AUTODESK **UNIVERSITY** 2013

Programming the Vault with the VDF

Dennis Mulonas – Autodesk
Doug Redmond – Autodesk

DV1670

The Autodesk Vault 2014 release adds a powerful set of programming tools: the Vault Development Framework (VDF). Learn how to unlock the power of the VDF. Get more features in your apps with less code. This class goes over the basics of the VDF, such as how to use the Login dialog.

Learning Objectives

At the end of this class, you will be able to:

- Explain the basics of the VDF
- Use UI components from the VDF
- Upload and download files through the VDF

About the Speakers

Dennis Mulonas is a software architect for the Vault. He has been working at Autodesk for 21 years.

Doug Redmond is a developer on Vault and PLM 360. He runs the blog It's All Just Ones and Zeros.

Introduction

This document is not just a repeat of the slide deck. Instead, this document will contain reference information and additional materials. The demo app shown in the class will be covered here, step-by-step.

Glossary

ADMS – Autodesk Data Management Server. The Vault server component that contains the database. In a multi-workgroup environment, there would be many ADMS installs, one for each workgroup.

AVFS – Autodesk Vault File Server. The Vault server component that contains the file store. In a multi-site environment, there would be many AVFS installs, one for each site.

Entity – A "base class" for the major currency types in Vault. Usually entity means file, folder, item, change order, or custom object.

Vault API – The full set of programming interfaces supported by Autodesk. It includes the VDF and the web service API.

VDF – Vault Development Framework. Value-add content for API developers, built on top of the web service API.

Web Service API – The low level API that talks directly to a Vault server through HTTP.

Working Folder – The location on a local computer that maps to a location in the Vault.

VDF Components

There are 4 DLLs that make up the VDF. These DLLs can be found in the "bin" directory of the SDK. VDF DLL's are identified by the "Autodesk.DataManagement" prefix. Vault-specific functionality will have "Vault" in the name, and UI-specific functionality will have "Forms" in the name.

Autodesk.DataManagement.Client.Framework.dll – The base workflow functionality.

Autodesk.DataManagement.Client.Framework.Forms.dll – The base UI functionality.

Autodesk.DataManagement.Client.Framework.Vault.dll – The Vault workflow functionality.

Autodesk.DataManagement.Client.Framework.Vault.Forms.dll – The Vault UI functionality.

When using the VDF you will usually reference to all 4 DLLs if any VDF UI components are involved. If you not using UI components, then just reference the 2 non-Forms DLLs. When deploying an EXE, you should deploy all referenced VDF DLLs. If you have a plug-in, you usually don't need to deploy any SDK DLLs because the hosting app should already have the VDF DLLs available.

DevExpress DLLs

The SDK bin directory also contains a bunch of DLLs with the "DevExpress" prefix. These contain third-party UI components used by the VDF. If your application does not use VDF UI, then you can safely ignore these DLLs. But if UI is used, then you may need to deal with DevExpress DLLs.

If you have the Vault client installed, these DLLs will be available in the GAC. If you do not have the Vault client installed, you will need to reference these DLLs explicitly in your project. The same rules apply when deploying. If the user has the Vault client, then you don't need to redistribute these DLLs. If the user does not have the Vault client, you need to redistribute the DLLs.

It's easiest to reference and redistribute all DevExpress DLLs in the SDK bin directory. There is no easy way to determine which exact DLLs you need.

