

SD321955

Autodesk Vault 2020 -- Programming 101

Markus Koechl
Autodesk – Central Europe

Jeffrey Fishman
Autodesk Inc.

Learning Objectives

- Vault API – Overview and Introduction
 - Understand different concepts accessing Vault software's programming interface for customization and automation
- Vault 2020 SDK – Templates
 - Learn how to leverage the new SDK templates to get done your first custom application, job and client extension quickly.
- Vault Inventor Server
 - Get insights on programming custom jobs using VaultInventorServer capabilities.

Description

Get started programming Vault Workgroup or Vault Professional applications, extensions, and custom jobs. The Vault 2020 Software Development Kit (SDK) shares new templates and sample code removing barriers accessing the entry-level in Vault Web Services, Vault Job Processor, and Vault Client programming. This class discusses real-life automation and extension tasks and guides the reader through the steps required to solve them.

Your AU Experts

Markus Koechl is a Solution Engineer for Vault Products. He targets customer needs, practical workflows, and is always eager to overcome barriers by extensions or automation. That's the simple reason that he started programming Inventor, Inventor iLogic, and Vault APIs with the background of a Mechanical Engineer.

Jeffrey Fishman is a Software Engineer for the Autodesk Vault product. He is driven by creative strategy and exploration -- seeking to understand and ultimately improve upon current methodologies and workflows in place today, for a more focused and streamlined tomorrow.

Contents

Autodesk Vault 2020 -- Programming 101.....	1
1 Introduction.....	5
1.1 Target Audience and Target Workflow.....	5
1.2 Class Materials	5
1.3 Setup Working Environment	6
1.3.1 Install Vault SDK.....	6
1.3.2 Install Visual Studio 2017.....	6
1.3.3 Install Class Templates.....	7
1.4 Autodesk Vault API Overview	7
1.4.1 General Access Using Webservices	8
1.4.2 Job Processor API	8
1.4.3 Web Services API – Event Handling	8
1.4.4 The Role of Vault Data Standard – Another API?	9
2 Hands-On Automation Sample.....	12
2.1 Use Case 1 Description	12
2.2 Use Case 1 Solution Steps	12
2.3 PowerShell Solution.....	12
2.3.1 Solution Step 1 – Create New Script.....	13
2.3.2 Licensing	14
2.3.3 Solution Step 2 – Load the Webservices Assembly	14
2.3.4 Solution Step 3 – Connect to Vault	15
2.3.5 Solution Step 4 – Execute the task	16
2.3.6 Solution Step 5 – Debugging, Error Handling and Return Values.....	19
2.4 C# Solution	20
2.4.1 Solution Step 1 – Create New Script.....	20
2.4.2 Licensing	20

2.4.3	Solution Step 3 – Connect to Vault	22
2.4.4	Solution Step 4 – Execute the task	23
2.4.5	Solution Step 5 – Debugging, Error Handling and Return Values.....	25
2.4.6	Alternate Solution – Avoid coding/scripting of user credentials	26
2.5	Use Case 1 Summary	26
3	Hands-On – Use Case 2 Job Handler Extension.....	27
3.1	Use Case 2 Description	27
3.2	Use Case 2 Solution Steps	27
3.2.1	Solution Step 1 – Create a New Custom Job	28
3.2.2	Validate Job Registration	28
3.2.3	Job Registration – Trouble Shooting	31
3.2.4	Solution Step 2 – Establish Job Handler Debugging	31
3.2.5	Solution Step 3 – Coding the Job’s Task.....	33
3.2.6	Solution Step 4 – Complete Error Handling.....	38
3.3	Use Case 3 Takeaways	38
4	Hands-On – Use Case 3 Event Handler.....	39
4.1	Use Case 3 Description	39
4.2	Use Case 3 Solution Steps	39
4.2.1	Solution Step 1 – Create New Event Handler “Add Folder”	40
4.2.2	Solution Step 2 – Coding	41
4.2.3	Solution Step 3 – Adding Restrictions to Move Folder Events.....	45
4.3	Use Case 3 Summary	46
5	Outlook	47

1 Introduction

Read this Handout to review all examples used and gather supplementary information in detail. The Download of “Additional Materials” shares the final solution of all samples and one more solution template.




1.1 Target Audience and Target Workflow

This class addresses administrators, consultants, applications engineers, and programmers experienced in codings against one or more Autodesk APIs, like Inventor iLogic, AutoCAD vLisp, or Inventor or AutoCAD .NET add-ins. Do you belong to this group, and are you eager to include Vault programming and expect some guidance about where to start? Or have you already tried building a solution based on SDK samples, but failed to get these stripped down to your essential requirement?

The approach and material of this class should lower the entry-level and get you started.

We will focus on general principles to get each project type set up, registered for use and accessible in debug mode, before writing code. Frequently I am getting asked to review “huge” projects to get these loaded or executed in Vault clients. And sometimes, we also are asked how to run the SDK sample projects. Because of these experiences, we are using the order – Define new project using the best matching template framework -> Compile and familiarize debugging -> Start coding.

1.2 Class Materials

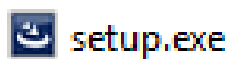
-  [API-Onboarding-PowerShell.ps1](#)
-  [API-Onboarding-PowerShell.zip](#)
-  [VStudio Solutions-UseCases1to3.zip](#)

1. Follow the presentation reviewing all samples code complete. The **VStudio Solutions-UseCases1to3.zip** shares a separate project for each sample.
2. Follow the hands-on chapters of this document and create new projects consuming the template shared in **API-Onboarding-PowerShell.zip** for Sample-01. Sample-01-CSharp, Sample-02, and Sample-03 base on templates of the SDK. Reusing these in your future coding at least will allow repeating the successful setup for each extension type.

1.3 Setup Working Environment






1.3.1 Install Vault SDK

Each Vault Client installation includes the full SDK containing API Help Documentation, samples, and tools to support you writing custom extensions, jobs, or applications for Vault. Note – The Autodesk Vault API requires no additional license in general; Vault API and SDK also include 3rd party libraries from DevExpress. These DLLs are free to use and redistribute within the context of the Vault API. However, direct use of the DLLs requires a developer license; in this case, buy permits from DevExpress (<http://www.devexpress.com/>).



Navigate to **C:\Program Files\Autodesk\Vault Professional 2020\SDK\Setup.exe** to run the SDK setup.

The SDK and its documentation can be found in **C:\Program Files\Autodesk\Autodesk Vault 2020 SDK\docs\VaultSDK.chm**.

This PC > Windows (C:) > Program Files > Autodesk > Autodesk Vault 2020 SDK				
<input type="checkbox"/> Name	Date modified	Type	Size	
 Autodesk Templates	03/03/2019 18:05	File folder		
 bin	03/03/2019 18:05	File folder		
<input checked="" type="checkbox"/>  docs	03/03/2019 18:05	File folder		
 VS17	03/03/2019 18:05	File folder		
 EULA.rtf	15/02/2019 03:27	Rich Text Format	50	

1.3.2 Install Visual Studio 2017

Visual Studio 2017 is available in free and commercial editions. All samples are written in C#.

Sample-01 offers two solutions: A PowerShell project and a C#



PowerShell Tools for Visual Studio 2017

A set of tools for developing and debugging PowerShell scripts and modules in Visual Studio.

project. To open .\Sample-01\Sample-01.sln you need Visual Studio with PowerShell Tools enabled.

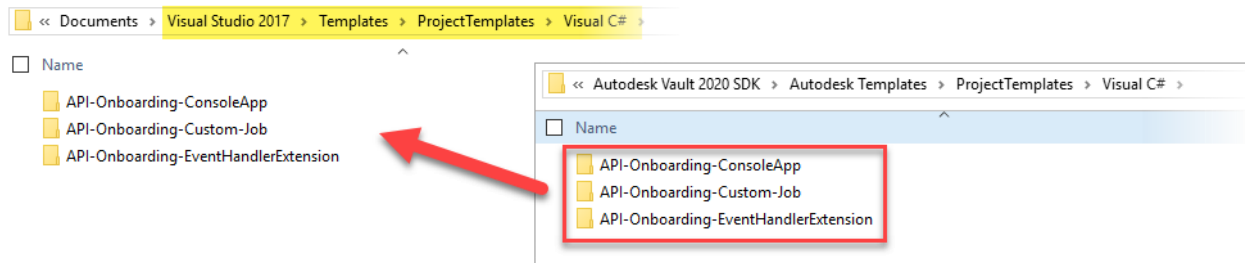
Alternatively, use Visual Studio Code Editor (Free), with PowerShell extension. This editor is smart, offering IntelliSense and comfortable access/view of variables in debug mode.



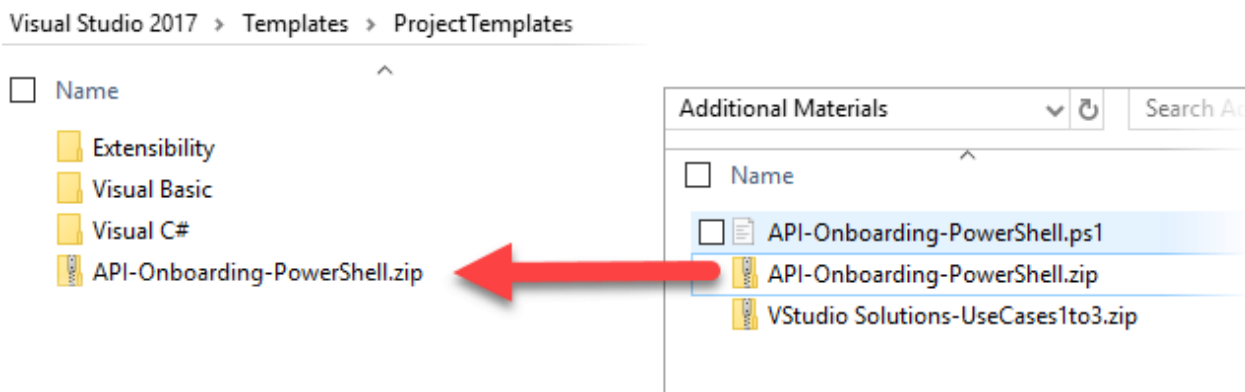
PowerShell
Microsoft

1.3.3 Install Class Templates

Copy the SDK templates to your Visual Studio installation's ProjectTemplates folder:

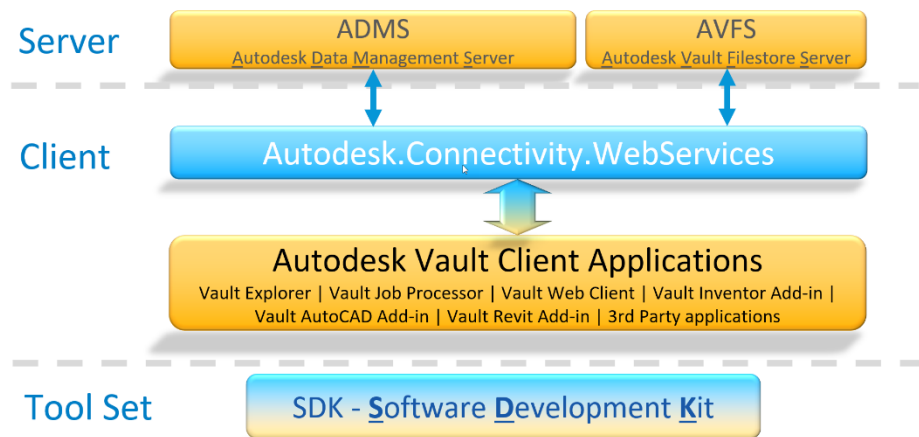


Add the PowerShell project template from this class “Additional Material” download:



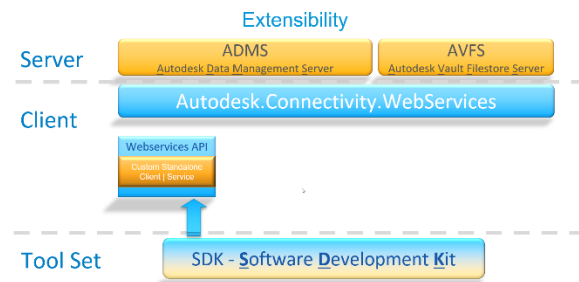
1.4 Autodesk Vault API Overview

Vault is a client/server architecture; the access to server components – data management server and filestore server leverages client-side web services. It requires either a Vault Client installed or the re-use of SDK libraries for independent applications.



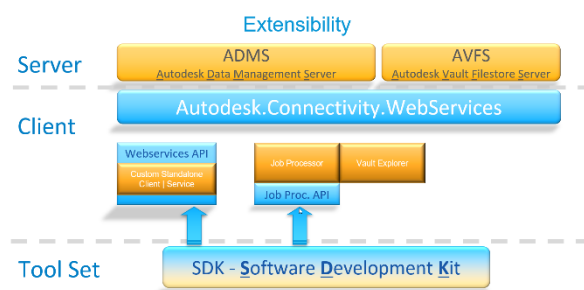
1.4.1 General Access Using Webservices

Hands-On / Use Case 1 leverages the main SDK library to use PowerShell scripts or C# code for Vault connections and executing folder create tasks in Vault.



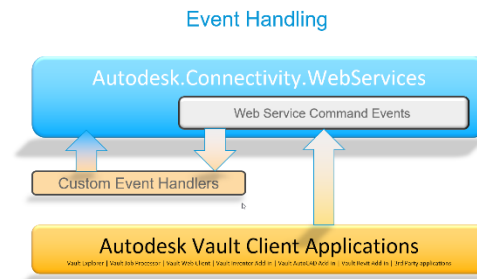
1.4.2 Job Processor API

Hands-On / Use Case 2 creates an extension adding a custom job handler; the job's task is creating and saving Thin Client links initiated on lifecycle changes.



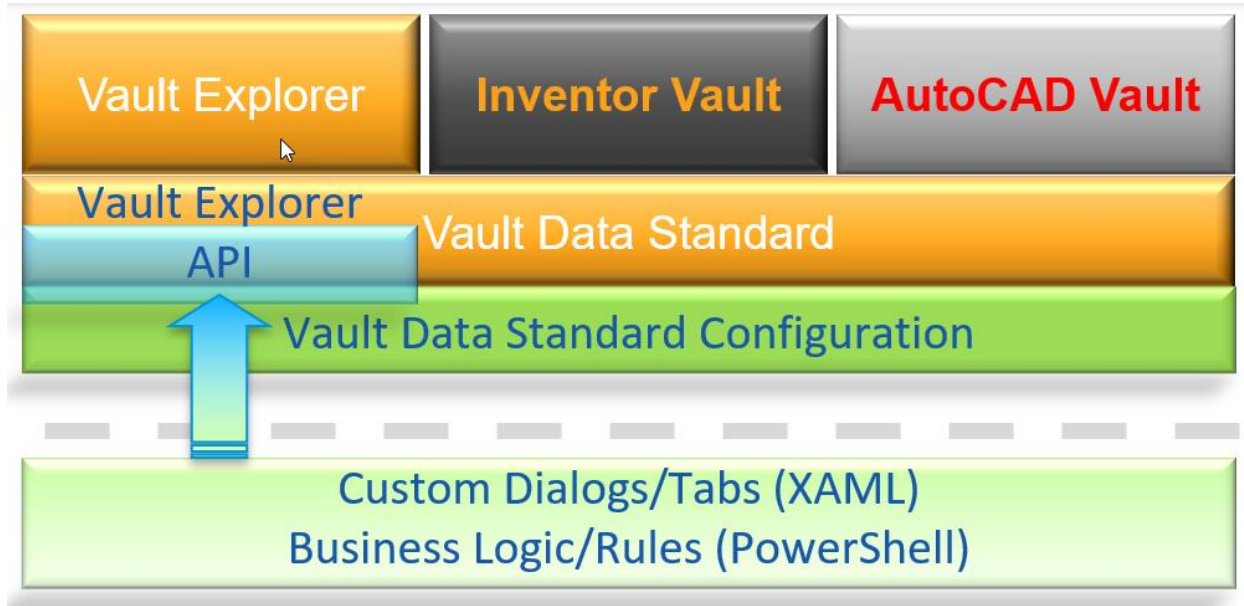
1.4.3 Web Services API – Event Handling

Hands-On / Use Case 3 implements an extension to leverage the event of adding new folders. The solution introduces how to add Thin Client links to project folders on the creation and how to prevent users from moving a project.



1.4.4 The Role of Vault Data Standard – Another API?

Vault Data Standard (VDS) installs as extensions for Vault Explorer, Inventor Vault and AutoCAD/AutoCAD Mechanical Vault.



VDS extends Vault Explorer by additional commands and workflows:

- Create a new file in the current folder consuming vaulted templates
- Create a folder with properties
- Extend custom commands, e.g., create a project folder with a subfolder structure
- Each control and dialog allow to apply custom rules, e.g., to enforce input or to autofill properties.

