Project: DataUp Development Guide

Version 1.0 - August, 2012

Abstract

This document describes how to contribute code to the DataUp Project.

For updates to this document and the rest of the DataUp Project, see dataup.cdlib.org

Contents

Installing Visual Studio 2010

You should already have a system that you can logon, can obtain administrative rights to, and the 'normal' software installed, e.g. **Windows, Internet Explorer, Office**.

To install Visual Studio

1. Install Visual Studio 2010

Install Source Control Tools

TBD

Installing Open XML SDK 2.0

Download and install OpenXMLSDKv2.msi from the following location:
http://www.microsoft.com/en-us/download/details.aspx?id=5124

Installing Windows Azure SDK for .NET - June 2012 (v 1.7)

Download and install windows azure SDK 1.7 from the following location:
http://www.microsoft.com/en-us/download/details.aspx?id=29988

Download Common Service Locator v1.0 binaries

Download and unzip `CommonServiceLocatorBinaries.zip` binaries from the following location: http://commonservicelocator.codeplex.com/releases/view/17694

Download DotNetZip Library v1.9.1.8 binaries

Download and unzip `DotNetZipLib-Runtime-v1.9.zip` binaries from the following location: http://dotnetzip.codeplex.com/releases/view/68268

Installing Entity Framework 4.3.1

Download Entity Framework 4.3.1 from the following location:
http://nuget.org/packages/EntityFramework/4.3.1

Installing Microsoft Unity 2.1

Download Microsoft Unity 2.1 from the following location:
http://www.microsoft.com/en-us/download/details.aspx?id=17866

Downloading Source Code from Bit Bucket

TBD

To connect to Source Control

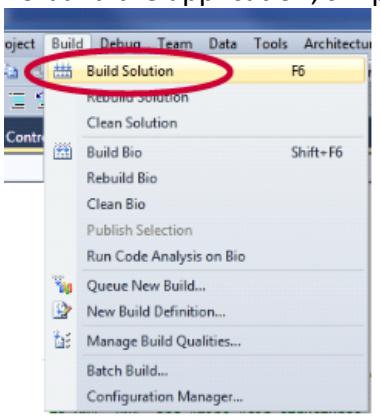
TBD

Pre-Build operations

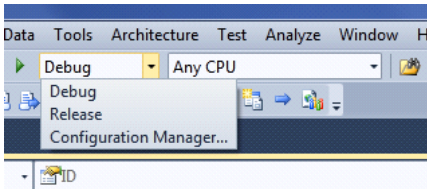
1. Create a folder called “Libs/Ionic” under the root of the workspace. ($/DataUp/Libs/Ionic)
2. Copy “Ionic.Zip.dll” from “DotNetZipLib-Runtime-v1.9/v1.9” to ($/DataUp/Libs /Ionic)
3. Create a following folder structure if not present “$/DataUp/WebClient\packages\Open XML SDK\V2.0\lib\”
4. Copy “DocumentFormat.OpenXml.dll” from “%program files%\Open XML SDK\V2.0\lib” to “$/DataUp/WebClient/packages/Open XML SDK/V2.0/lib/”

Building Excel Add-in Solution

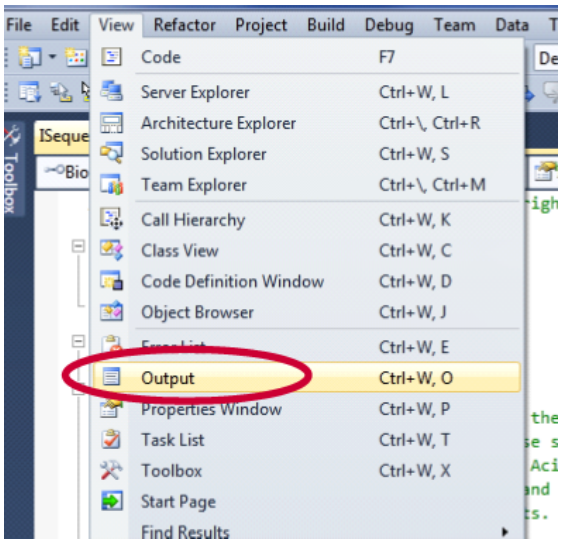
To build the application, simply select the Build toolbar at the top and Build Solution:



