumber = 12345678;
            this.Address = new XCRI.Address()
            {
                Street = "123 Fake Street",
                Town = "My Town",
                Postcode = "AB12 1AB"
            };
            this.AddDescription("This is an example provider to show what needs to be implemented for a valid XCRI feed");
        }
        public override IEnumerable<XCRI.Course> Courses
        {
            get
            {
                Course course = new Course()
                {
                    Title = "HNC Business Computing",
                    Uri = new Uri("http://www.craighawker.co.uk/courses/hnd-business-computing/")
                };
                course.AddIdentifier(new Identifier()
                {
                    Value = "http://www.craighawker.co.uk/courses/hnd-business-computing/"
                });
                course.Descriptions.Add(XCRI.Interfaces.DescriptionTypes.prerequisites, new DescriptionData()
                {
                    IsHtmlEncoded = true,
                    Data = "<p>hello world</p>"
                });
                course.AddPresentation(new Presentation()
                {
                    AttendanceMode = XCRI.Interfaces.AttendanceModes.Campus,
                    AttendancePattern = XCRI.Interfaces.AttendancePatterns.Daytime,
                    Start = new DateTime(2011, 10, 01)
                });
                yield return course;
            }
        }
    }
}
```

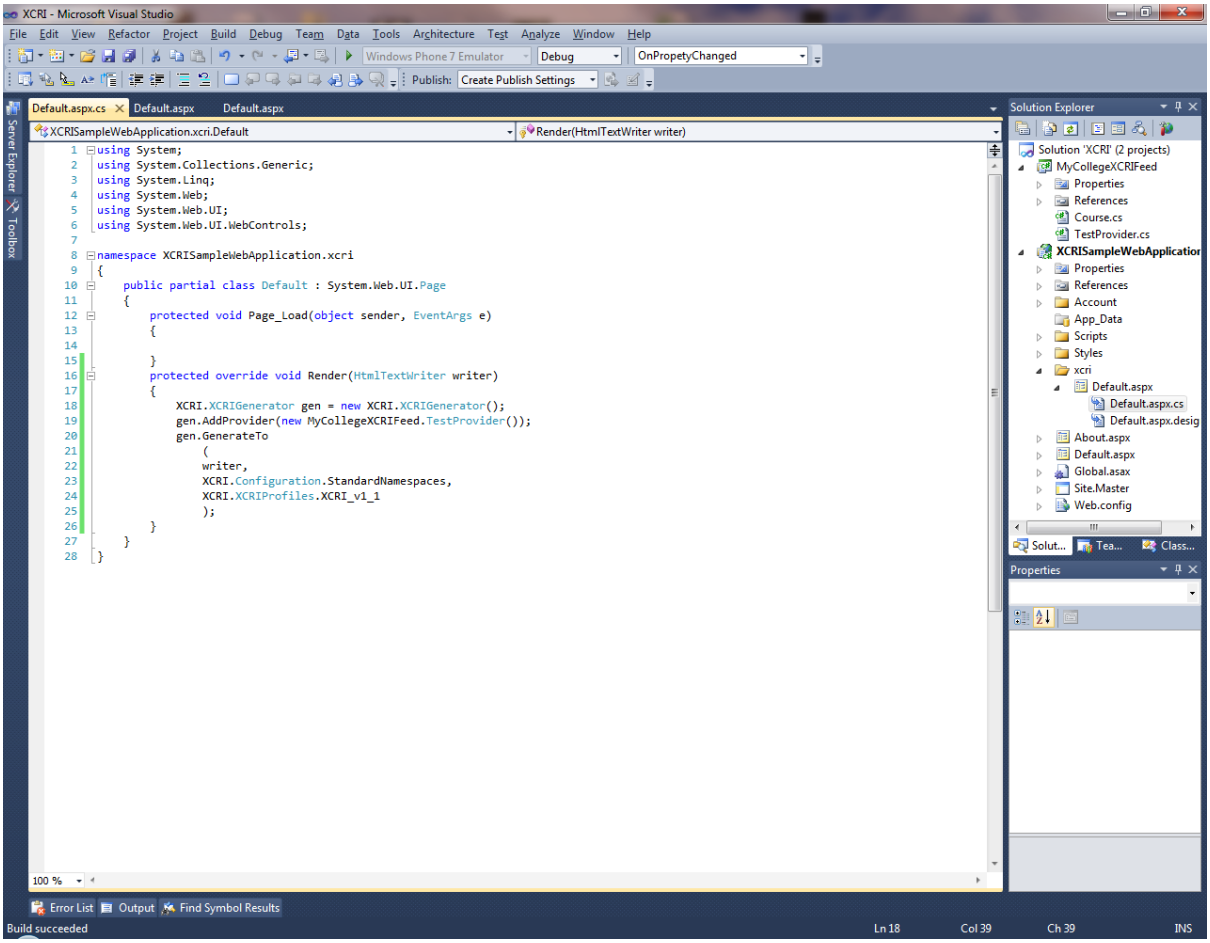
7. At this point your object model is complete and returning (sample) data. Next it's on to modifying the web application to return a valid XCRI feed.

The next step is to modify the sample web application to output your XCRI feed when someone requests it. Obviously this stage is artificial and you would incorporate this step into your public-facing website. Please note that, due to the potential size of the XCRI feed, you may also want to investigate caching this information.