VDS also extends the user interface by customizable dialogs and tabs:

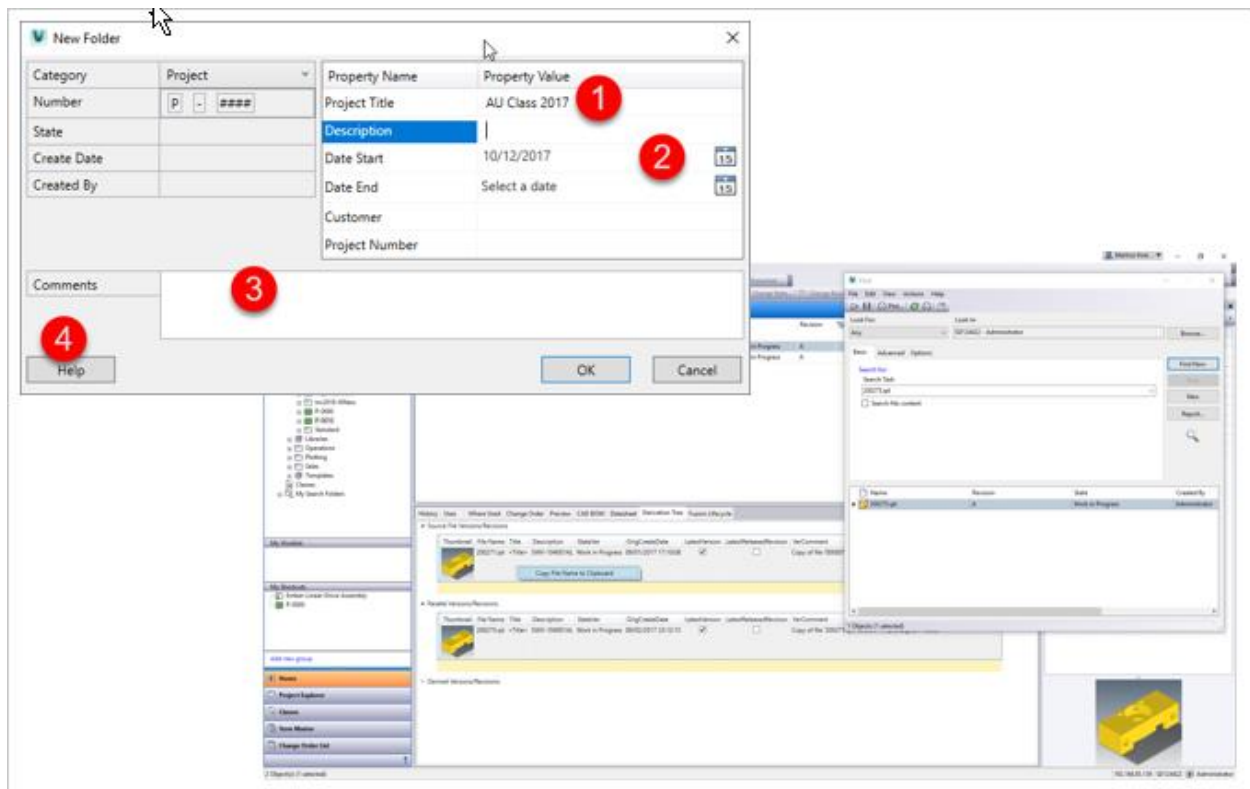
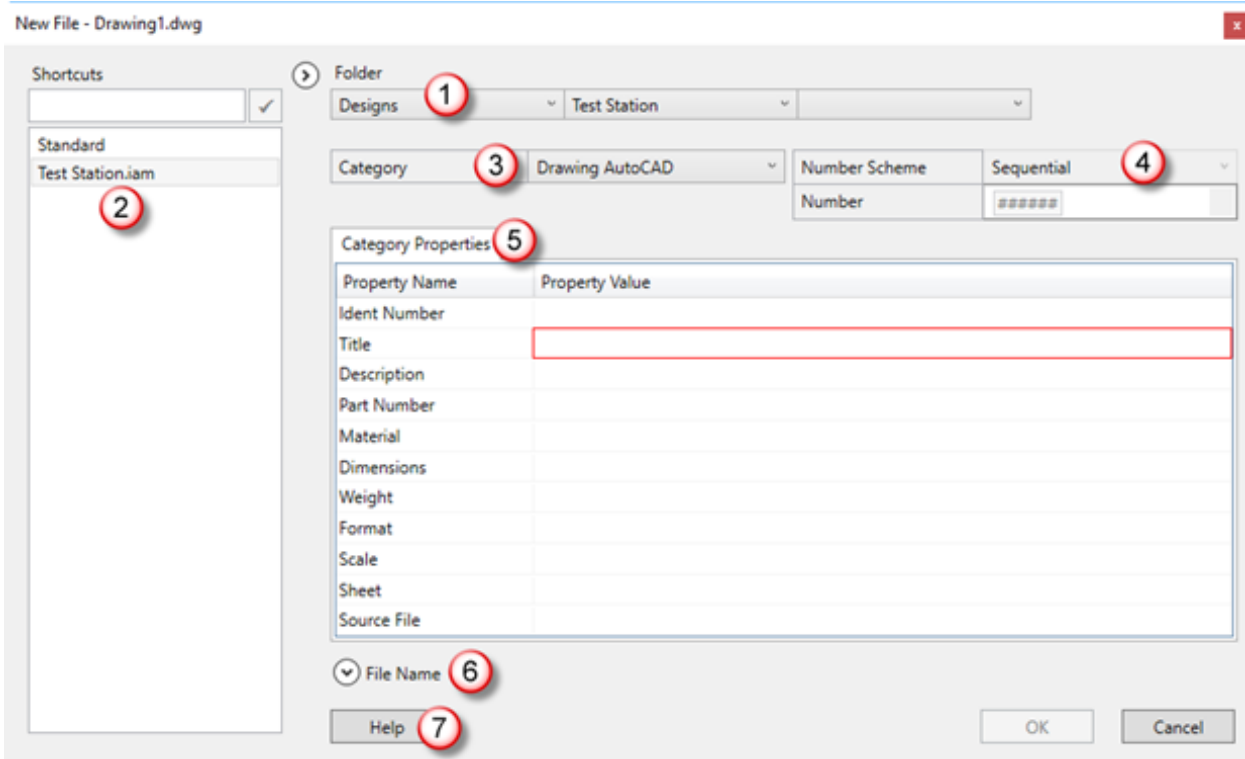
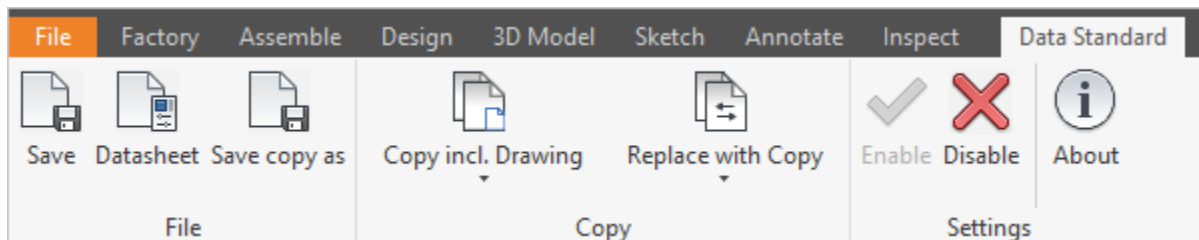


Figure 1 - Vault Data Standard UI Samples

For CAD applications Inventor and AutoCAD/AutoCAD Mechanical VDS shares custom dialogs on save events; handling it, VDS not only offers the dialog but also checks against custom rules for metadata.



And last but not least, VDS for Inventor also adds workflow commands, e.g., to replace components by new copy/new copy including drawing and others, that usually require a series of commands and tasks.



The primary purpose of VDS is to apply company standards to workflows enforcing rules and guidance handling metadata and file locations.

To adopt standards, customize rules, and user interface VDS uses Microsoft WPF dialog and UI elements (Windows Presentation Foundation – WPF) plus PowerShell scripting to implement custom rules. The dialog templates and scripts consume Vault Web Services API; so, we see VDS more as an addition customization layer than API access. However – advanced scripting allows using Vault API Web Services API libraries as well.

2 Hands-On | Automation Sample

2.1 Use Case 1 | Description

Workflows frequently span multiple systems. E.g., engineering projects can get kicked off by an order submitted within a CRM system. Our task is to allow any system to start a command creating new project folders in Vault.

2.2 Use Case 1 | Solution Steps

Use Case 1 Solutions Steps are available for PowerShell and C# languages. The overall procedure of connecting to Vault, executing a task, and releasing the connection are the same. The prerequisites and files to include for distribution to other computers are different.

Use Case 1 | Solution Options

Use Case 1 | PS Solution

- PowerShell Script Editor
 - New PowerShell Project¹ or
 - Open "API-Onboarding-PowerShell.ps1" ²
- Load the .NET Vault API Assembly
- Write code to...
 - Connect to Vault
 - Execute "Create Project Folder"
 - Disconnect
- Debug...if needed



Use Case 1 | C# Solution

- Visual Studio IDE
 - New API-Onboarding Console Application
- Done - Reference the .NET Vault API Assembly
- Write code to...
 - Connect to Vault
 - Execute "Create Project Folder"
 - Disconnect
- Debug...if needed

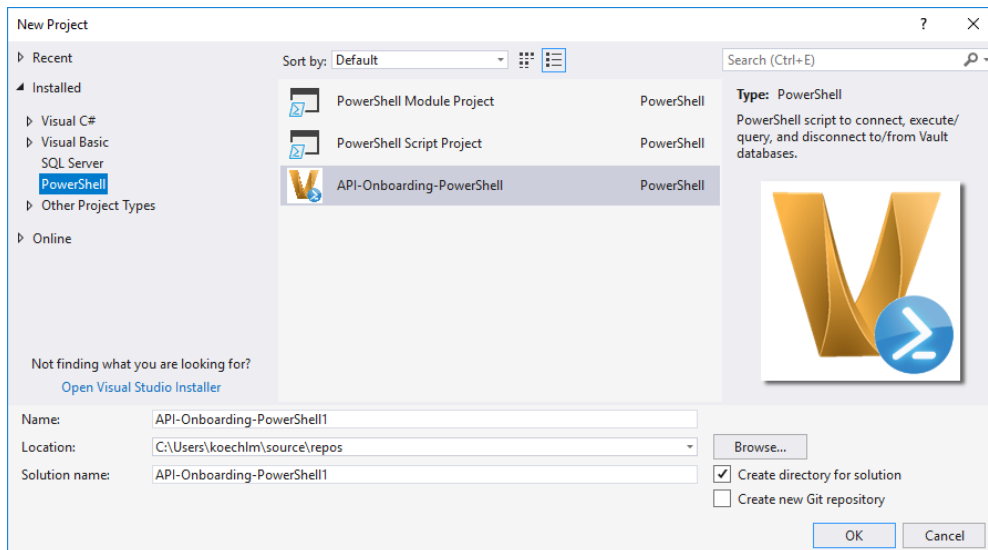


2.3 PowerShell Solution

Before you start using PowerShell, ensure that the execution policy allows running unsigned scripts on the client.

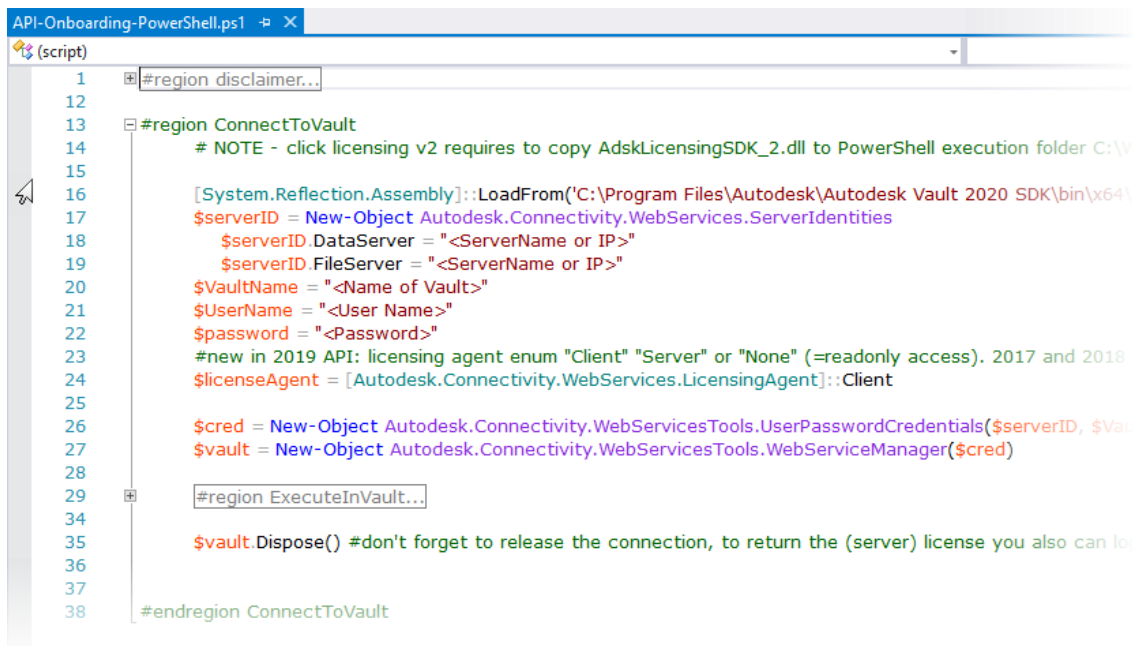
2.3.1 Solution Step 1 – Create New Script

Start a new PowerShell project selecting the script template installed before¹.



Alternatively, open the template script .\Additional Material\ API-Onboarding-PowerShell.ps1 in VSCode and save as a new copy.

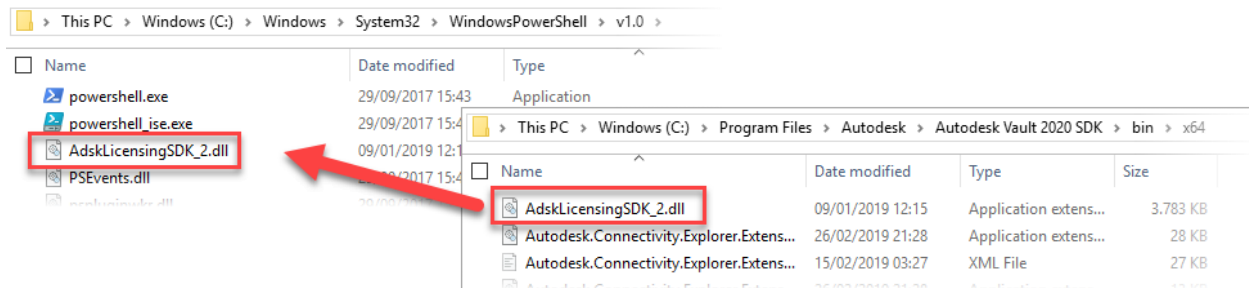
Reviewing the existing regions in the script Vault-Script.ps1 file, you quickly will recognize areas for implementation (#region) but also code to establish the connection.



¹ To install class templates review instruction in chapter 1.3.3

2.3.2 Licensing

PowerShell runtime works as the hosting application while a script executes. To enable Vault access, we need to distribute the Vault SDK licensing library alongside the primary executable. You need to copy the file AdskLicensingSDK_2.dll to PowerShell execution folder C:\Windows\System32\WindowsPowerShell\v1.0 on any computer running the script.

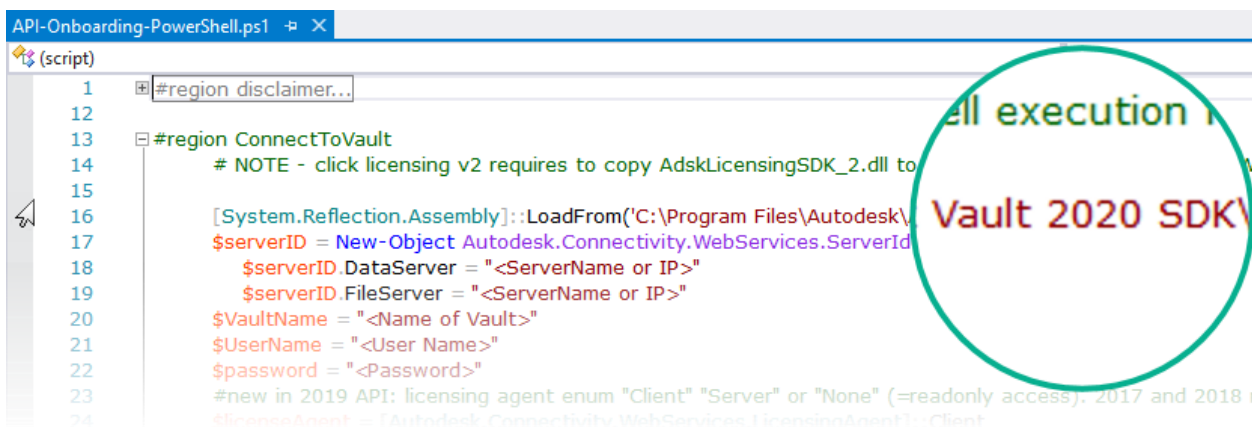


Any license type requires this step. We select the license type in section “Connect to Vault”.