Demo Application

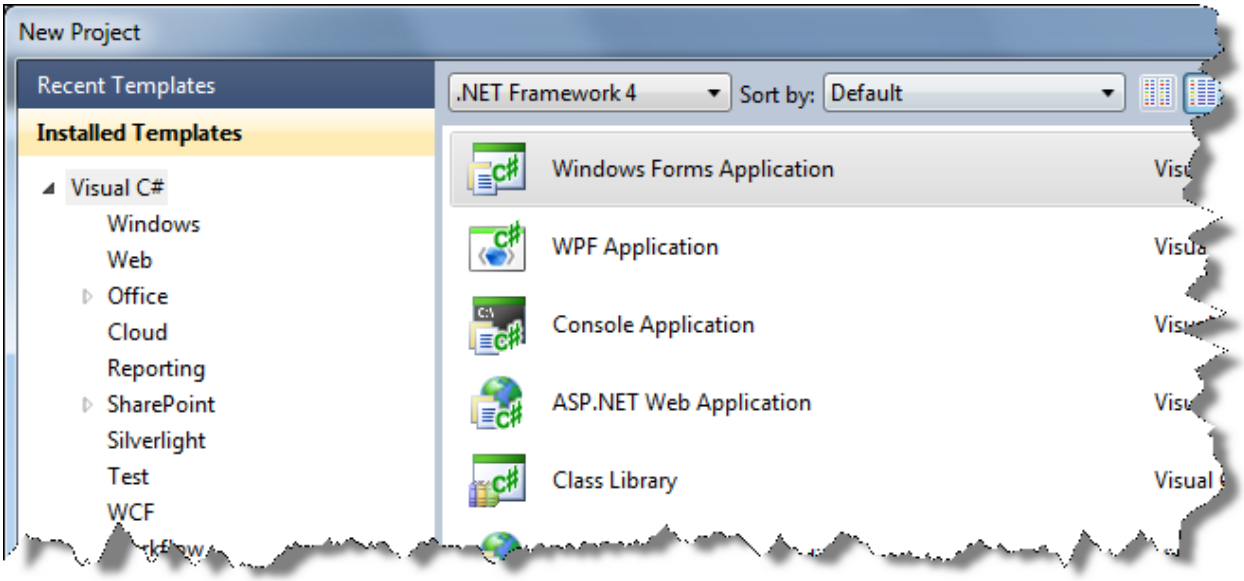
This is a step-by-step guide to creating an EXE that leverages key components on the VDF. The app will let the user log in to vault, browse the Vault, select a file and download/open a file by double clicking on it.

I'll be using Visual Studio 2010 for this example, but the steps should be the same for later editions. The code samples are in C#, but it should be easy to follow for VB developers. There are also several tools available online for converting C# and VB.NET code.

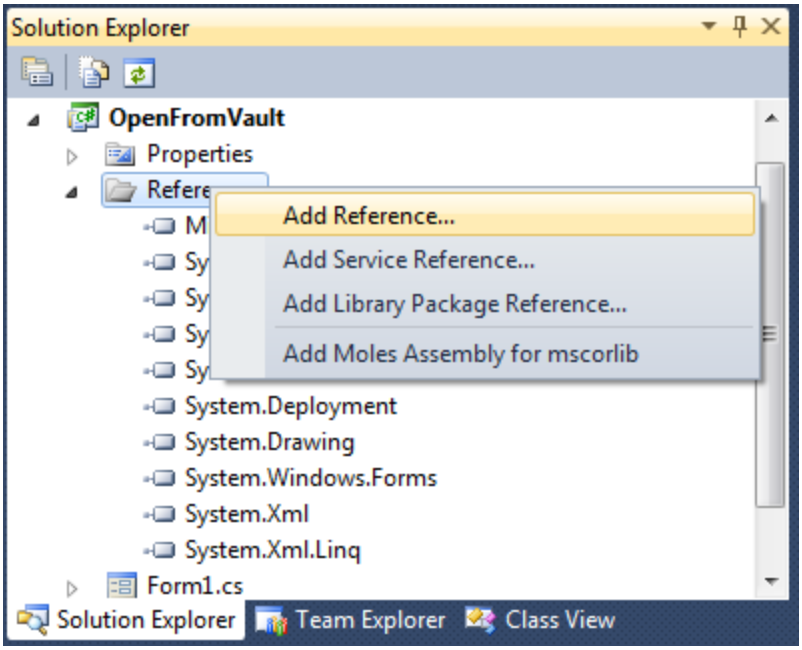
This application can be found in the Additional Class Materials .zip file.

Configuring the Project

Startup Visual Studio and create a new project. Choose a **Windows Forms Application** type. VDF UI is built on Windows Forms, not WPF. The VDF is also not compatible with Windows App Store development. Use .**NET Framework 4** since that is the version the VDF 2014 DLLs were built with.



Once the project is created, the next step is to set the references. Go to the **Solution Explorer**, right-click the **References folder** and select **Add Reference**.