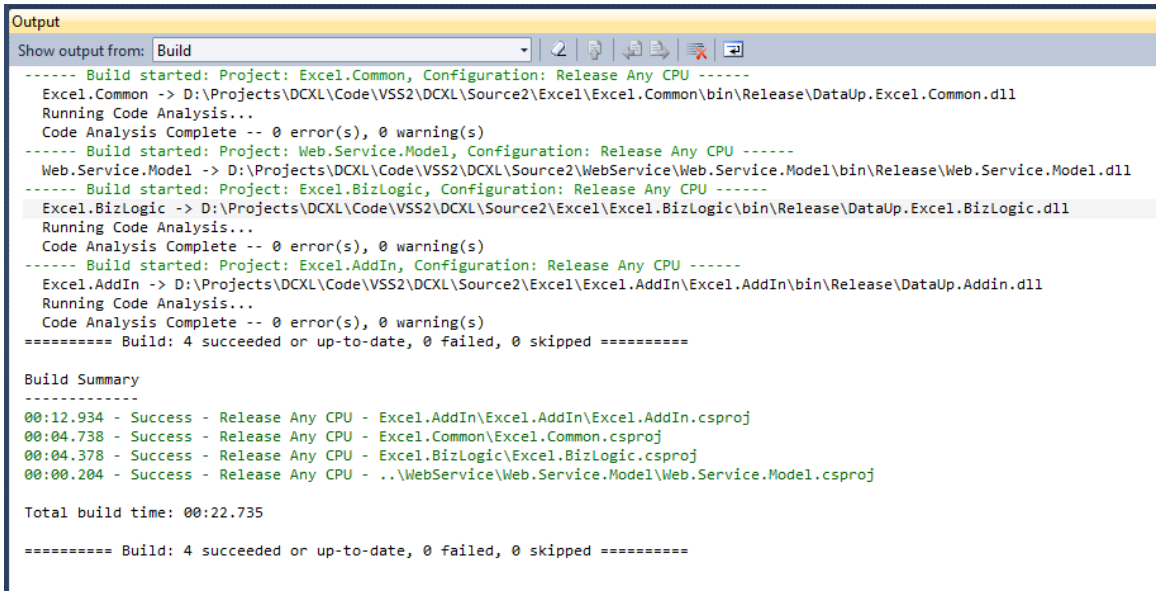
The options to build Release or Debug versions of the code are set via the drop down in the top toolbar:



To monitor the progress of the build, enable the Output window. This can be done by selecting the **View** option and then selecting the **Output** option:

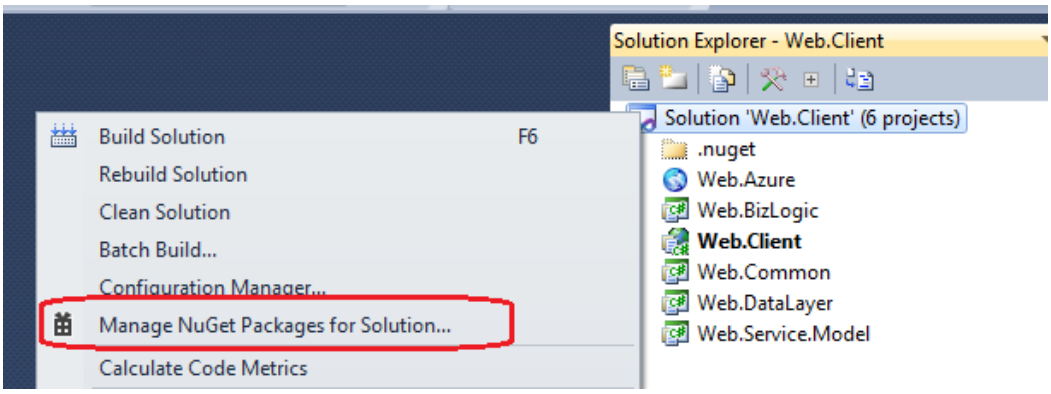


This is what you'll see when the build is completed:



Building Web Client Solution

1) Adding and associating NuGet Package References
   a) Open "Manage NuGet Packages for Solution", by right click on the solution as shown below.



b) Install and associate following packages to the projects from NuGet Manager:
   - ASP.NET Universal Providers (v 1.0.1)
     - Web.Client
   - CommonServiceLocator (v 1.0)
     - Web.Client
   - EntityFramework (v 4.3.1)
     - Web.Client
     - Web.DataLayer
   - Unity (v2.1.505.0)
     - Web.Client
   - Windows Azure (v1.7.0.0)
     - Web.Client
     - Web.BizLogic
     - Web.DataLayer