2.3.3 Solution Step 2 – Load the Webservice Assembly

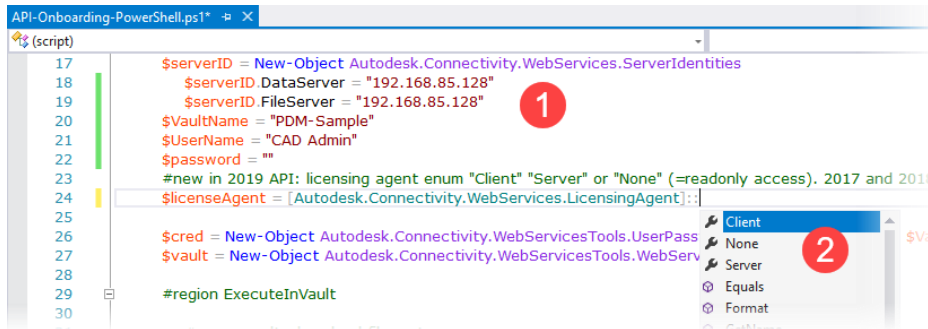
Vault WebServices are the core – you surely remember the overview discussed before. We need to load the assembly first to interact with Vault.

Check the Vault version in case the template applies to futures releases of Vault SDK.



2.3.4 Solution Step 3 – Connect to Vault

Specify the connection parameters (1) and the license type (2):



```

17 $serverID = New-Object Autodesk.Connectivity.WebServices.ServerIdentities
18 $serverID.DataServer = "192.168.85.128"
19 $serverID.FileServer = "192.168.85.128"
20 $VaultName = "PDM-Sample"
21 $UserName = "CAD Admin"
22 $password = ""
23 #new in 2019 API: licensing agent enum "Client" "Server" or "None" (=readonly access). 2017 and 2018
24 $licenseAgent = [Autodesk.Connectivity.WebServices.LicensingAgent]::Client
25
26 $cred = New-Object Autodesk.Connectivity.WebServicesTools.UserPass
27 $vault = New-Object Autodesk.Connectivity.WebServicesTools.WebServiceManager($cred, $VaultName)
28
29 #region ExecuteInVault
30

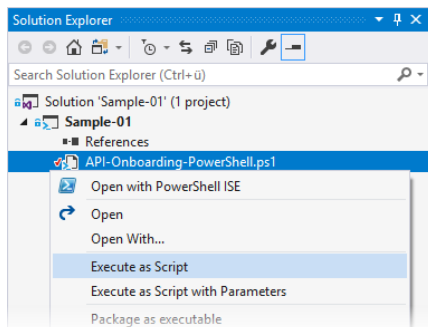
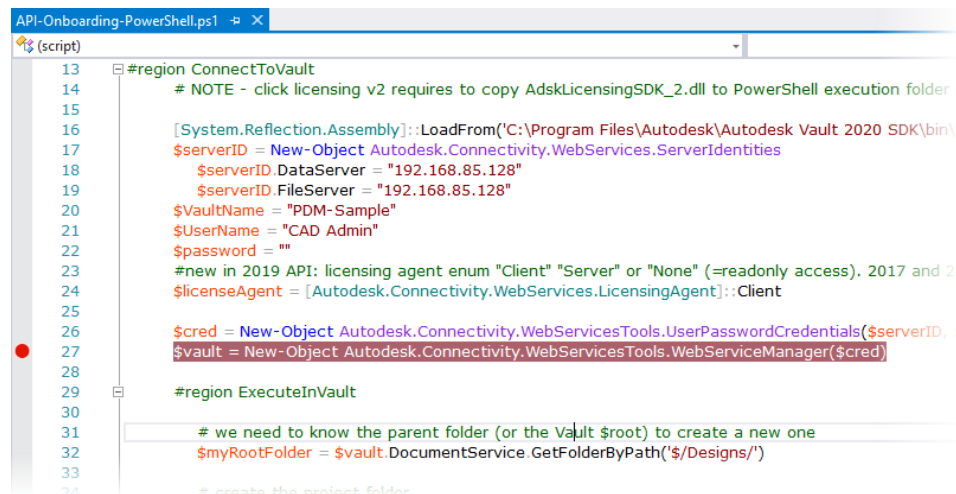
```

The screenshot shows a PowerShell script editor with a dropdown menu open for the `$licenseAgent` property. The dropdown lists "Client", "None", "Server", "Equals", and "Format". The "Client" option is selected, indicated by a red circle with the number 2. A red circle with the number 1 points to the server ID configuration lines.

Note – Selecting the license type “Client” requires a Vault Client installed on your machine and activated. Autodesk Licensing allows computer-based activation (single user) or network licensing. Any application needs to include the licensing library to communicate either with the local (computer) license service or the network license server.

If you are concerned about sharing user/password information in scripts, you find an alternate solution approach reading chapter 2.4; it is applicable in PowerShell as well.

Test your connectivity by executing selections or step into activating a breakpoint.

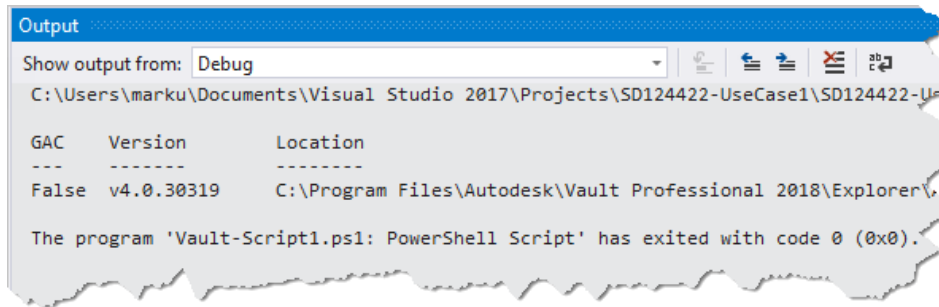
```

13 #region ConnectToVault
14 # NOTE - click licensing v2 requires to copy AdskLicensingSDK_2.dll to PowerShell execution folder
15
16 [System.Reflection.Assembly]::LoadFrom('C:\Program Files\Autodesk\Autodesk Vault 2020 SDK\bin\
17 $serverID = New-Object Autodesk.Connectivity.WebServices.ServerIdentities
18 $serverID.DataServer = "192.168.85.128"
19 $serverID.FileServer = "192.168.85.128"
20 $VaultName = "PDM-Sample"
21 $UserName = "CAD Admin"
22 $password = ""
23 #new in 2019 API: licensing agent enum "Client" "Server" or "None" (=readonly access). 2017 and 2018
24 $licenseAgent = [Autodesk.Connectivity.WebServices.LicensingAgent]::Client
25
26 $cred = New-Object Autodesk.Connectivity.WebServicesTools.UserPasswordCredentials($serverID, $password)
27 $vault = New-Object Autodesk.Connectivity.WebServicesTools.WebServiceManager($cred, $VaultName)
28
29 #region ExecuteInVault
30
31 # we need to know the parent folder (or the Vault $root) to create a new one
32 $myRootFolder = $vault.DocumentService.GetFolderByPath('$/Designs/')
33
34 # create the project folder

```

The screenshot shows a PowerShell script editor with a red dot indicating a breakpoint on line 27. The script defines a `#region ConnectToVault` and a `#region ExecuteInVault`. The `$licenseAgent` is set to `Client`. The `$cred` is created using `UserPasswordCredentials`. The `$vault` is created using `WebServiceManager`. The `$myRootFolder` is retrieved using `GetFolderByPath`.

In case of failure – review the output window:



```

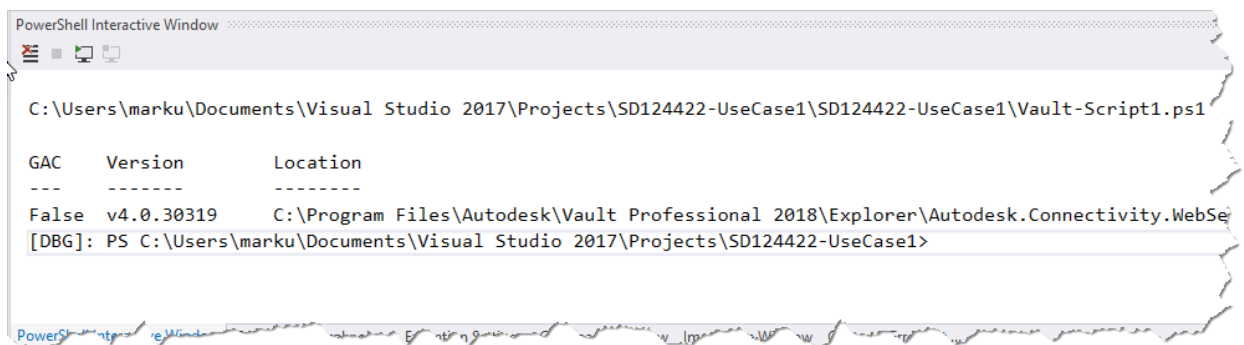
Output
Show output from: Debug
C:\Users\marku\Documents\Visual Studio 2017\Projects\SD124422-UseCase1\SD124422-UseCase1\Vault-Script1.ps1

GAC    Version    Location
---    -
False  v4.0.30319  C:\Program Files\Autodesk\Vault Professional 2018\Explorer\

The program 'Vault-Script1.ps1: PowerShell Script' has exited with code 0 (0x0).

```

or better the PowerShell Interactive Window:



```

PowerShell Interactive Window
C:\Users\marku\Documents\Visual Studio 2017\Projects\SD124422-UseCase1\SD124422-UseCase1\Vault-Script1.ps1

GAC    Version    Location
---    -
False  v4.0.30319  C:\Program Files\Autodesk\Vault Professional 2018\Explorer\Autodesk.Connectivity.WebService\

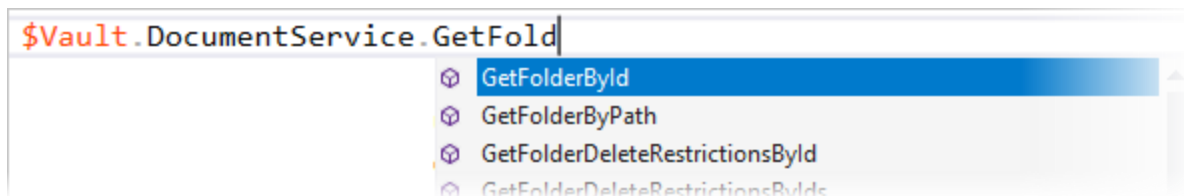
[DBG]: PS C:\Users\marku\Documents\Visual Studio 2017\Projects\SD124422-UseCase1>

```

2.3.5 Solution Step 4 – Execute the task

Starting to write the script, you might ask how to find the methods/method names required to perform our desired actions.

Once we have the WebServiceManager \$vault, things are getting more comfortable, as IntelliSense will help to find appropriate methods and properties.



```

$Vault.DocumentService.GetFold

```

To enable this, don't miss to run the code as far as it is.

Note – You can use Telerik's Fiddler extended by coolOrange's vapiTrace to record all Vault API calls of manually performed tasks in Vault. For installation and further information, visit the web pages of Telerik,

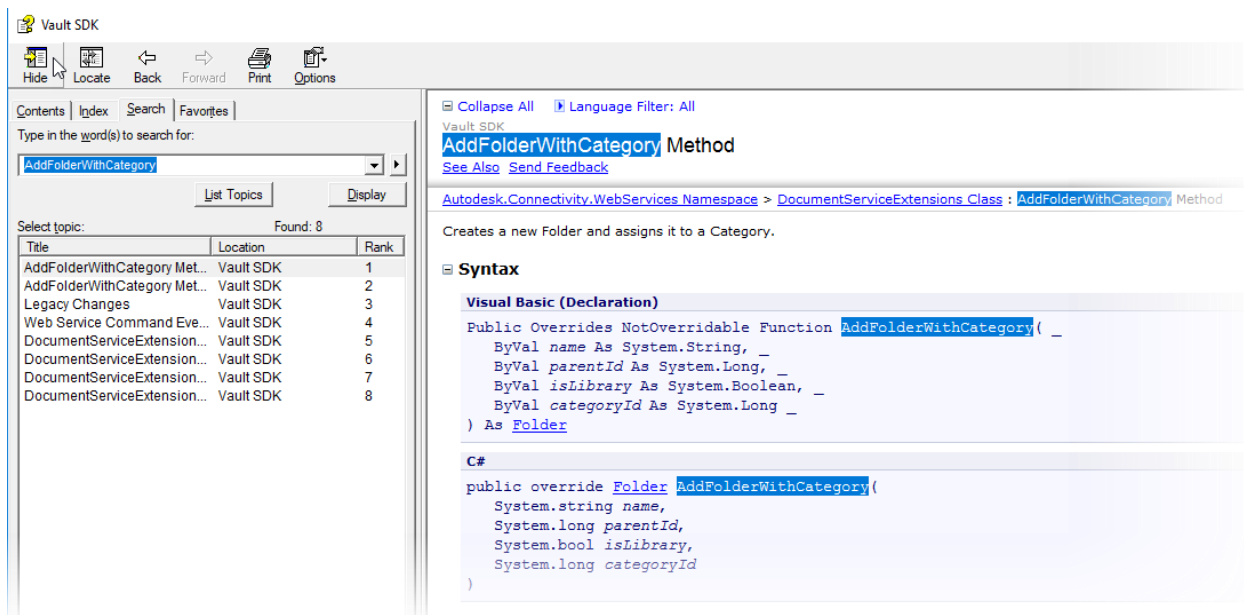
<http://www.telerik.com/fiddler>, and coolOrange, https://www.coolorange.com/en/apps_vault.php. A video tutorial in Vault Knowledge Network instructs how to configure and use this powerful tool with the sample adding a folder with the project category in Vault. The video is available here: <http://autode.sk/2ltSYYS>

To add a new project folder, we have two options:

- Add a new folder, then update the category
- Add a new folder with a given category

The class solution project Sample-01.sln contains both possibilities. Let's follow the second approach for now.

We use the method `AddFolderWithCategory`; compare the Help documentation to get detailed information about which service it belongs to, and about the input- and return values.



The screenshot shows the Vault SDK help interface. On the left, a search bar contains 'AddFolderWithCategory' and a table lists search results. On the right, the documentation for the 'AddFolderWithCategory' method is displayed, including its syntax in Visual Basic and C#.

Title	Location	Rank
AddFolderWithCategory Met...	Vault SDK	1
AddFolderWithCategory Met...	Vault SDK	2
Legacy Changes	Vault SDK	3
Web Service Command Eve...	Vault SDK	4
DocumentServiceExtension...	Vault SDK	5
DocumentServiceExtension...	Vault SDK	6
DocumentServiceExtension...	Vault SDK	7
DocumentServiceExtension...	Vault SDK	8

AddFolderWithCategory Method
 Autodesk.Connectivity.WebServices.Namespace > DocumentServiceExtensions.Class : AddFolderWithCategory Method
 Creates a new Folder and assigns it to a Category.