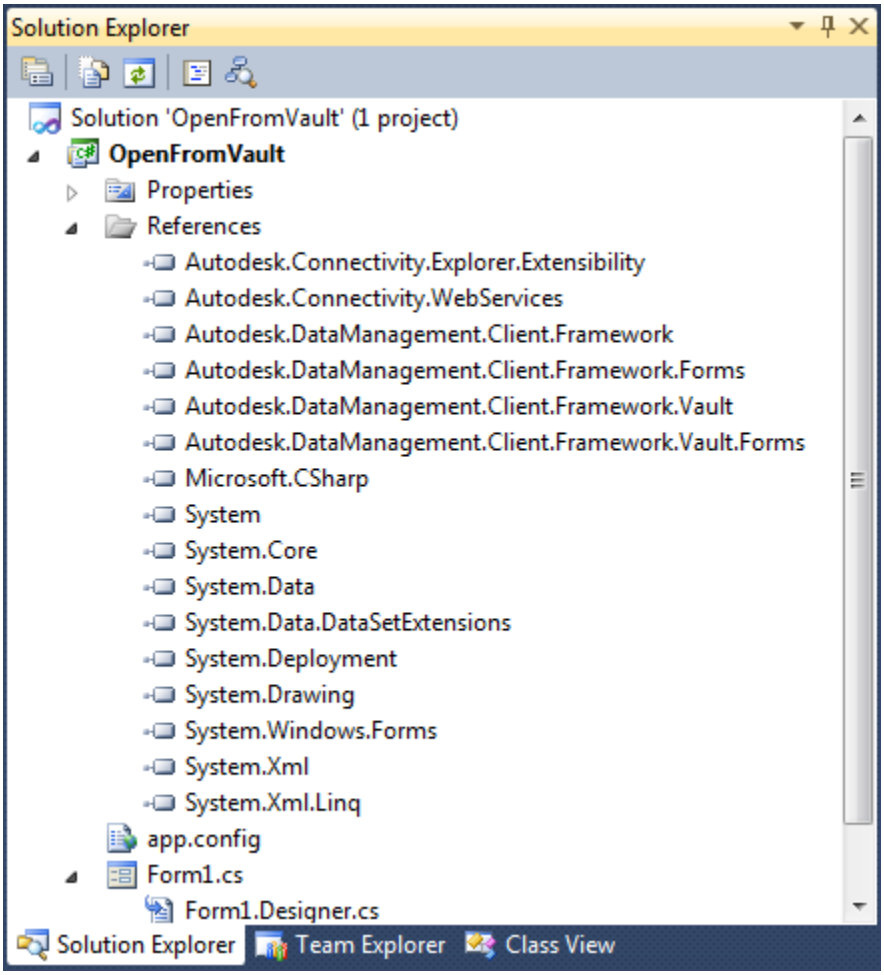


In the **Add Reference** dialog, click on the **Browse** tab. Navigate to the **bin** directory of the SDK folder and select the following 6 DLLs:

