

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.

Document history

Version	Release Date	Author(s)	Notes
1.0	2011-02-13	Craig Hawker	Compatibility with release 0.1
1.1	2011-04-12	Craig Hawker	Updated for compatibility with 0.3.0a

Contents

Document history	2
The Project.....	4
Project aims.....	4
Project licence	4
Obtaining the code.....	4
Object model structure	4
Project feedback.....	4
Creating a new XCRI feed	5
Setting up a project within Visual Studio	5
Creating your Object Model	5
Generating an XCRI-CAP feed.....	8
Feed validation	9
Notes	10
Reliable Feed Location	10

The Project

Project 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 allows provides the ability to create XCRI-CAP documents using the XCRI-CAP 1.1 and XCRI-CAP 1.2 namespaces, plus additional namespaces such as the XCRI-CAP 1.1 Terms (built-in) or others (through extension).

The library aims to support 100% of the XCRI-CAP 1.1 standard including the [delta-update-pattern](#).

The library does not aim to provide any validation. It is expected that external validation tools be used, <shameless_plug> such as the XCRI-CAP Online Validator at <http://www.craighawker.co.uk/xcri/> </shameless_plug>.

Project licence

Please note that this project in both source and compiled forms is licensed under a Creative Commons Attribution Non-Commercial ShareAlike licence. You may use this software library and its source code for free if you work directly for an educational institution (HE or FE) in a full-time capacity. If you are a consultant, contractor, or third party who provides services to an educational institution (regardless of whether those services are free or paid), you may not use this software without additional licensing. Please contact the project owner at <http://www.craighawker.co.uk> for further licensing information.

Obtaining the code

The code for the XCRICAP-NET can be obtained in two ways:

1. By downloading the (compiled) binary files from <http://code.google.com/p/xcricap-net/downloads/list>.

This is the route most developers should take as the binary files contain everything required to use the library. Simply add a reference to the DLL and follow the guide in this document to start producing XCRI feeds.

2. By downloading the source C# files from the Google Code repository at <http://code.google.com/p/xcricap-net/source/checkout>.

This is the route you should take if you would like to help with the project by looking at the source code, submitting patches, or identifying issues. Please note that this is a Mercurial (hg) repository and you will need local tools to work with it.

Object model structure

Full details on the project model are kept within the Google Code wiki under the [Getting Started](#) page.

Project feedback

Any project feedback should be directed to Craig Hawker, preferably through <http://twitter.com/CraigHawker>.

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 "XCRIExampleWebApplication".
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 "MyCollegeXCRIAdapter".
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.

Creating your Object Model

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.

Please note that you may choose to extend the XCRICAP-NET classes, as shown below, or you may decide to instantiate them and populate them as you go. Your choice will depend upon your documented coding style and how you intend on exposing the information. For the sake of this document, we'll deal with extending the Provider object and returning a dummy set of data.

1. In the Class Library, remove the "Class1.cs" file which is automatically generated.
2. Add a class file and name it "MyCollegeProvider". 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 up the provider class within Visual Studio and modify the class to extend the [XCRI.Provider](#) class. 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.

```
public MyCollegeProvider()
    : base()
{
    this.Url = new Uri("http://www.craighawker.co.uk");
    this.Titles.Add(new XCRI.Title() { Value = "Craig Hawker College
(test)" });
    this.ReferenceNumber = 12345678;
    this.Location = new XCRI.Location()
    {
        Street = "123 Fake Street",
        Town = "My town",
        Postcode = "AB12 1AB"
    };
    this.Descriptions.Add(new XCRI.Description()
    {
        Value = "This is an example provider to show what must be
implemented for a valid XCRI feed."
    });
}
```

4. For the purpose of this document we're going to fake loading the Course information from a database and, instead, manually populate the course and presentation information. In a real scenario, this would interact with your current course datasource.

```

        // This is generic information about the course
        XCRI.Course course1 = new XCRI.Course()
        {
            Url = new Uri("http://www.craighawker.co.uk/courses/hnc/business-
computing/")
        };
        course1.Identifiers.Add(new XCRI.Identifier()
        {
            Value = course1.Url.ToString()
        });
        course1.Titles.Add(new XCRI.Title()
        {
            Value = "HNC Business Computing"
        });
        course1.Descriptions.Add(new
XCRI.Vocabularies.XCRICAP11.Terms.Description()
        {
            XsiTypeValue =
XCRI.Vocabularies.XCRICAP11.Terms.DescriptionTypes.prerequisites,
            Value = "5 GCSE grades A-C, preferably including a computing
element"
        });
        // This is a means in which a student can take the course.
        course1.Presentations.Add(new XCRI.Presentation()
        {
            AttendanceMode = new
XCRI.Vocabularies.XCRICAP11.Terms.AttendanceMode()
            {
                Value =
XCRI.Vocabularies.XCRICAP11.Terms.AttendanceModeTypes.Campus
            },
            AttendancePattern = new
XCRI.Vocabularies.XCRICAP11.Terms.AttendancePattern()
            {
                Value =
XCRI.Vocabularies.XCRICAP11.Terms.AttendancePatternTypes.Daytime
            },
            Start = new XCRI.Date(new DateTime(2011, 10, 01))
        });
        // This is a qualification which a student can achieve when
undertaking the course - hacky code
        course1.Qualifications.Add(new XCRI.Qualification());
        course1.Qualifications[0].Titles.Add(new XCRI.Title() { Value = "HNC
in Business Computing" });
        // Add the course to the course list
        this.Courses.Add(course1);

```

5. 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.