Syntax

Visual Basic (Declaration)

```
Public Overrides NotOverridable Function AddFolderWithCategory( _
    ByVal name As System.String, _
    ByVal parentId As System.Long, _
    ByVal isLibrary As System.Boolean, _
    ByVal categoryId As System.Long _
) As Folder
```

C#

```
public override Folder AddFolderWithCategory(
    System.string name,
    System.long parentId,
    System.bool isLibrary,
    System.long categoryId
)
```

First, we need a name for the new folder. Note – For simplification, we use the direct coding of the new folder's name. In reality, you will use this as a parameter starting the script. The implementation (transferring the new name and a title property value) for this looks like:

```
Param([String]$FolderName, [String]$Title)
```

The command expects a parent folder where to start. It could be the root '\$/' or any subfolder, like '\$/Designs/':

```
# we need to know the parent folder (or the Vault $root) to create a new one
$myRootFolder = $vault.DocumentService.GetFolderByPath('$/Designs/')
```

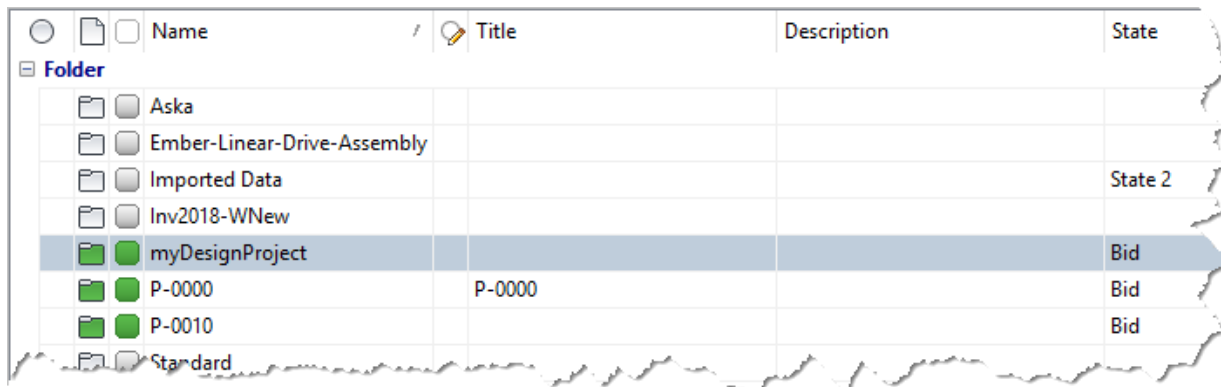
To set the category, we need to get the project category's definition ID:

```
# to create a folder with given category, the category ID of 'Project' is required
$FolderCategories = $vault.CategoryService.GetCategoriesByEntityClassId('FLDR', $true)
$ProjectFldrCatId = ($FolderCategories | Where-Object {$_.Name -eq 'Project' }).Id
```

Now, we can call the method:

```
# with folder Id and category Id we are set to create the project folder
$myProjectFolder = $vault.DocumentServiceExtensions.AddFolderWithCategory('myDesignProject', $myRootFolder.Id, $false, $ProjectFldrCatId)
```

Run the script and review the result in Vault:

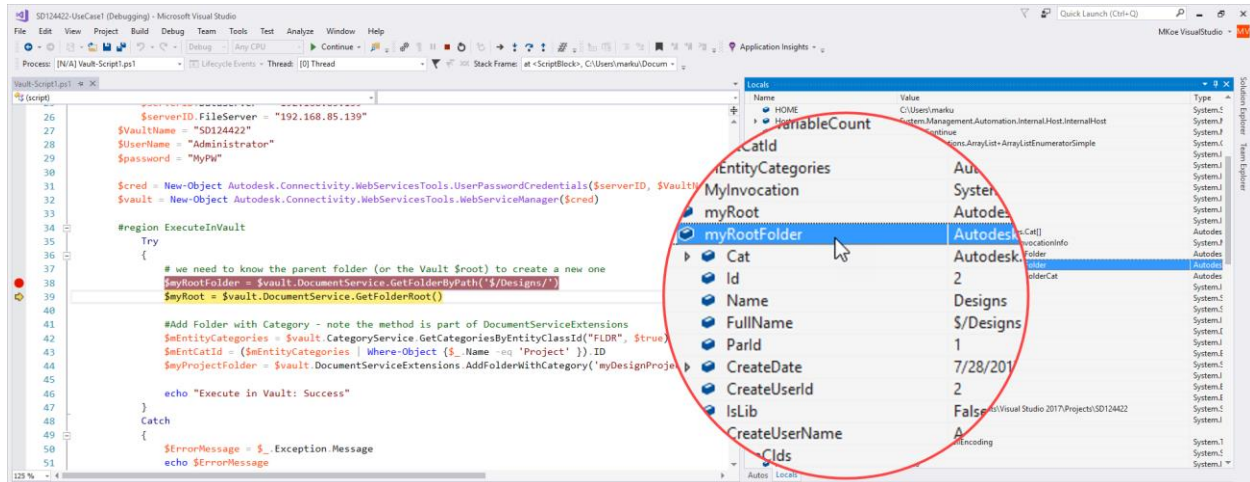


	Name	Title	Description	State
Folder	Aska			
	Ember-Linear-Drive-Assembly			
	Imported Data			State 2
	Inv2018-WNew			
	myDesignProject			Bid
	P-0000	P-0000		Bid
	P-0010			Bid
	Standard			

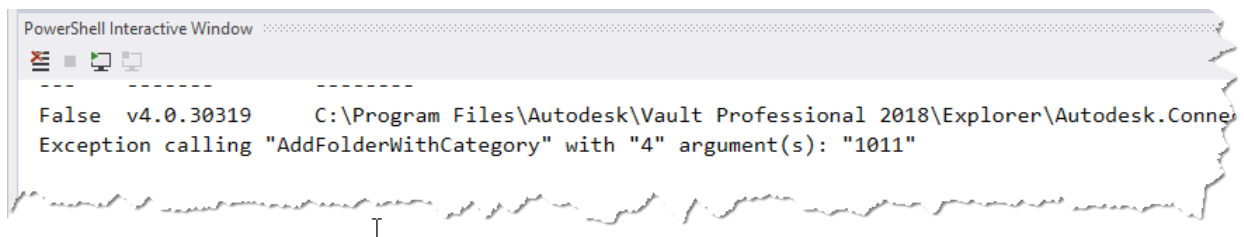
As we coded the new name as a static value in the script, any repetitive run will fail as the folder already exists. We proceed using this failure to review the debug options and error handling in the next steps.

2.3.6 Solution Step 5 – Debugging, Error Handling and Return Values

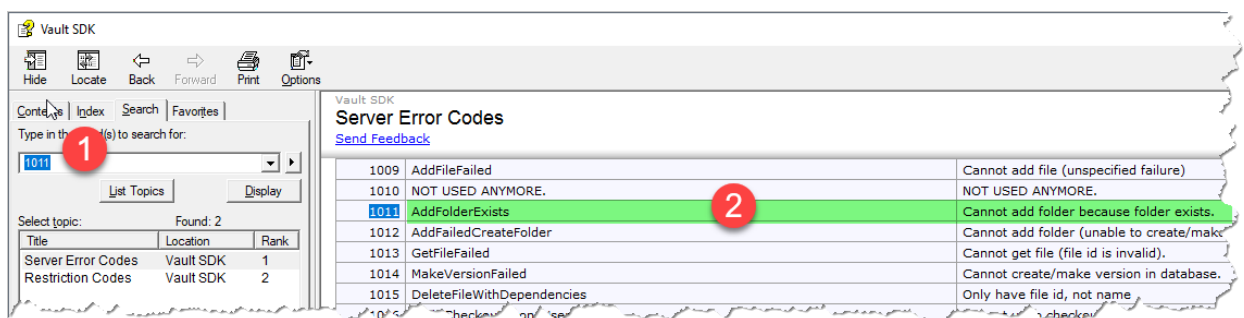
Don't miss to set a breakpoint as Visual Studio PowerShell doesn't allow stepping into from the beginning. Open the Local (Variables) Window to review each's step return values:



The next run (with the new project folder already created) will exit with the error "1011".



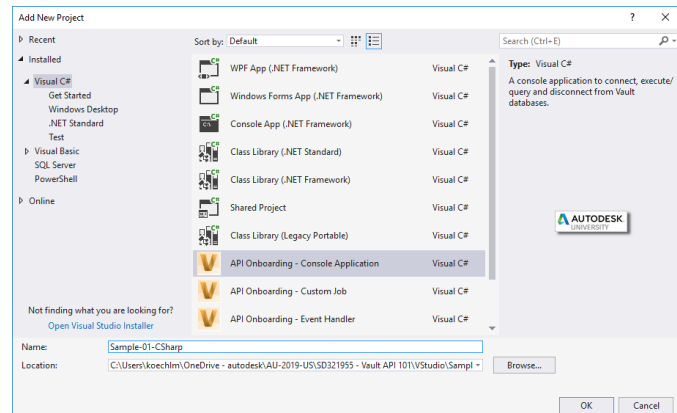
Search in Vault SDK Help for the error value and confirm that 1011 is the code reflecting our failure:



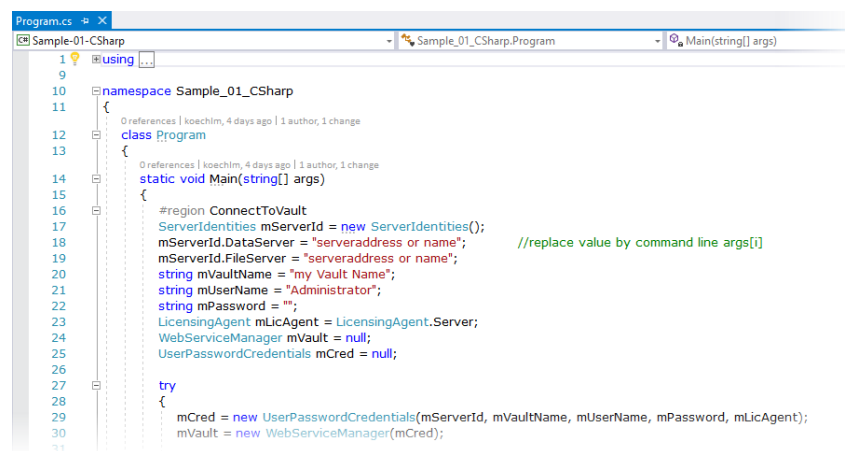
2.4 C# Solution

2.4.1 Solution Step 1 – Create New Script

Start a new C# project selecting the script template installed before².



Open the class “Program.cs”:



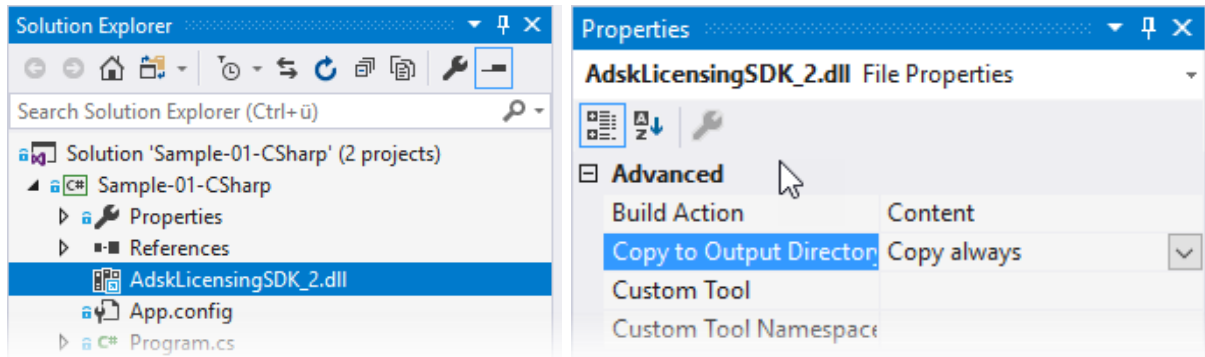
Reviewing the existing regions in the script Vault-Script.ps1 file, you quickly will recognize areas for implementation (#region) but also code to establish the connection.

2.4.2 Licensing

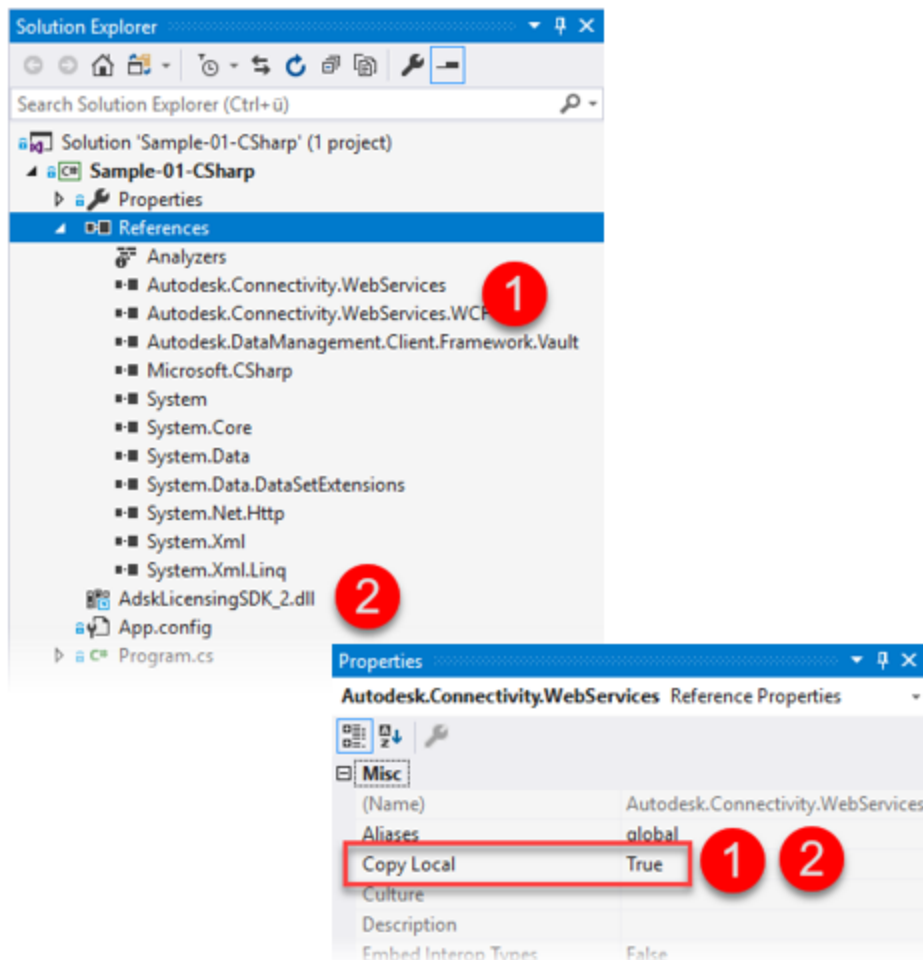
Compiling the project “Sample-01-CSharp” results in a standalone executable. To enable Vault access, we need to distribute the Vault SDK licensing library alongside the primary executable. The SDK template API-Onboarding Console Application references the required library

² To install SDK templates review instruction in chapter 1.3.3

“AdskLicensingSDK_2.dll” and also enabled copying to the output directory.



The same applies to all other SDK references:



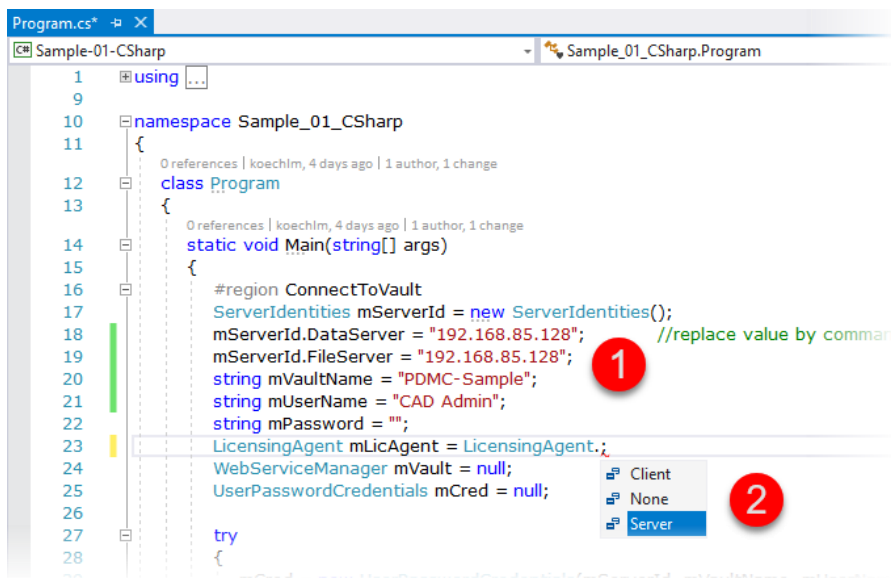
Autodesk Licensing allows computer-based activation (single user) or network licensing. Any application needs to include the licensing library to communicate either with the local (computer) license service or the network license server.

The library needs to copy with your application files.

There is detailed documentation about the core references required in the Knowledgebase of SDK Help: chapter WCF and the SDK.

2.4.3 Solution Step 3 – Connect to Vault

Specify the connection parameters (1) and the license type (2):



```

1  using ...
9
10 namespace Sample_01_CSharp
11 {
12     0 references | koechlm, 4 days ago | 1 author, 1 change
13     class Program
14     {
15         0 references | koechlm, 4 days ago | 1 author, 1 change
16         static void Main(string[] args)
17         {
18             #region ConnectToVault
19             ServerIdentities mServerId = new ServerIdentities();
20             mServerId.DataServer = "192.168.85.128"; //replace value by command
21             mServerId.FileServer = "192.168.85.128";
22             string mVaultName = "PDMC-Sample";
23             string mUserName = "CAD Admin";
24             string mPassword = "";
25             LicensingAgent mLicAgent = LicensingAgent.;
26             WebServiceManager mVault = null;
27             UserPasswordCredentials mCred = null;
28
29             try
30             {
31                 mCred = new UserPasswordCredentials(mVaultName, mUserName, mPassword);
32             }
33         }
34     }
35 }
  
```

Note – Selecting the license type “Client” requires a Vault Client installed on your machine and activated.

If you are concerned about sharing user/password information in scripts, you find an alternate solution approach reading chapter 2.4; it is applicable in PowerShell as well.

Test your connectivity; build and run or step into activating a breakpoint.



2.4.4 Solution Step 4 – Execute the task

Starting to write the script, you might ask how to find the methods/method names required to perform our desired actions.