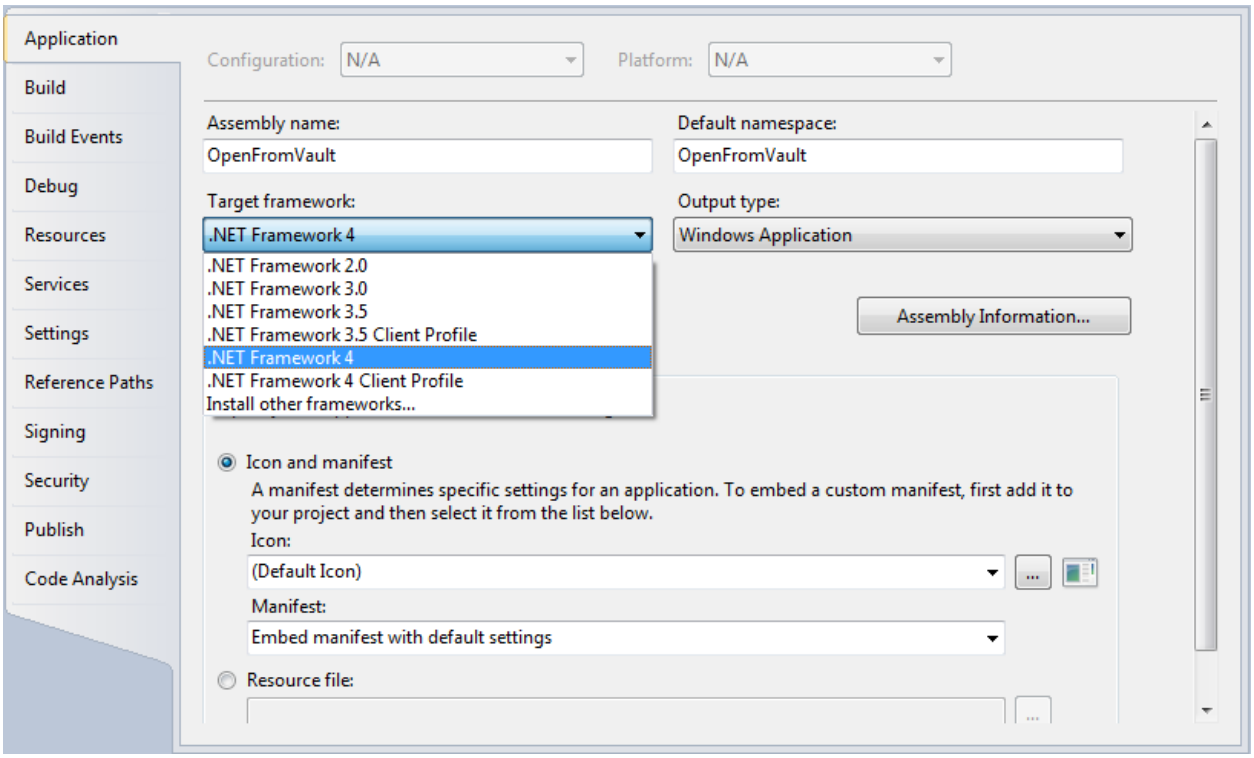
- Autodesk.Connectivity.Extensibility.Framework.dll
- Autodesk.Connectivity.WebServices.dll
- Autodesk.DataManagement.Client.Framework.dll
- Autodesk.DataManagement.Client.Framework.Forms.dll
- Autodesk.DataManagement.Client.Framework.Vault.dll
- Autodesk.DataManagement.Client.Framework.Vault.Forms.dll

By default, the SDK folder is *C:\Program Files (x86)\Autodesk\Autodesk Vault 2014 SDK*. If you don't have that folder, you may need to install the SDK. Go to your Vault client install folder (ex. *C:\Program Files\Autodesk\Vault Professional 2014*), go to the SDK folder and run the install.

Once you have your references in place, it should look something like this:



If you are having problem compiling, go to your project settings an make sure that the target framework is **.NET Framework 4**. Do not use a framework that says “Client Profile”.



Logging in to Vault

Visual Studio will create a default dialog, Form1, but you can delete it. All UI will be coming from the VDF.

Inside your **Program.cs** file. Set up your using statements (Imports in VB). Here are the statements I will be adding for this example.

```
using System.Diagnostics;
using Autodesk.DataManagement.Client.Framework.Vault.Currency.Connections;
using Autodesk.DataManagement.Client.Framework.Vault.Currency.Entities;
using Autodesk.DataManagement.Client.Framework.Vault.Currency.Properties;
using VDF=Autodesk.DataManagement.Client.Framework;
```