c) Close NuGet Manager

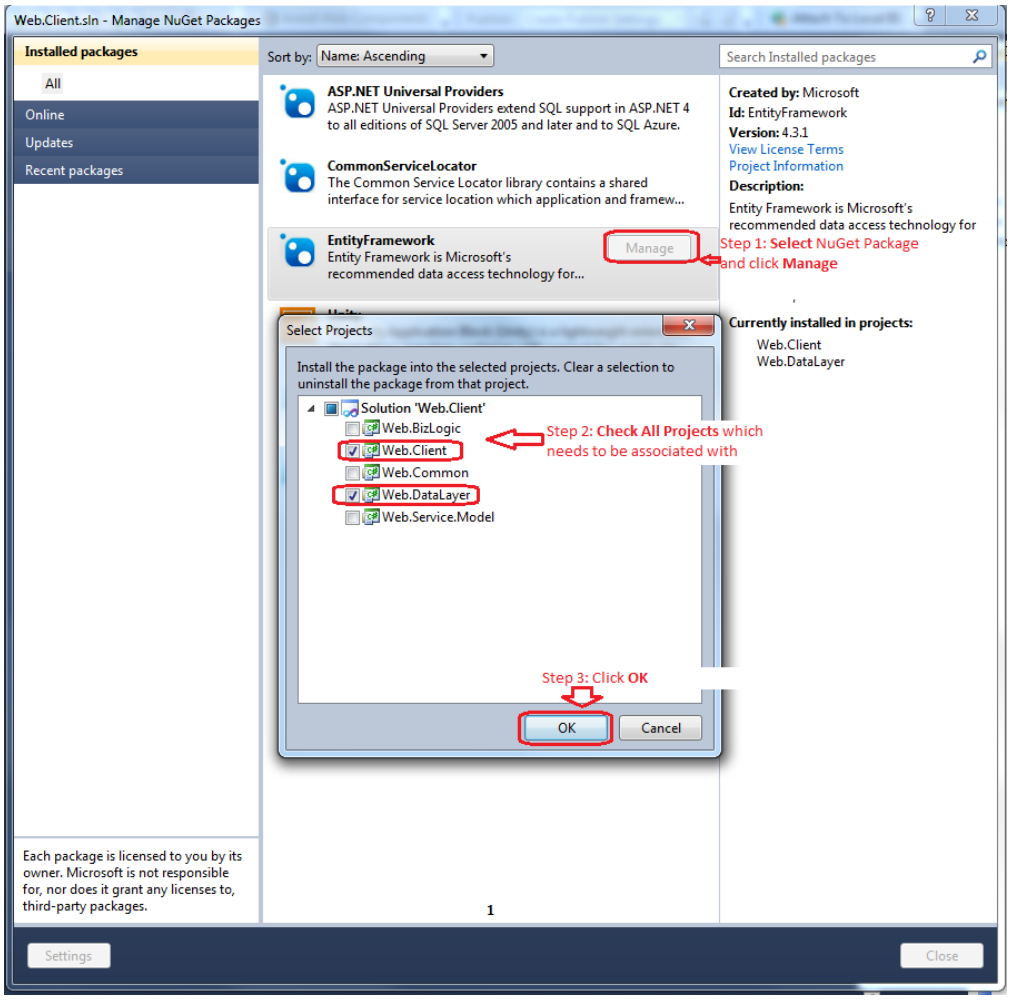
2) Build Web.Client as we did for Excel.Add-in

Note:

For associating NuGet packages with the Project

1. Click “Manage
2. Check all project for which the selected NuGet package has to be associated with
3. Click “OK”

For Example:



Building Web Admin Solution

1) Adding and associating NuGet Package References (Refer previous section for steps on NuGet manager and how to associate projects.)
   a) Associate the following NuGet packages to the projects.
   - ASP.NET Universal Providers (v 1.0.1)
     - Web.Admin
   - CommonServiceLocator (v 1.0)
     - Web.Admin
   - EntityFramework (v 4.3.1)
     - Web.Admin
     - Web.Admin.DataLayer
   - Unity (v2.1.505.0)
     - Web.Admin
   - Windows Azure (v1.7.0.0)
     - Web.Admin
     - Web.Admin.DataLayer

2) Build Web.Service as we did for Excel.Add-in

Conclusion

This document was intended to provide a quick introduction in how to get started developing for Project DataUp. Please read the Coding Guidelines document, as well gain a thorough understanding of how the various parts interact, before attempting to modify the code base. Gaining a good background on the project will help to make sure your first code review goes smoothly.

Welcome to the Project DataUp!

Adding New Repository:

Implement “IRepository” for the new Repository

Implement the following methods in interface “`IRepository`”.

- `GetIdentifier` :- This method will be used to get the unique identifier for the document.
- `PostFile` :- This method will be used for publishing the data to the specified repository.