1. Create a new folder within the application named “xcri” and create a Default.aspx page within it:

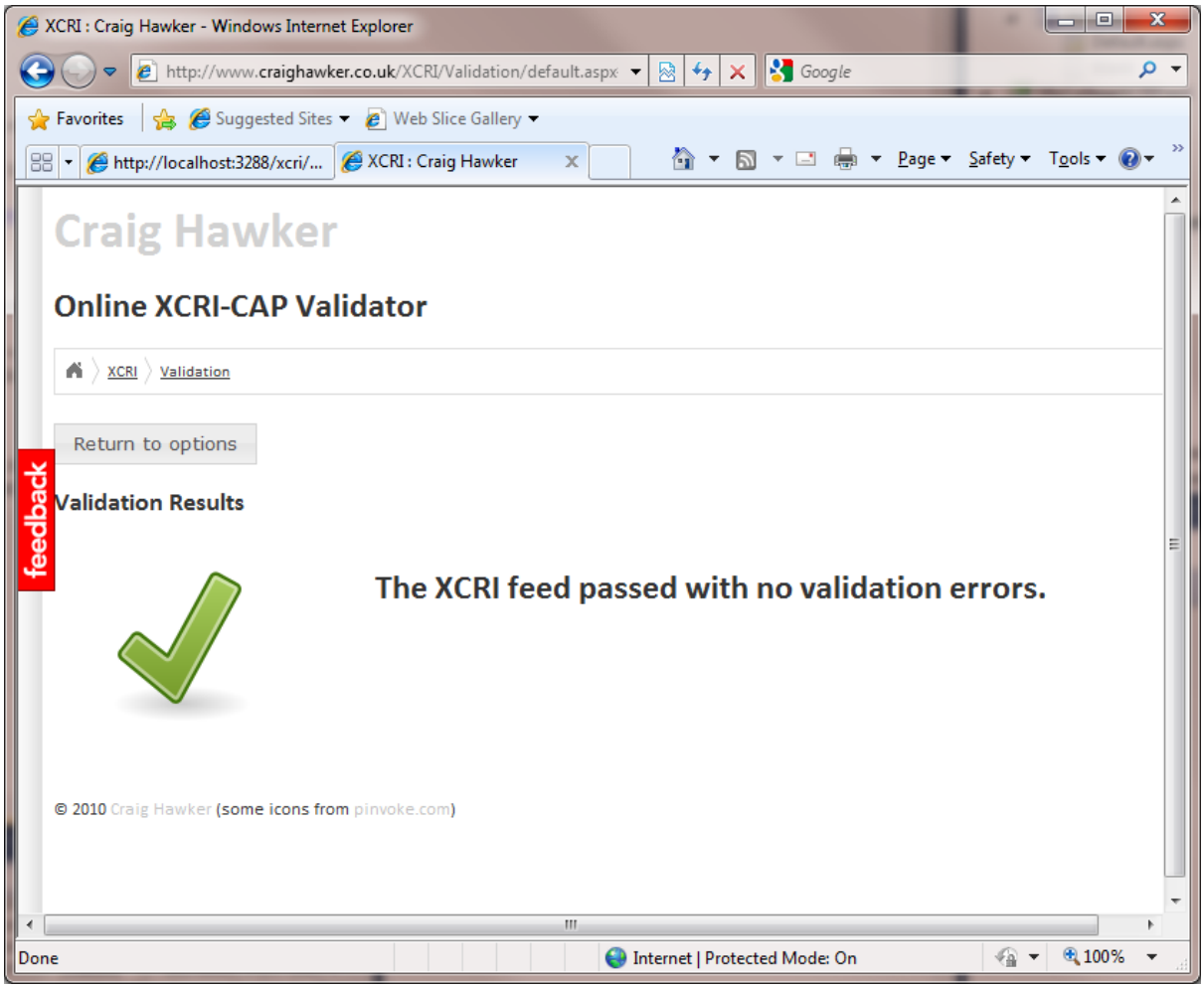


2. Go into the code view for the page and override the Render method, creating a new instance of XCRI.XCRIGenerator, adding your provider to the list, then calling XCRIGenerator.GenerateTo:



Please note that in a real-life scenario you should also ensure that HTTP headers are correctly set to ensure that the file is served as an XML file.

3. Just to go full-circle, let's view the source of the page and put it through the online validator at http://www.craighawker.co.uk/xcri/:



Next, a couple of notes on the XCRI location: there're two recommended XCRI location patterns mentioned within the blog. Both auto-discovery features can be tested using the online XCRI validator:

1. The "reliable feed location" pattern (http://www.xcri.org/wiki/index.php/Reliable_feed_location)
   The XCRI feed is always in a location which is expected – this is defined as http://www.myeducationalinstitution.ac.uk/xcri/. Please note that Uri locations are case-sensitive (even though IIS/Windows implementations aren't).
2. The "feed autodiscovery" pattern (http://www.xcri.org/wiki/index.php/Feed_autodiscovery)
   The XCRI feed is located through the use of a HTML "meta tag". To use it, the provider includes a <link /> tag within the <head> section of their HTML source code. This then, in turn, points to the XCRI feed.