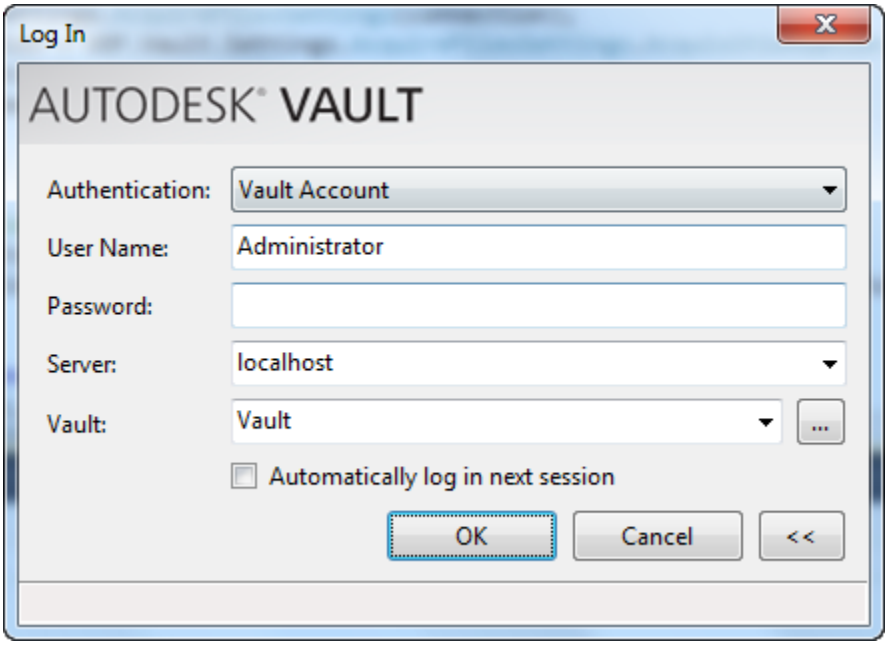
Inside my main function, comment out all the Form1 stuff. So that it looks like:

```
[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    //Application.Run(new Form1());

    // -- put program logic here --
}
```

The first part of the program logic is to log in. Popping up the VDF login dialog is easy. The below code will log the user in. If the return value is non-null the user is properly logged in.

```
// log in
Connection connection = VDF.Vault.Forms.Library.Login(null);

if (connection != null)
{
    // -- put Vault operations here --
}
```

When run, the user will see the standard Vault login dialog.



Selecting a File in Vault

Now that you are logged in, the next step is to browse the Vault for a file. The **SelectEntity** dialog does most of the work for you. You just have to worry about the settings.

The dialog uses a grid, and you need to set the initial state of the grid. The user can change the grid settings, but you need a starting configuration. At a minimum, the grid configuration needs a persistence key (to remember the user settings), a view column and a sort column. You can use any property for the view/sort settings, but this example will use EntityIcon and EntityName, which are out-of-the-box properties.

```
var config = new VDF.Vault.Forms.Controls.VaultBrowserControl.Configuration(connection,
    "Autodesk.OpenFromVaultSample.grid", null);
config.AddInitialColumn(PropertyDefinitionIds.Client.EntityIcon);
config.AddInitialColumn(PropertyDefinitionIds.Server.EntityName);
config.AddInitialSortCriteria(PropertyDefinitionIds.Server.EntityName, true);
```

The EntityName is the display name for the object. The EntityIcon is the object icon, which helps the user can distinguish between files and folders.