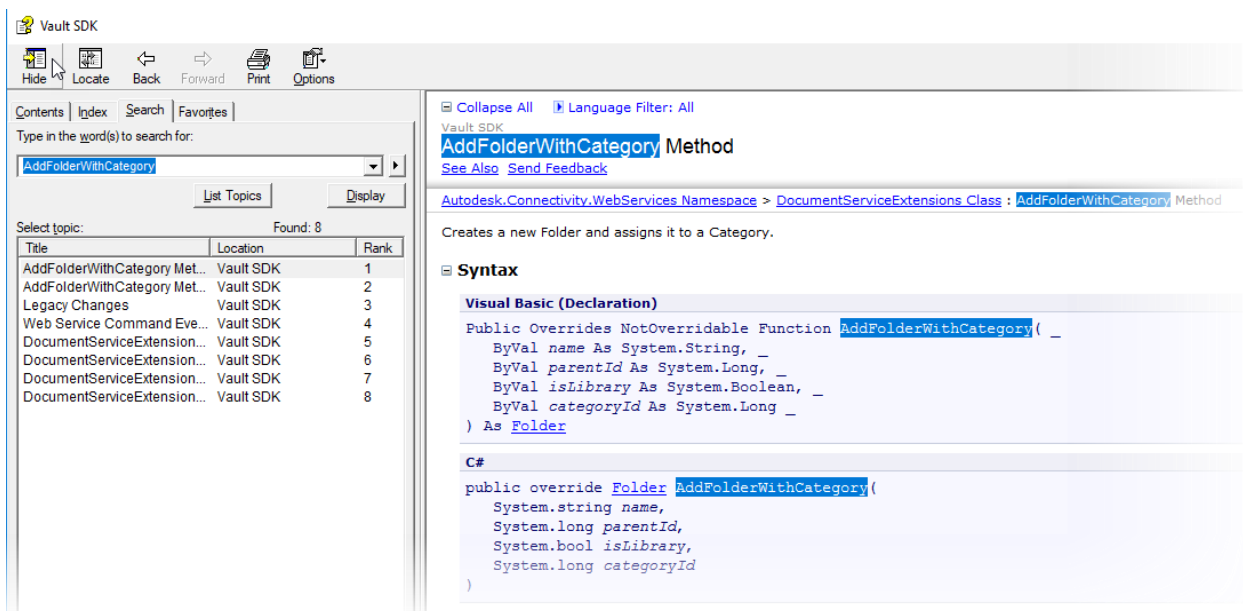
Note – You can use Telerik's Fiddler extended by coolOrange's vapiTrace to record all Vault API calls of manually performed tasks in Vault. For installation and further information, visit the web pages of Telerik, <http://www.telerik.com/fiddler>, and coolOrange, https://www.coolorange.com/en/apps_vault.php. A video tutorial in Vault Knowledge Network instructs how to configure and use this powerful tool with the sample adding a folder with the project category in Vault. The video is available here: <http://autode.sk/2ltSYys>

To add a new project folder, we have two options:

- Add a new folder, then update the category
- Add a new folder with a given category

The class solution project Sample-01-CSharp.sln follows the second approach.

We use the method `AddFolderWithCategory`; compare the Help documentation to get detailed information about which service it belongs to, and about the input- and return values.



The screenshot shows the Vault SDK help interface. On the left, a search bar contains 'AddFolderWithCategory' and a table lists search results. On the right, the documentation for the 'AddFolderWithCategory' method is displayed, including its syntax in Visual Basic and C#.

Title	Location	Rank
AddFolderWithCategory Met...	Vault SDK	1
AddFolderWithCategory Met...	Vault SDK	2
Legacy Changes	Vault SDK	3
Web Service Command Eve...	Vault SDK	4
DocumentServiceExtension...	Vault SDK	5
DocumentServiceExtension...	Vault SDK	6
DocumentServiceExtension...	Vault SDK	7
DocumentServiceExtension...	Vault SDK	8

AddFolderWithCategory Method
 See Also Send Feedback

Autodesk.Connectivity.WebServices.Namespace > DocumentServiceExtensions Class : AddFolderWithCategory Method

Creates a new Folder and assigns it to a Category.

Syntax

Visual Basic (Declaration)

```
Public Overrides NotOverridable Function AddFolderWithCategory( _
    ByVal name As System.String, _
    ByVal parentId As System.Long, _
    ByVal isLibrary As System.Boolean, _
    ByVal categoryId As System.Long _
) As Folder
```

C#

```
public override Folder AddFolderWithCategory(
    System.string name,
    System.long parentId,
    System.bool isLibrary,
    System.long categoryId
)
```


First, we need a name for the new folder. Note – For simplification, we use the direct coding of the new folder's name.

```
//the new project's name
string mFldrName = "myDesignProject"; //replace value by command line args[i]
```

In reality, you will use this as a parameter starting the script.

The add folder command expects a parent folder where to start. It could be the root '\$/' or any subfolder, like '\$/Designs/':

```
//we need to know the parent folder (or the Vault $root) to create a new one
Folder mRootFolder = mVault.DocumentService.GetFolderByPath("$$/Designs");
```

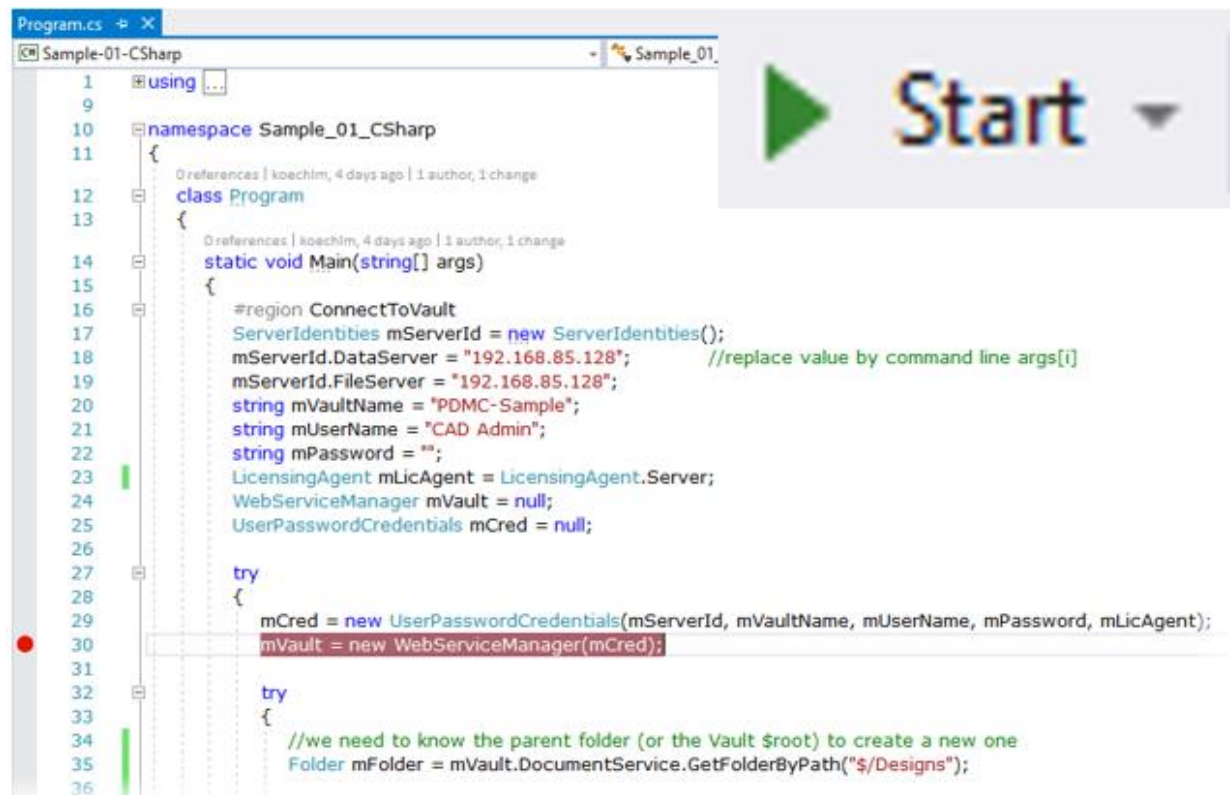
To set the category, we need to get the project category's definition ID:

```
//to create a folder with category; the category ID of 'Project' is required
Cat[] mFolderCategories = mVault.CategoryService.GetCategoriesByEntityClassId("FLDR", true);
long mProjectFldrCatId = mFolderCategories.Where(n => n.Name == "Project").FirstOrDefault().Id;
```

Now, we can call the method:

```
//with parent folder Id and category Id we are set create folder with category in one call
Folder mProjectFolder = mVault.DocumentServiceExtensions.AddFolderWithCategory(mFldrName, mRootFolder.Id, false, mProjectFldrCatId);
```

Build the project script, step into,



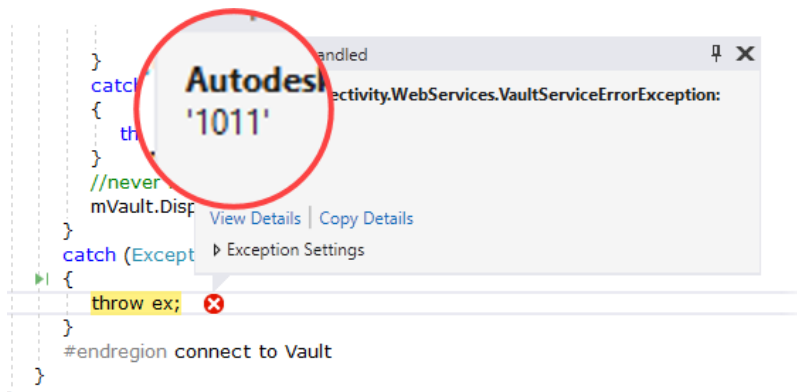
and review the result in Vault:

	Name	Title	Description	State
Folder				
	Aska			
	Ember-Linear-Drive-Assembly			
	Imported Data			State 2
	Inv2018-WNew			
	myDesignProject			Bid
	P-0000	P-0000		Bid
	P-0010			Bid
	Standard			

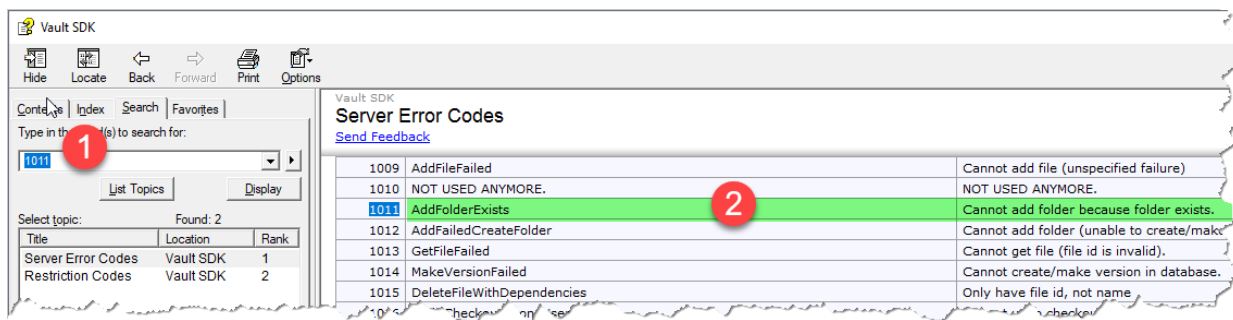
As we coded the new name as a static value in the script, any repetitive run will fail as the folder already exists. We proceed using this failure to review the debug options and error handling in the next steps.

2.4.5 Solution Step 5 – Debugging, Error Handling and Return Values

The next run (with the new project folder already created) will exit with the error “1011”.



Search in Vault SDK Help for the error value and confirm that 1011 is the code reflecting our failure:



2.4.6 Alternate Solution – Avoid coding/scripting of user credentials

Instead of adding user credentials to the code to a script or command line parameters, we can leverage the Vault Development Framework API. It offers a predefined user dialog to log-on and options to auto-login in with given encrypted credentials. Compare the project “Sample-01-CSharp-VDF” on how to do this. The pre-requisite is adding the framework libraries:

```
using Autodesk.Connectivity.WebServices;
using Autodesk.Connectivity.WebServicesTools;
using VDF = Autodesk.DataManagement.Client.Framework;
using VdfForms = Autodesk.DataManagement.Client.Framework.Vault.Forms;
```

The code looks like this:



```
#region ConnectToVault
ServerIdentities mServerId = new ServerIdentities();
mServerId.DataServer = "192.168.1.128";
mServerId.FileServer = "192.168.1.128";
string mVaultName = "PDMConnect";
string mUserName = "CAD";
string mPassword = "12345678";
LicensingAgent mLicAgent = LicensingAgent.LicensingAgent;
WebServiceManager mVault = null;
UserPasswordCredentials mCred = null;

try
{
    mCred = new UserPasswordCredentials(mUserName, mPassword);
    mVault = new WebServiceManager(mServerId, mVaultName, mCred);
}
catch { }
}

#region ConnectToVault
VdfForms.Settings.LoginSettings mloginSettings = new VdfForms.Settings.LoginSettings();
//first time log-in will ask for user credentials, set to Autologin = Enabled to avoid future credential requests
mloginSettings.AutoLoginMode = VdfForms.Settings.LoginSettings.AutoLoginModeValues.RestoreAndExecute;
VDF.Vault.Currency.Connections.Connection mConn = null;
WebServiceManager mVault = null;
try
{
    mConn = VdfForms.Library.Login(mloginSettings);
    if (mConn == null)
    {
        Debug.Print("Login failed or canceled");
        return;
    }
    else
    {
        mVault = mConn.WebServiceManager;
    }
}
catch { }
}
```

2.5 Use Case 1 | Summary

PowerShell scripting is straightforward accessing Vault. It requires no compiler or specific editor.

However, consuming the console template for C# having all libraries and references set, also is quickly ready to run.

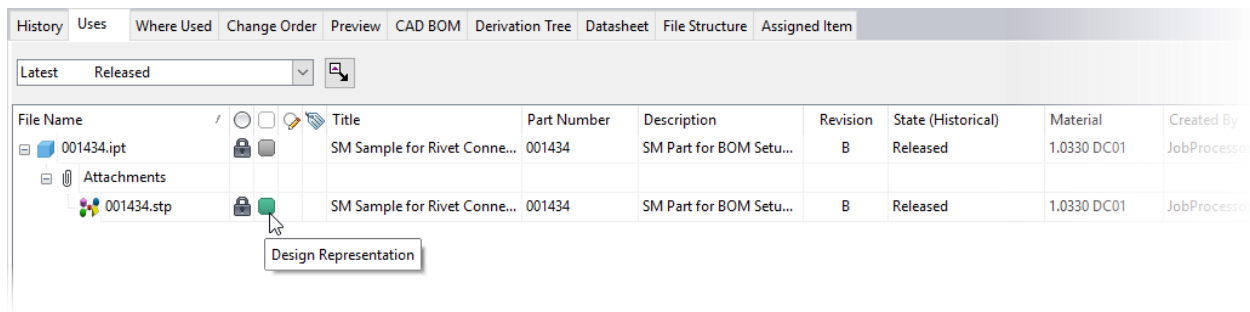
Besides the preferred programming language, you always should consider limiting your server calls to the minimum. We listed two options to create a folder with a category. Both options require as many requests as many projects you need to add. The first approach combines one more: Update the added folder(s)' categories. In this case, only a single additional method call solves it, because the function updates multiple folders.

3 Hands-On – Use Case 2 | Job Handler Extension

3.1 Use Case 2 | Description

Frequently other business systems expect information, files, or links from Vault. If the information is related to CAD files, usually neutral format files like DWF, PDF, STEP, DXF, or others copy to the enterprise system or repository. Vault comes with default jobs for DWF and PDF, and this sample enables you creating custom jobs translating to any other format supported by Inventor.

The solution demonstrates how to create a STEP export for part or assembly components and how to add the result as a Design Representation Attachment:

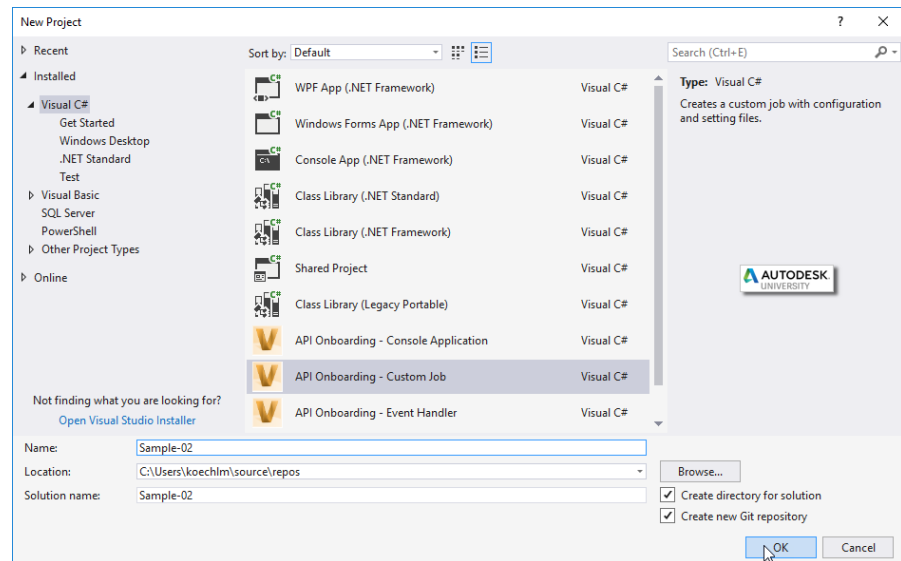


3.2 Use Case 2 | Solution Steps

Based on a class template, less complicated than any SDK sample, we are going to establish a custom job handler, validate its registration and access for debugging. Once you are familiar with these preconditions, coding can start.