Adding new Repository Type

1) Go to admin service and add a new Repository type running following SQL command:

```
INSERT INTO [dbo].[RepositoryType]
           ([RepositoryTypeId] -- ID of the repository type. This
is normally Next Sequence number in the list
           ,[RepositoryType] -- Type name of the repository
           ,[PasswordRequired]) -- Whether user has to enter
password at the time of accessing the repository
     VALUES
           (3 -- Replace with Actual value.
           ,'newType' -- Replace with Actual value.
           ,1)
```

2) Update the following method in “`RepositoryFactory`” class

   a) `GetRepositoryInstance`: This method will return the instance of which repository implementation should be used based on the type of the repository.

Adding new Repository

Use DataUp admin client for adding new repository for the newly created repository type

Sample Implementation

In this sample, we will be creating a new file repository. This repository will store the file in a separate folders identified by the Identifier.

Implementing IRepository

```
using System;
using System.Globalization;
using System.IO;
using System.Text;
using DataUp.Web.Service.Model;
```

```
namespace DataUp.Web.Admin
{
    public class FileRepository : IRepository
    {
        enum FileCreationType
        {
            ERC,
            EML,
            ManiFest
        }

        public string GetIdentifier(DQueryData queryData, DRepositoryBase repo
sitoryModel)
        {
            string restResponse;

            try
            {
                restResponse = GetNextDirectoryname();
            }
            catch (Exception exception)
            {
                return "false|" + exception.Message;
            }

            // Get Identifier from the response.
            return "true|" + restResponse;
        }

        private static string GetNextDirectoryname()
        {
            string nextDirectoryname;
            nextDirectoryname = Guid.NewGuid().ToString();

            foreach (var character in Path.GetInvalidPathChars())
            {
                nextDirectoryname = nextDirectoryname.Replace(character.ToStri
ng(), string.Empty);
            }
            return nextDirectoryname;
        }

        public string PostFile(DQueryData queryData, DRepositoryBase repositor
yModel, DFile file)
        {
            DQueryData request = queryData;
            Encoding encoding = Encoding.UTF8;

            string fileName = file.FileName.Trim();
            if (string.IsNullOrEmpty(fileName))
            {
                return "false|A non-empty file name is needed.";
            }

            //Ensure there's no directory path in the file name
```

```
            fileName = Path.GetFileName(fileName);

            if (file.FileContent == null || file.FileContent.Length == 0)
            {
                return "false|File is empty.";
            }

            string identifier = request["ARK"].Value;

            string saveFolder = Path.Combine(Constants.TempDownloadPath, ident
ifier);

            Directory.CreateDirectory(saveFolder);
            try
            {
                DownloadFile(fileName, file.FileContent, file.FileExtentsion,
file.isCompressed, saveFolder);

            }
            catch (Exception exception)
            {
                return "false|" + exception.Message;
            }

            return "true|Success";
        }

        private static void DownloadFile(string fileName, byte[] fileContents,
 string fileExtentsion, bool isCompressed, string tempFolder)
        {
            string tempFileName = fileName;

            bool isMultiCSVPassed = false;
            if (isCompressed)
            {
                switch (fileExtentsion.ToLower())
                {
                    case Constants.ZIP:
                        tempFileName = string.Format(CultureInfo.InvariantCult
ure, Constants.TempFileName, Constants.ZIP);
                        isMultiCSVPassed = true;
                        break;
                    case Constants.TAR:
                        tempFileName = string.Format(CultureInfo.InvariantCult
ure, Constants.TempFileName, Constants.TAR);
                        isMultiCSVPassed = true;
                        break;
                    case Constants.GZIP:
                        tempFileName = string.Format(CultureInfo.InvariantCult
ure, Constants.TempFileName, Constants.GZIP);
                        isMultiCSVPassed = true;
                        break;
                }
            }
```

```
            string filePath = Path.Combine(tempFolder, tempFileName);
            FileStream fs = File.Create(filePath);
            fs.Close();

            File.WriteAllBytes(filePath, fileContents);

            if (isMultiCSVPassed)
            {
                ZipUtilities.UnZipFiles(filePath, tempFolder, "", true);
                File.Delete(filePath);
            }
        }
    }
}
```

Adding File Repository Type

1) Inserting a new Repository Type.

```
INSERT INTO [dbo].[RepositoryType]
           ([RepositoryTypeId]
           ,[RepositoryType]
           ,[PasswordRequired])
     VALUES
           (3
           ,' FileType'
           ,0)
```

2) Updating RepositoryFactory.GetRepositoryInstance

```
public static IRepository GetRepositoryInstance(string instanceName)
        {
            switch (instanceName)
            {
                case Constants.MerrittRepositoryName:
                    return new MerritRepository();
                case "FileType":
                    return new FileRepository();
                default:
                    return new MerritRepository();
            }
        }
```