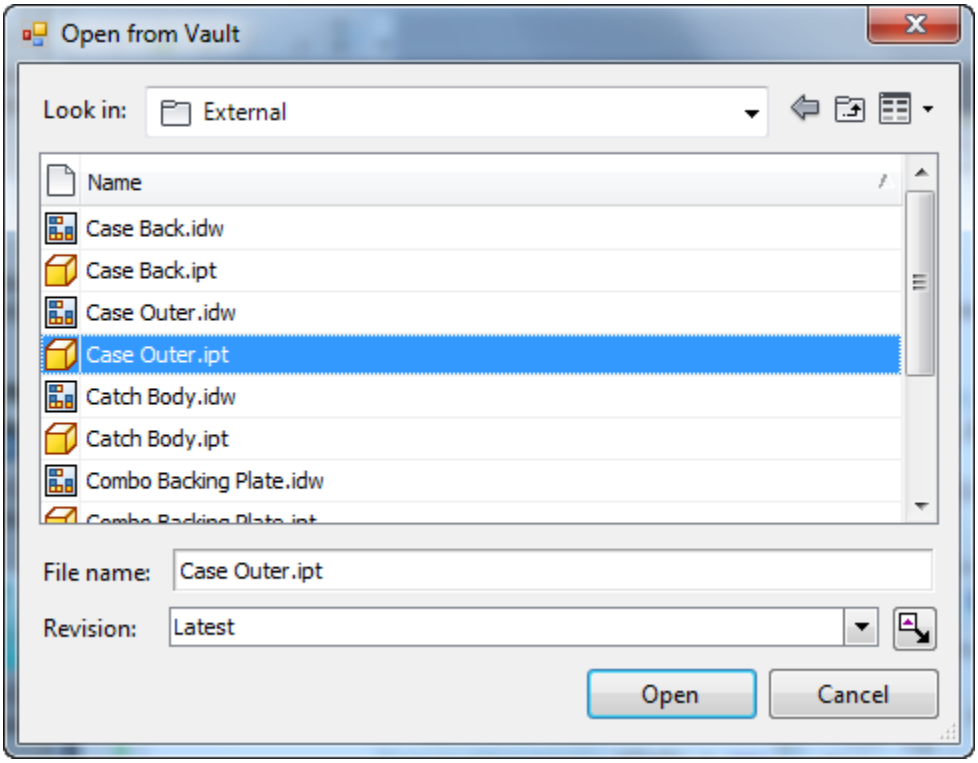
There is another settings object for the dialog itself. It has its own persistence key and contains a reference to the grid. For this example, the user should only be selecting Files, so ActionableEntityClassID is set to Files.

```
var settings = new VDF.Vault.Forms.Settings.SelectEntitySettings();
settings.PersistenceKey = "Autodesk.OpenFromVaultSample";
settings.ActionableEntityClassID = EntityClassIds.Files;
settings.GridConfig = config;
```

Now you are ready to invoke the dialog. If the user selects a file, you will get a non-null result back.

```
var results = VDF.Vault.Forms.Library.SelectEntity(connection, settings);

if (results != null && results.SelectedEntities.Any())
{
    // -- do operations on the selection --
}
```

When run, the user will see a dialog that lets them navigate the Vault folders and select a file.



Downloading and Opening the File

The user has selected the file, now it is time to download it. **AcquireFiles** is the function you want to use. As usual, some settings are required and results are returned. AcquireFiles can be used to download and/or checkout, so you need to set **DefaultAcquisitionOption** to just Download. You specify which file(s) to download by calling the **AddEntityToAcquire** function.

```
// download the file
var settings = new VDF.Vault.Settings.AcquireFilesSettings(connection);
settings.DefaultAcquisitionOption =
    VDF.Vault.Settings.AcquireFilesSettings.AcquisitionOption.Download;
settings.AddEntityToAcquire(entity);
var results = connection.FileManager.AcquireFiles(settings);
if (results.FileResults.Any())
{
    // -- perform operations on downloaded file --
}
```

This AcquireFiles call doesn't specify a download location, so it defaults to the working folder. The VDF provides an easy way to look up that location by using the **WorkingFoldersManager** on the Connection object.

```
// figure out where the file was downloaded on disk
FileIteration file = results.FileResults.First().File;
string localPath =
    connection.WorkingFoldersManager.GetPathOfFileInWorkingFolder(file).FullPath;
```

Lastly you open the file using the default application associated with that file extension. This code doesn't use the VDF. It uses the System.Diagnostics library in .NET.

```
// Open the file
ProcessStartInfo pInfo = new ProcessStartInfo();
pInfo.FileName = localPath;
Process p = Process.Start(pInfo);
```

That's it for the demo app. It's a good starting point for using the VDF. This example used the default settings for most things, but there is a lot of additional functionally available to you by simply adjusting the settings.

Useful Links

It's All just Ones and Zeros
http://justonesandzeros.typepad.com/
A blog just for the Vault and PLM 360 APIs. Updates weekly. Sample apps every month, usually with source code.

ADN Manufacturing DevBlog
http://adndevblog.typepad.com/manufacturing/
A blog for the Autodesk Developer Network to post tips, tricks and sample code. Covers all Autodesk Manufacturing products, including Vault.

Vault Online Help
http://help.autodesk.com/view/VAULT/2014/ENU/
The official help documentation for Vault. Contains a section on using the VDF.

Vault Customization Discussion Group
http://forums.autodesk.com/t5/Vault-Customization/bd-p/301
A discussion group for Vault API developers. If you get stuck, this is where you go.