3.2.1 Solution Step 1 – Create a New Custom Job

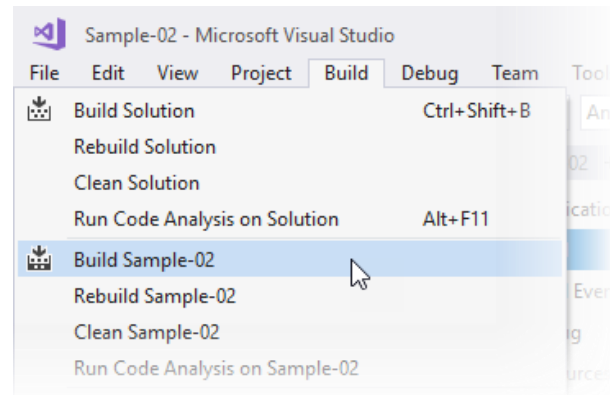
Start a new solution/project selecting the API-Onboarding Custom Job template³.



3.2.2 Validate Job Registration

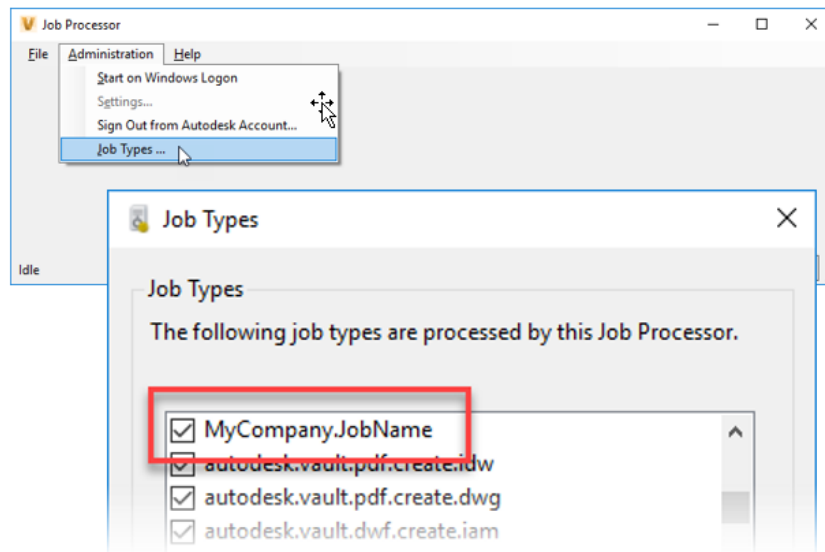
Build the project; the template has everything set:

- The output folder
- The job name to register in Vault Job Processor



³ To install class templates review instruction in chapter 1.3.3

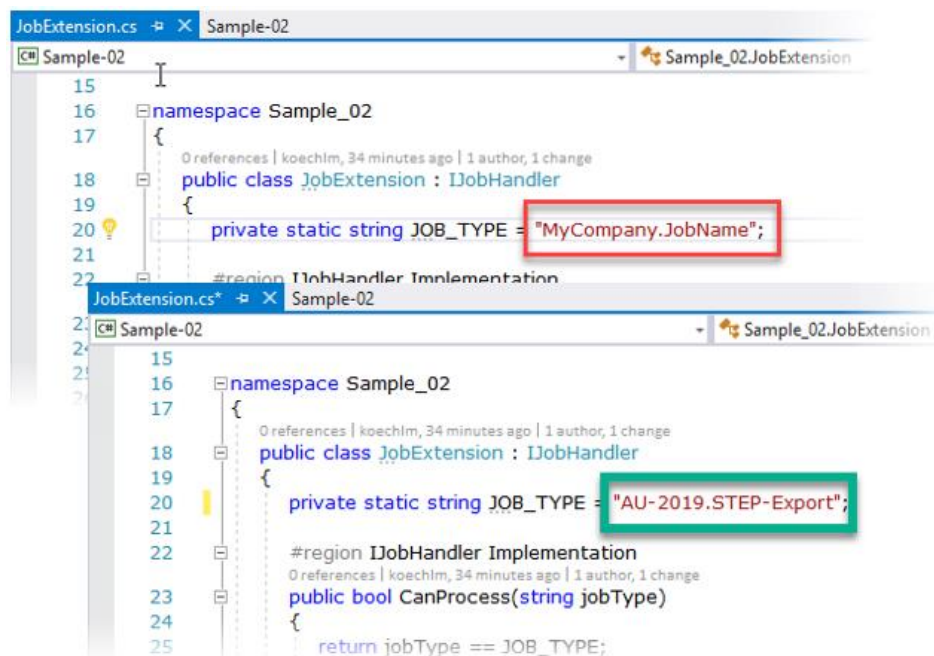
Start the Job Processor and review the new Job in the Job Types window.



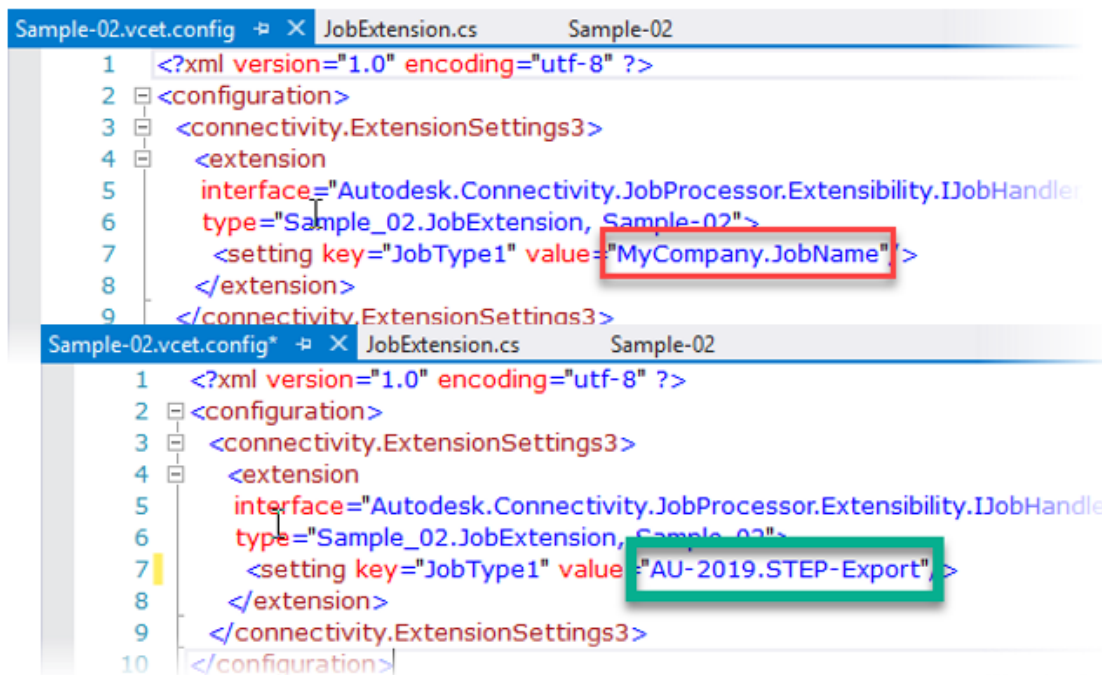
We verified that the new job is ready to register and run on our job processor. Before we proceed, we should change the name of the new job. As a best practice a job name follows these recommendations:

- Select a name that explains the job's primary purpose, in our sample it is STEP-Export
- Add your company name; this will allow differentiating multiple jobs of similar names.

Continue changing the name in the Extension class code:



Repeat the rename action also in the extension's config file:

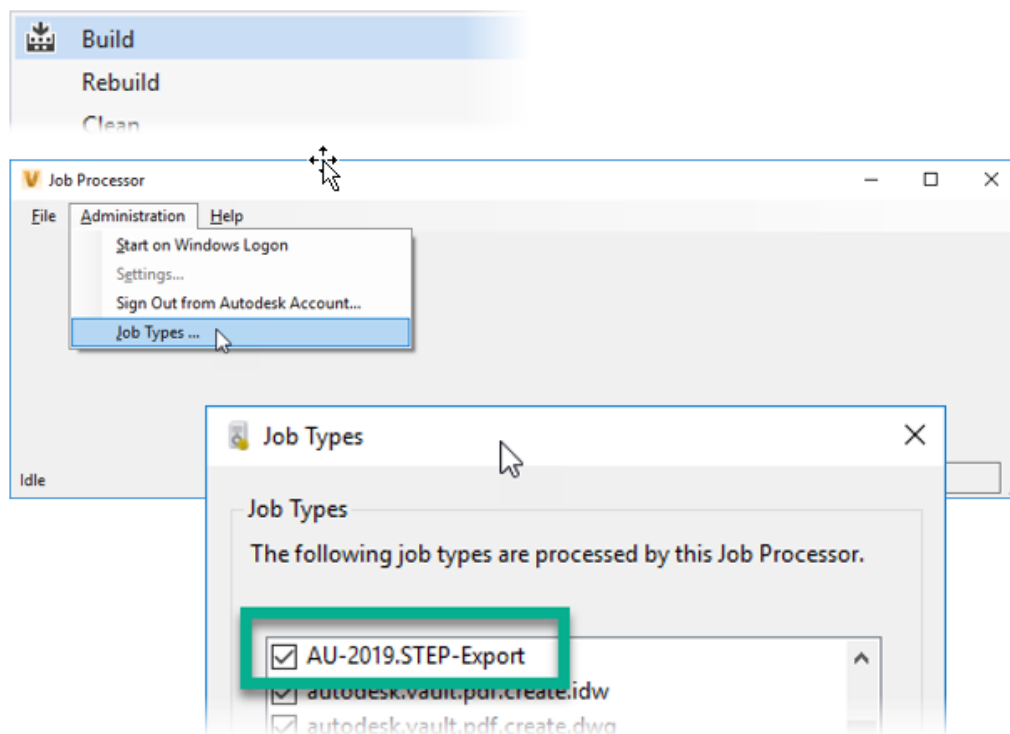


```

Sample-02.vcet.config  JobExtension.cs  Sample-02
1  <?xml version="1.0" encoding="utf-8" ?>
2  <configuration>
3  <connectivity.ExtensionSettings3>
4  <extension
5  interface="Autodesk.Connectivity.JobProcessor.Extensibility.IJobHandler
6  type="Sample_02.JobExtension, Sample-02">
7  <setting key="JobType1" value="MyCompany.JobName"/>
8  </extension>
9  </connectivity.ExtensionSettings3>

Sample-02.vcet.config*  JobExtension.cs  Sample-02
1  <?xml version="1.0" encoding="utf-8" ?>
2  <configuration>
3  <connectivity.ExtensionSettings3>
4  <extension
5  interface="Autodesk.Connectivity.JobProcessor.Extensibility.IJobHandler
6  type="Sample_02.JobExtension, Sample-02">
7  <setting key="JobType1" value="AU-2019.STEP-Export"/>
8  </extension>
9  </connectivity.ExtensionSettings3>
10 </configuration>
  
```

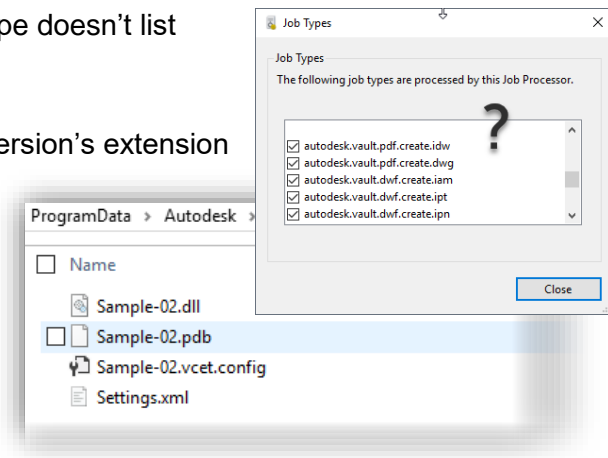
Re-build the project to activate the new Job name's registration. Start the job processor again and validate the new name:



3.2.3 Job Registration – Trouble Shooting

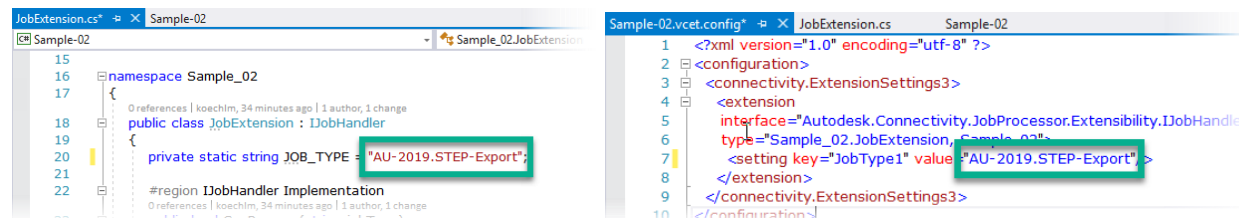
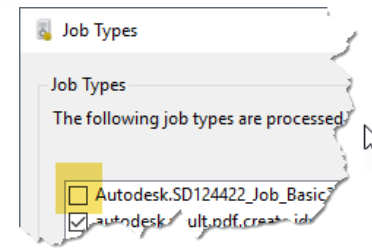
What should you cross-check first, if your Job Type doesn't list your new job?

- Is the output folder matching your Vault version's extension folder? (Program Data\Autodesk\Vault <version>\Extensions\<JobName>
- Are the required files listed?



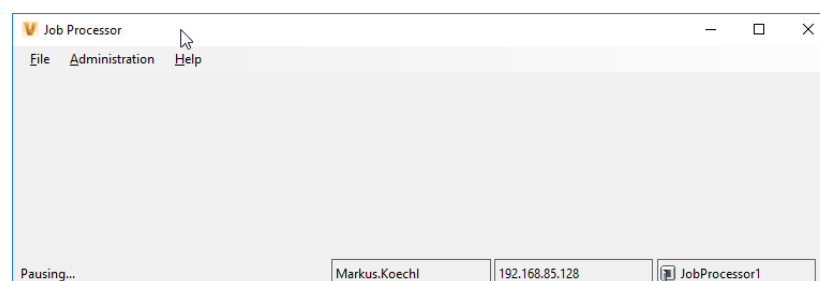
What should you compare and adjust, if your job is listed, but not enabled in the Job Types dialog?

- Ensure that names match in dll and .vcet.config



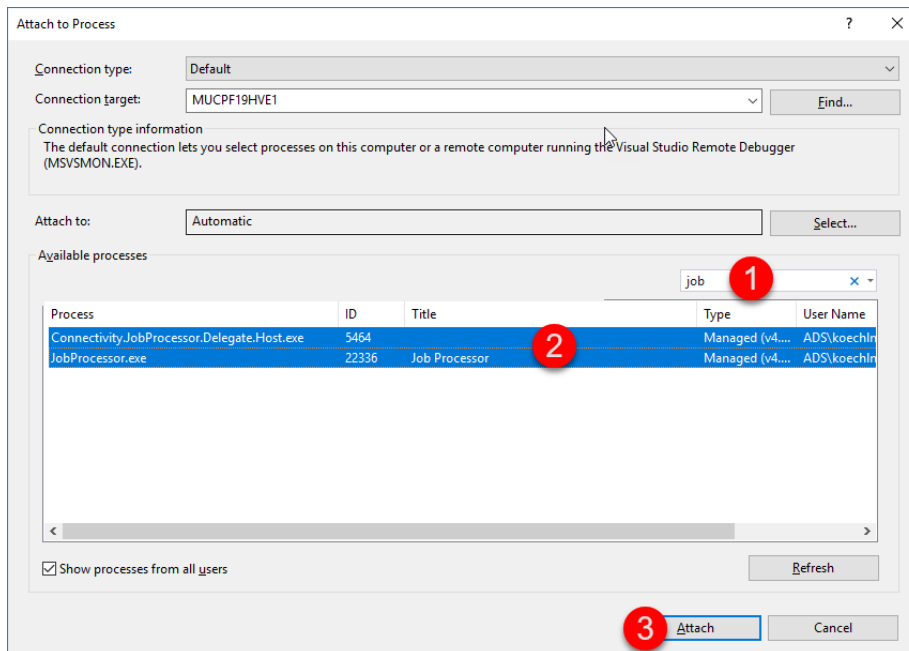
3.2.4 Solution Step 2 – Establish Job Handler Debugging

Let's establish debugging as the last step before coding the job's core functionality. Make sure that your Job Processor is up and running:



To capture all the events we are going to attach two processes:

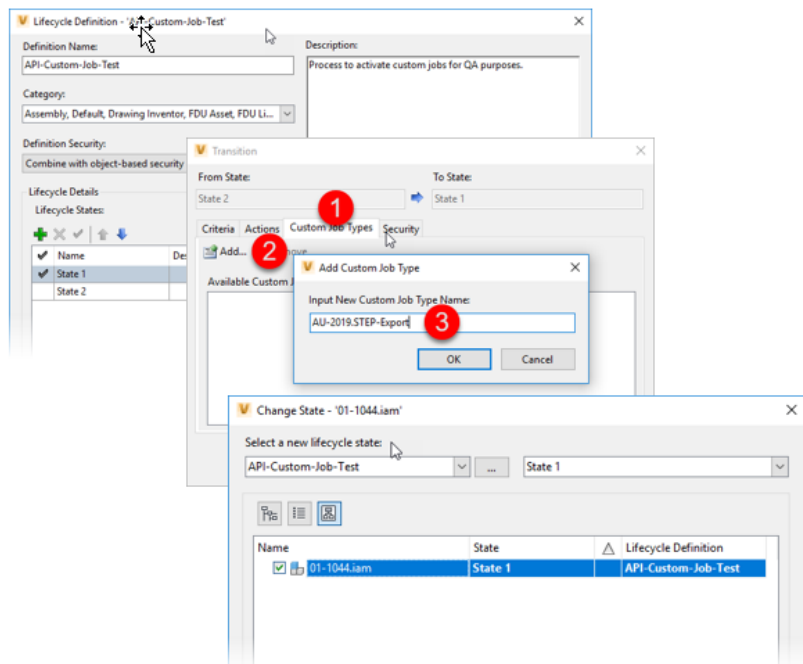
1. The job processing delegate host: Connectivity.JobProcessor.Delegate.Host.exe
2. The job processor application itself



Our new job is ready for execution, the debugger waiting to step into a job's execution. But how can we submit this job to the queue?