Generating an XCRI-CAP 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 the resultant document.

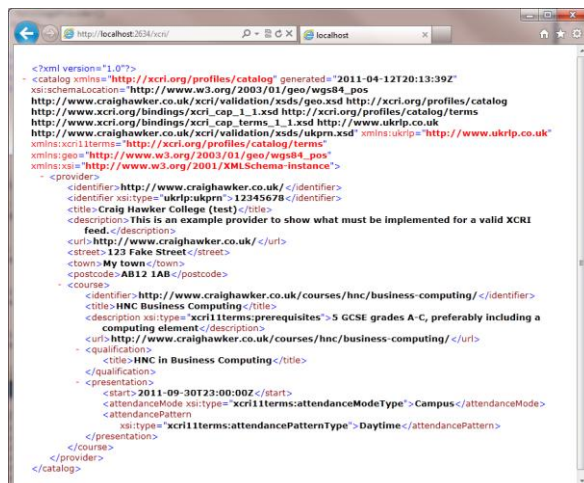
1. Create a new folder within the application named “xcri” and create a Default.aspx page within it ([read more about the reliable feed location pattern](#)).

```
<%@ Page
    Language="C#"
    AutoEventWireup="true"
    CodeBehind="Default.aspx.cs"
    Inherits="XCRIExampleWebApplication.xcri.Default"
    ContentType="text/xml"
%>
```

2. Ensure that the content type for the page is set to “text/xml”.
3. Override the Render method and output the XML to the response stream. Note that this generates an XCRI-CAP 1.1 document ([XCRIProfiles.XCRI v1 1](#)). Outputting an XCRI-CAP 1.2 document would involve calling `XCRI.XmlGeneration.XmlGeneratorFactory.GetXmlGenerator`, passing a value of `XCRI.XCRIProfiles.XCRI_v1_2`. All other code would be identical and could use the same object model.

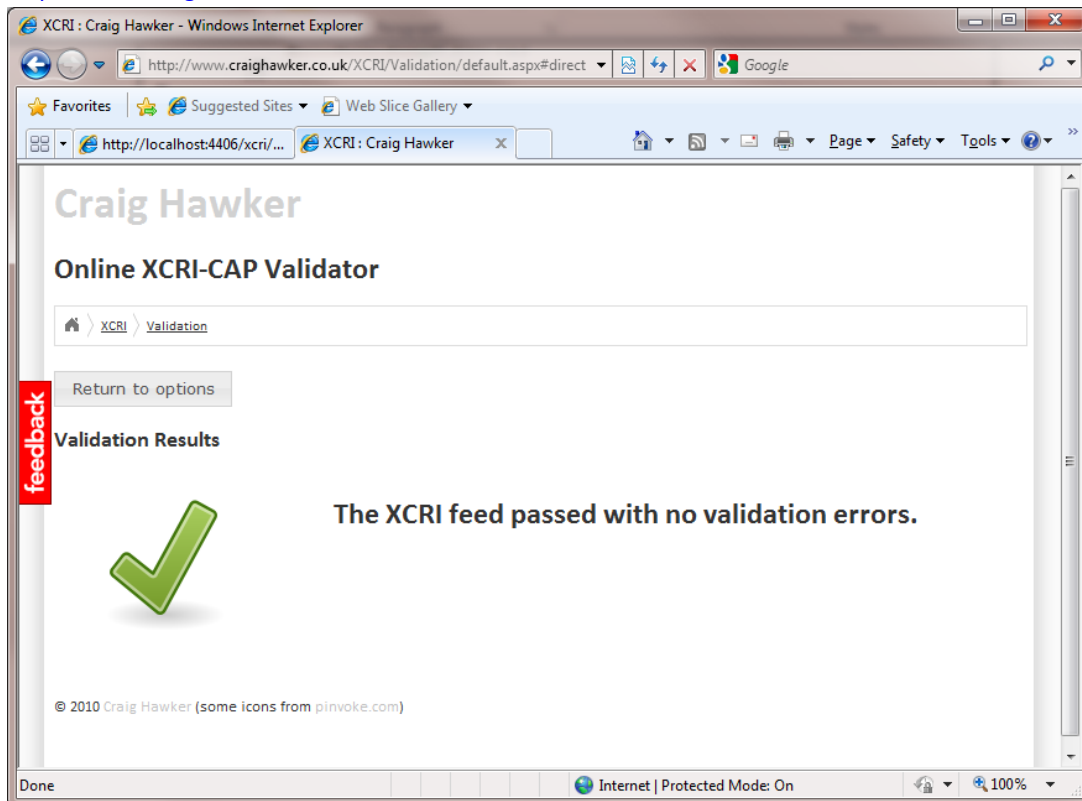
```
protected override void Render(HtmlTextWriter writer)
{
    // Create an XML generator for XCRI-CAP 1.1
    XCRI.XmlGeneration.Interfaces.IXmlGenerator
        gen =
XCRI.XmlGeneration.XmlGeneratorFactory.GetXmlGenerator(XCRI.XCRIProfiles.XC
RI_v1_1);
    // This will be the root element (valid in most scenarios)
    XCRI.Catalog catalog = new XCRI.Catalog();
    // Add our provider
    catalog.Providers.Add(new
MyCollegeXCRIAdapter.MyCollegeProvider());
    // Set the root element
    gen.RootElement = catalog;
    // Force generation
    gen.Generate(writer);
}
```

Run the code and you should get a browser window open with an XML (XCRI-CAP 1.1) document displayed similar to the one on the right.



Feed validation

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/>:



Notes

Reliable Feed 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](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](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.