The usual way of doing this is, adding the job name to a lifecycle transition.

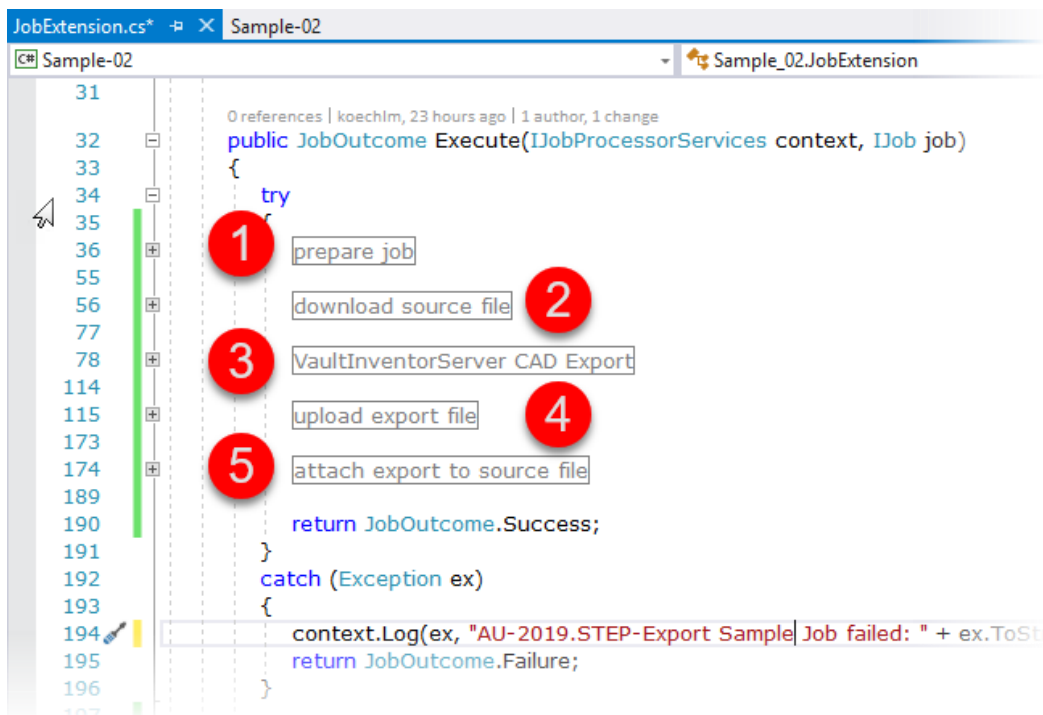
To submit a job, I prefer using a “special” lifecycle called “API-Custom-Job-Test”; it defines two states, and each transition enables the job submission. So lifecycle changes on a file attached to it behave like a toggle.



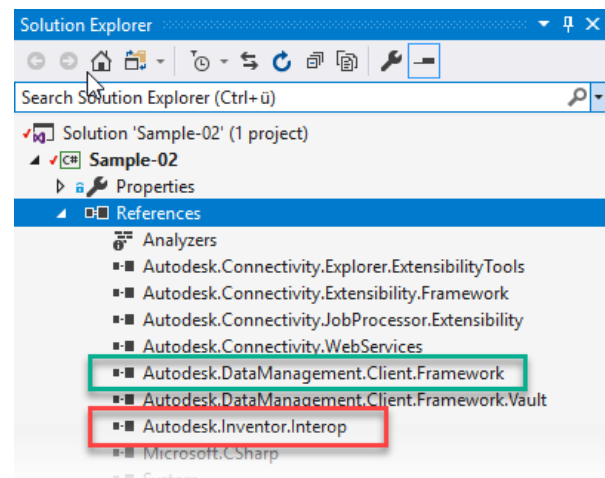
Once you submitted the job to the queue, the template's built-in execution brings up a message box "Hello World".

3.2.5 Solution Step 3 – Coding the Job's Task

During the introduction, we already mentioned that this class is not about learning each API method and its parameter details but more the concepts and systematic approach to implement automation and extensions. So again, open the class' final solution Sample-02.sln in parallel for review. We separated the execution into five regions:



For these 5 sub-tasks, additional APIs are needed – the Vault Development Framework VDF and Inventor API. Add these to the project references:



```
JobExtension.cs Sample-02
C# Sample-02 Sample_02.JobExtension
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using Autodesk.Connectivity.Extensibility.Framework;
5 using Autodesk.Connectivity.JobProcessor.Extensibility;
6 using ACW = Autodesk.Connectivity.WebServices;
7 using VDF = Autodesk.DataManagement.Client.Framework;
8 using Inventor;
```

3.2.5.1 Pickup the job's context, connect to Vault Inventor Server and set conditions to proceed

We pick-up the job context to get the WebServiceManager object; several services and methods will need it later:

```
//pick up this job's context
Connection connection = context.Connection;
Autodesk.Connectivity.WebServicesTools.WebServiceManager mWsMgr = connection.WebServiceManager;
long mEntId = Convert.ToInt64(job.Params["EntityId"]);
ACW.File mFile = mWsMgr.DocumentService.GetFileById(mEntId);
```

We also take the file iteration object (it is a particular version of the file). Note – it might happen that before the job got the file from the queue another user increased the file version already. The job will refuse execution (built-in restriction) if the file is not the latest version. To avoid this, additional steps to get the latest available file version are applicable.

The job's context also shares the configured Inventor Object; as a default, it is InventorServer – the JobProcessor.exe.config file allows switching to full Inventor also.

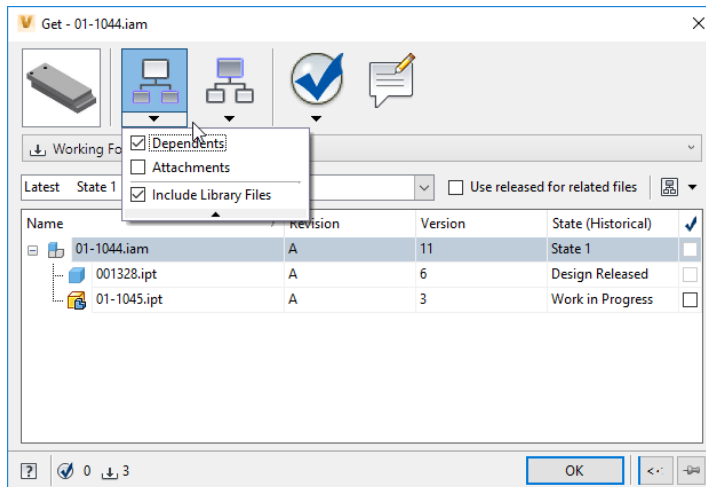
```
//establish InventorServer environment including translator addins; differentiate her in case full Inventor.exe is used
Inventor.InventorServer mInv = context.InventorObject as InventorServer;
ApplicationAddIns mInvSrvAddIns = mInv.ApplicationAddIns;
```

As many file types may add to the queue, we need to filter target file types for CAD format export.

```
// only run the job for ipt and iam file types,
List<string> mFileExtensions = new List<string> { ".ipt", ".iam" };
ACW.File mFile = mWsMgr.DocumentService.GetFileById(mEntId);
if (!mFileExtensions.Any(n => mFile.Name.Contains(n)))
{
    return JobOutcome.Success;
}
```

3.2.5.2 Get the model file from Vault and open it using VaultInventorServer

Downloading a file, you need to set all options that a user would do in the user interface of Vault Explorer. Considering assembly files, we set download options securing a full loaded assembly, including all components.



Note – In the API are more options available handling file resolution. That is important if a job processor uses temporary working folders instead of the default configured one.

Don't weaken a process by having existing files in your working folder. We recommend overwriting existing files for job execution always. You better might also clean-up your working folder regularly.

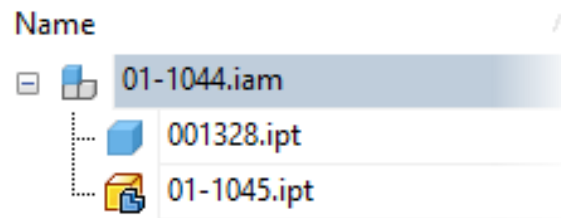
```
#region download source file
//download the source file iteration, enforcing overwrite if local files exist
VDF.Vault.Settings.AcquireFilesSettings mDownloadSettings = new VDF.Vault.Settings.AcquireFilesSettings(connection);
mDownloadSettings.OptionsRelationshipGathering.FileRelationshipSettings.IncludeChildren = true;
mDownloadSettings.OptionsRelationshipGathering.FileRelationshipSettings.IncludeLibraryContents = true;
mDownloadSettings.OptionsRelationshipGathering.FileRelationshipSettings.ReleaseBiased = true;
mDownloadSettings.OrganizeFilesRelativeToCommonVaultRoot = true;
```

With the completion of the download settings, the download executes:

```
//execute download
VDF.Vault.Results.AcquireFilesResults mDownloadResult = connection.FileManager.AcquireFiles(mDownloadSettings);
```

The file results contain all files and do not follow the hierarchical order. Therefore, we need to filter our source file.

Inventor API to open documents expects the full local path. The file extension is another handy detail to replace it later by the Translator format's given extension. We can pick-up the primary file like this:



```
//pickup the primary file from result details
VDF.Vault.Results.FileAcquisitionResult fileAcquisitionResult =
    mDownloadResult.FileResults.Where(n => n.File.EntityName == mFileIteration.EntityName).FirstOrDefault();
string mDocPath = fileAcquisitionResult.LocalPath.FullPath;
string mExt = System.IO.Path.GetExtension(mDocPath);
```

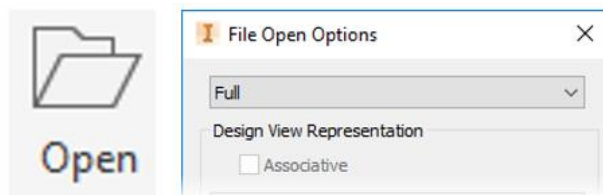
3.2.5.3 Create the export format using VaultInventorServer

VaultInventorServer uses the same API as Inventor. All translator add-ins are available to VaultInventorServer for activation.

```
#region VaultInventorServer CAD Export
//activate STEP Translator environment,
Document mDoc = null;
try
{
    TranslatorAddIn mStepTrans = mInvSrvAddIns.ItemById["{90AF7F40-0C01-11D5-8E83-0010B541CD80}"] as TranslatorAddIn;
    TranslationContext mTransContext = mInv.TransientObjects.CreateTranslationContext();
    NameValueMap mTransOptions = mInv.TransientObjects.CreateNameValueMap();
    if (mStepTrans.HasSaveCopyAsOptions[mDoc, mTransContext, mTransOptions] == true)
    {
        mTransOptions.Value["ApplicationProtocolType"] = 3; //AP 2014, Automotive Design
        mTransOptions.Value["Description"] = "Sample-Job Step Translator using VaultInventorServer";
        mTransContext.Type = IOMechanismEnum.kFileBrowseIOMechanism;
    }
}
```

Lookup the translator add-in's id in the bin\addins*.addin file or debug print listing all. Each translator's option is listed in the Inventor API Help documentation. Access it from Inventor -> Help -> Programming Help.

Now, open the source file in Full Mode by applying the OpenOptions:



```
//open (enforce full load on open, as translators don't work in express mode)
NameValueMap mOpenOptions = mInv.TransientObjects.CreateNameValueMap();
mOpenOptions.Add("ExpressModeBehavior", "OpenFull");
mDoc = mInv.Documents.OpenWithOptions(mDocPath, mOpenOptions, false);
```

The translator saves the new file:



```
//translate writing the export file
DataMedium mData = mInv.TransientObjects.CreateDataMedium();
mData.FileName = mDocPath.Replace(mExt, ".stp");
mStepTrans.SaveCopyAs(mDoc, mTransContext, mTransOptions, mData);
mDoc.Close(true);
```

3.2.5.4 Pick-up the result (export file) and upload to vault

Adding the exported file to Vault requires a pre-check: “Does the file already exist?”

If no, we continue adding the file.

```
ACW.File wsFile = mWsMgr.DocumentService.FindLatestFilesByPaths(vaultFilePath.ToArray()).First();
if (wsFile == null || wsFile.Id < 0)
{
    // upload file to Vault
    if (mFolder == null || mFolder.Id == -1)
        throw new Exception("Vault folder " + mFolder.FullName + " not found");

    var folderEntity = new Autodesk.DataManagement.Client.Framework.Vault.Currency.Entities.Folder(connection, mFolder);
    try
    {
        addedFile = connection.FileManager.AddFile(folderEntity, "Created by Job Processor", null, null, ACW.FileClassification.DesignRepresentation,
        mExpFile = addedFile;
    }
    catch (Exception ex)
```

If it exists, then we are supposed to update the existing one.

To do this – check-out the existing and check-in the new.

Note – ensure not to overwrite the local (new) export file while checking out.

```
// checkin new file version
VDF.Vault.Settings.AcquireFilesSettings aqSettings = new VDF.Vault.Settings.AcquireFilesSettings(connection)
{
    DefaultAcquisitionOption = VDF.Vault.Settings.AcquireFilesSettings.AcquisitionOption.Checkout
};
mResOpt.OverwriteOption = VDF.Vault.Settings.AcquireFilesSettings.AcquireFileResolutionOptions.OverwriteOptions.NoOverwrite;
vdfFile = new VDF.Vault.Currency.Entities.FileIteration(connection, wsFile);
aqSettings.AddEntityToAcquire(vdfFile);
var results = connection.FileManager.AcquireFiles(aqSettings);
try
{
    mUploadedFile = connection.FileManager.CheckinFile(results.FileResults.First().File, "Created by Job Processor", false, null, null, false, null, ACW.FileClassification.DesignRepresentation,
    mExpFile = mUploadedFile;
}
catch (Exception ex)
{
}
```

3.2.5.5 Attach the uploaded export file to the source file as Design Representation

Attaching the uploaded file is the final step in this introduction.

```
ACW.FileAssocParam mAssocParam = new ACW.FileAssocParam();
mAssocParam.CldFileId = mExpFile.Id;
mAssocParam.ExpectedVaultPath = mWsMgr.DocumentService.FindFoldersByIds(new long[] { mFile.FolderId }).First().FullName;
mAssocParam.RefId = null;
mAssocParam.Source = null;
mAssocParam.Type = ACW.AssociationType.Attachment;
mWsMgr.DocumentService.AddDesignRepresentationFileAttachment(mFile.Id, mAssocParam);
```

For production usage, you should also synchronize status and properties from the source to the export file. To learn these additional two actions, review the fully functional sample available here: <https://github.com/koechlm/Vault-Sample---STEPExportJob>

3.2.6 Solution Step 4 – Complete Error Handling

The template implements a rough structure to feedback errors to the Vault Job Queue. As a minimum, always feedback the job success/failure and the step of failure. As a best practice – don't leave possible shortcomings up to the exception handling. Handle them, stop processing, and feedback specifically to the Job Queue.

```
    catch (Exception ex)
    {
        throw new Exception("Job could attach the export result.", ex);
    }
    #endregion attach

    return JobOutcome.Success;
}
catch (Exception ex)
{
    context.Log("AU-2019.STEP-Export Sample Job failed: " + ex.ToString(), MessageType.eError);
    return JobOutcome.Failure;
}
```

3.3 Use Case 3 | Takeaways

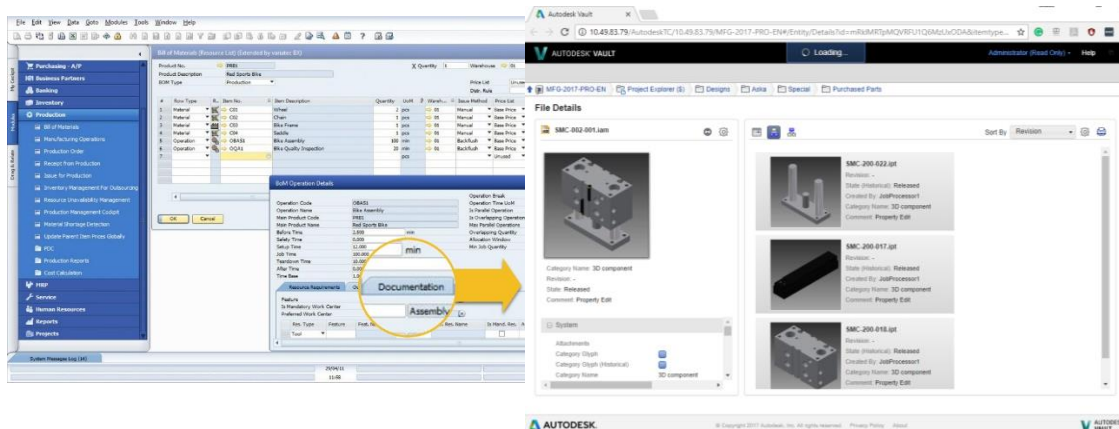
- Always validate your custom job's framework before writing code
 - Job name registration | Job submission | Debugging
- Download settings are essential for successful job execution
 - Plan these using the user interface
- The sample solution relies on enforced Vault settings for the Inventor project and working folder.

- Compare the complete sample mentioned below, how to pro-actively handle Inventor project files.
- The sample solution does not synchronize file status and properties
 - Get a fully functional sample solution from my GitHub repository.

4 Hands-On – Use Case 3 | Event Handler

4.1 Use Case 3 | Description

Frequently other business systems expect information, files, or links from Vault. Instead of copying files, we are going to implement Thin Client links for sharing access to them.



Once a new Vault adds a new project, the event “new folder added” should retrieve a persistent Id. This Id is a permanent reference to an entity within a Thin Client Server URL.

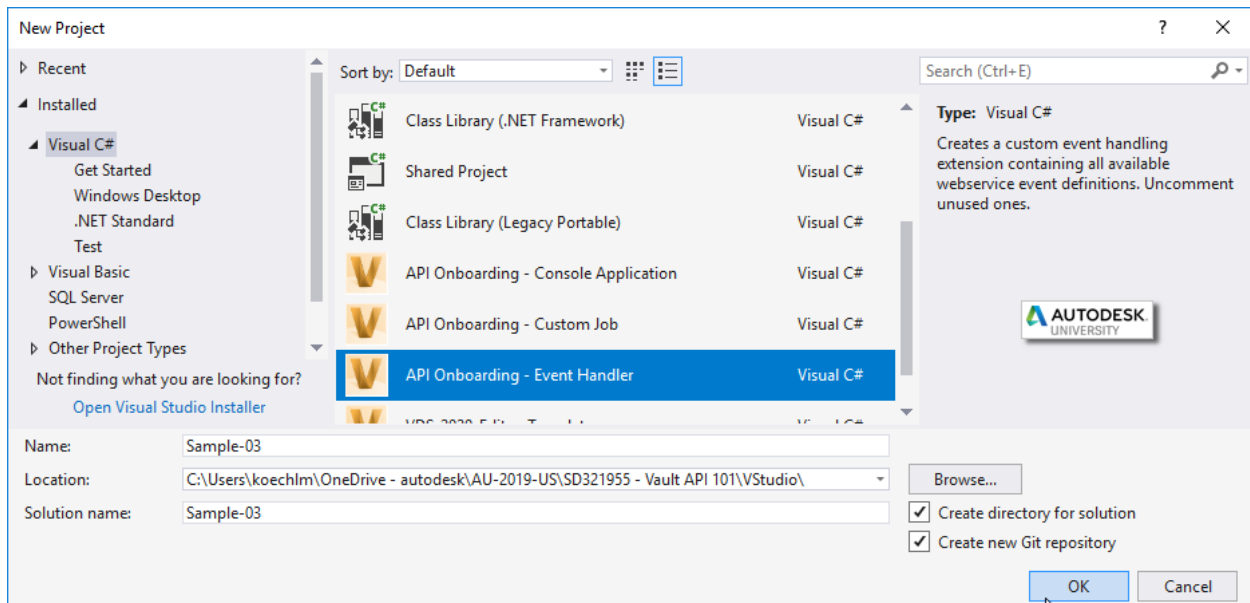
4.2 Use Case 3 | Solution Steps

Create an event handler extension for Vault Clients that subscribes to the Event “AddFolder”. Not any folder should fire share a link, only folders with category “Project”.

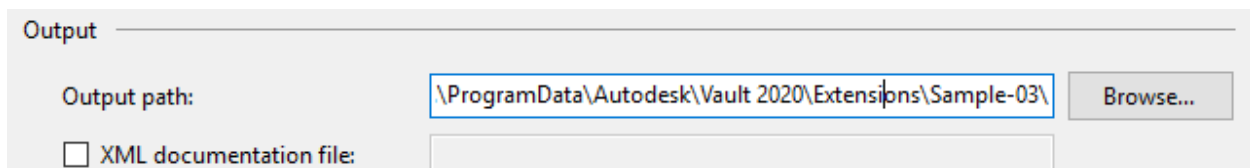
Another Event to subscribe is the MoveFolder; with that we easily can prevent projects from being moved.

4.2.1 Solution Step 1 – Create New Event Handler “Add Folder”

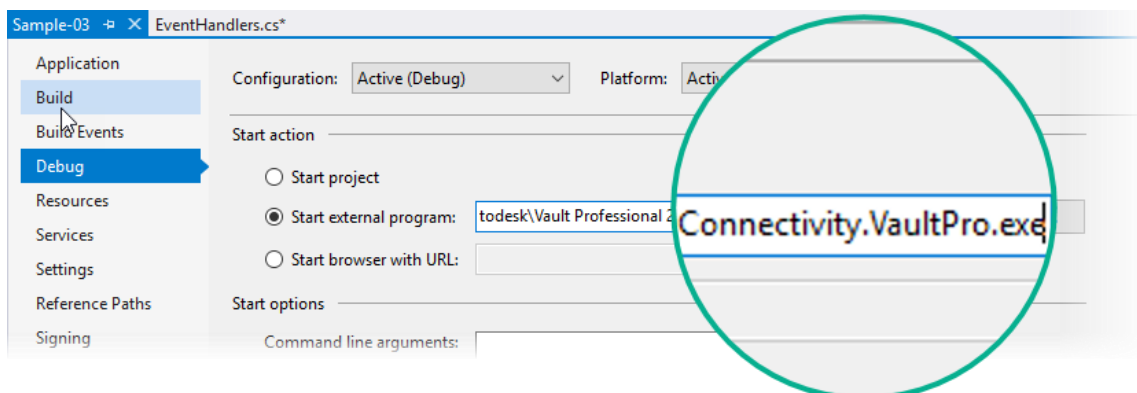
Start a new project/solution selecting the SDK template “API-Onboarding Event Handler”⁴:



The output path should reflect the project’s name as a default; you may change it as well:



Prepare to debug and activate the start external program for C:\Program Files\Autodesk\Vault Professional 2020\Explorer\Connectivity.VaultPro.exe:



⁴ To install class templates review instruction in chapter 1.3.3

4.2.2 Solution Step 2 – Coding

The extension template contains all available web service events; I suggest to uncomment all unused ones.

```
// Folder Events
//DocumentService.AddFolderEvents.GetRestrictions += new EventHandler<AddFolderComm
//DocumentService.AddFolderEvents.Pre += new EventHandler<AddFolderComm
DocumentService.AddFolderEvents.Post += new EventHandler<AddFolderComm
//DocumentService.DeleteFolderEvents.GetRestrictions += new EventHandler<D
//DocumentService.DeleteFolderEvents.Pre += new EventHandler<DeleteFolderC
//DocumentService.DeleteFolderEvents.Post += new EventHandler<DeleteFolderC
DocumentService.MoveFolderEvents.GetRestrictions += new EventHandler<Mov
//DocumentService.MoveFolderEvents.Pre += new EventHandler<MoveFolderCor
//DocumentService.MoveFolderEvents.Post += new EventHandler<MoveFolderC
```

Each web service event offers to split into three notifications; for adding folders we can hook on AddFolderEvents.Post. The get restriction fires before any action starts. It allows adding a restriction if we encountered missing conditions or pre-requisites. Restrictions are Vault objects sharing a GUI to display the restriction text (reason of the restriction).

The pre-event fires before the entity originate. At this stage we could retrieve additional information that might help to decide whether the folder should create or that we restrict the operation.

The post-event allows us to grab the new entity and, e.g., add metadata.

We split the planned action into two sub-tasks:

```
private void AddFolderEvents_Post(object sender, AddFolderCommandEventArgs e)
{
    //several functions require the entity class name of Folder
    string mEntCls = "FLDR";

    //get the new folder's object
    Autodesk.Connectivity.WebServices.Folder mFolder = e.ReturnValue;
    if (mFolder.Cat.CatName != "Project")
    {
        return; //we want TC link for projects only
    }

    create TC Link
    Add property/value (TC Link)
}
```

1
2

4.2.2.1 Create Thin Client Link

The event's sender is the DocumentServiceExtensions. We can cast the sender object to it.

The WebServiceManager is the parent of all services, so we get it from the DocumentServiceExtension.

Persistent IDs are supposed to be static identifiers unchanged during future Vault release migrations. Entity Ids are database keys and might change if a database scheme updates.

Therefore, Persistent IDs are the most reliable reference to pick files, folders, items, change orders, or custom entities.

```
//the sender is DocumentServiceExtension ; cast the object
DocumentServiceExtensions mDocSvcExtensions = (DocumentServiceExtensions)sender;
//get the WebServiceManager object; we need later to use other services than the DocumentService
Autodesk.Connectivity.WebServicesTools.WebServiceManager mWsMgr = mDocSvcExtensions.WebServiceManager;

//use KnowledgeVaultService to retrieve the persistent id; folders don't have history, the option Latest is not relevant here
string[] mPersIDs = mWsMgr.KnowledgeVaultService.GetPersistentIds(mEntCls, new long[] { mFolder.Id }, EntPersistOpt.Latest);
string mPersID = mPersIDs[0];
mPersID = mPersID.TrimEnd(mPersID[mPersID.Length - 1]);
```

The Persistent Id is part of the full URL of entities directing to the Thin Client Server:

<ThinClient Service URL>/AutodeskTC/<ServerName>/<VaultName>/#/Entity/Entities?folder=<PersistentID>&start=0

You can get the scheme hitting the user interface command Send Link; the resulting email gives a sample for each entity type you selected.

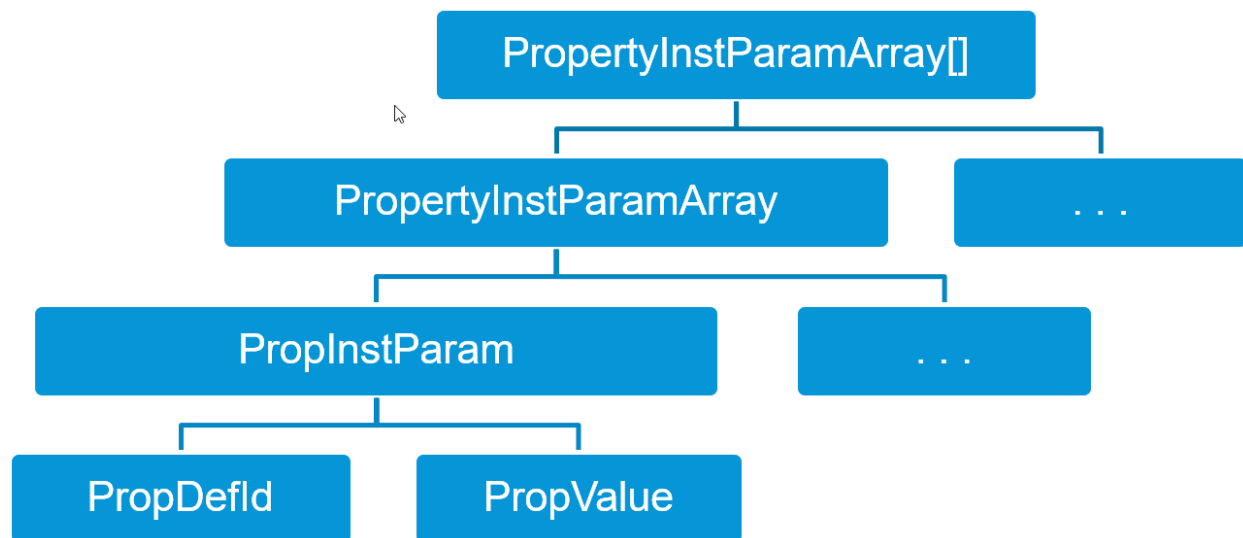
The variables of the URL fill best with values from the DocumentService URL and the active user credentials:

```
//put together the Thin Client Link;
string mTCLink = null;
Uri mServerUri = new Uri(mDocService.Url);
string mVaultName = mWsMgr.WebServiceCredentials.VaultName;
```

4.2.2.2 Add Property and Value to the Project Folder

The method `DocumentServiceExtensions.UpdateFolderProperties()` is pretty powerful; it allows us to update a set of properties for many folders in a single call. Well, in our case, we have one sole property for a single folder, and the method might look over-engineered. Jump on the opportunity to get prepared for future big property transfers! Build the structure from the ground, and it demystifies – I am confident.

First get familiar with the structure of the property instance array arrays:



To fill (or read) these nested arrays always follow bullheaded the same structure:

Note – I prefer handling `List<>` over array objects and convert these as late as possible.

```

List<long> mFldrIds = new List<long>{...};

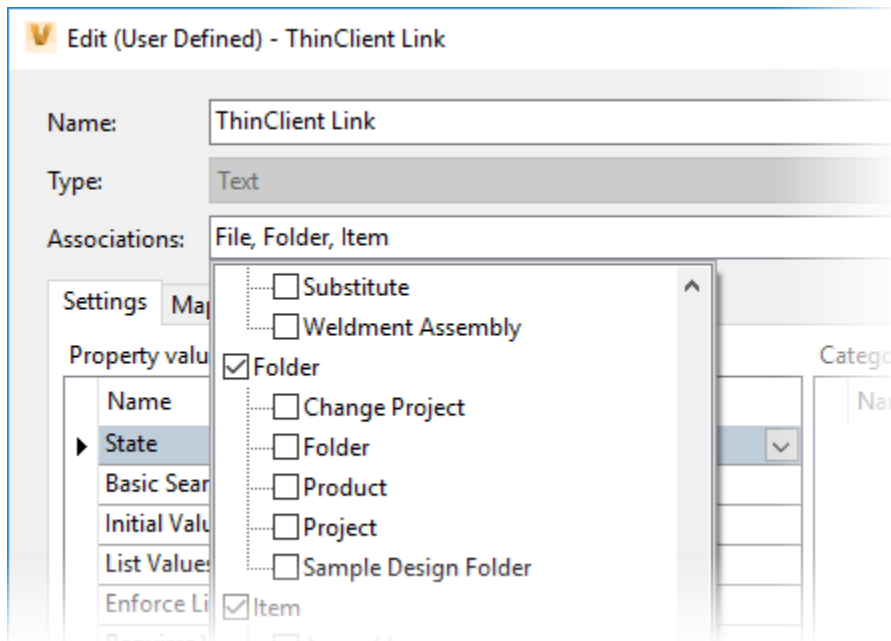
//get the folder property definition Id using the displayname
PropDef[] mPropDefs = mWsMgr.PropertyService.GetPropertyDefinitionsByEntityClassId(mEntCls);
PropDef mPropDef = mPropDefs.SingleOrDefault(n => n.DispName == "ThinClient Link");

PropInstParam mPropInstParam = new PropInstParam();
PropInstParamArray mPropInstParamArray = new PropInstParamArray();
List<PropInstParam> mPropInstParamList = new List<PropInstParam>();
List<PropInstParamArray> mPropInstParamArrayList = new List<PropInstParamArray>();

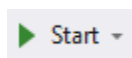
if (mPropDef != null) //create the nested property instance array(s) and run the update
{
    mPropInstParam.PropDefId = mPropDef.Id;
    mPropInstParam.Val = mTCLink;
    mPropInstParamList.Add(mPropInstParam);
    mPropInstParamArray.Items = mPropInstParamList.ToArray();
    mPropInstParamArrayList.Add(mPropInstParamArray);

    mWsMgr.DocumentServiceExtensions.UpdateFolderProperties(mFldrIds.ToArray(), mPropInstParamArrayList.ToArray());
}
  
```

Don't forget to create a User Defined Property in the Vault behavior configuration. Activate it for folder entities:

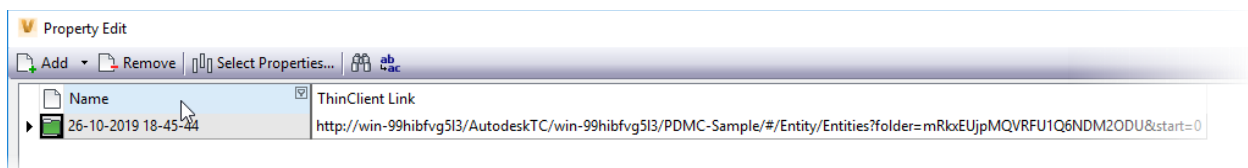


That's it! Time to set a breakpoint and start the debugging session to follow in detail on how it works.



To invoke the event, create a new project folder in Vault Explorer.

Right after you can also run the Sample-01 code; it will not hit the breakpoint, but should also get the new project including a property "ThinClient Link":



4.2.3 Solution Step 3 – Adding Restrictions to Move Folder Events

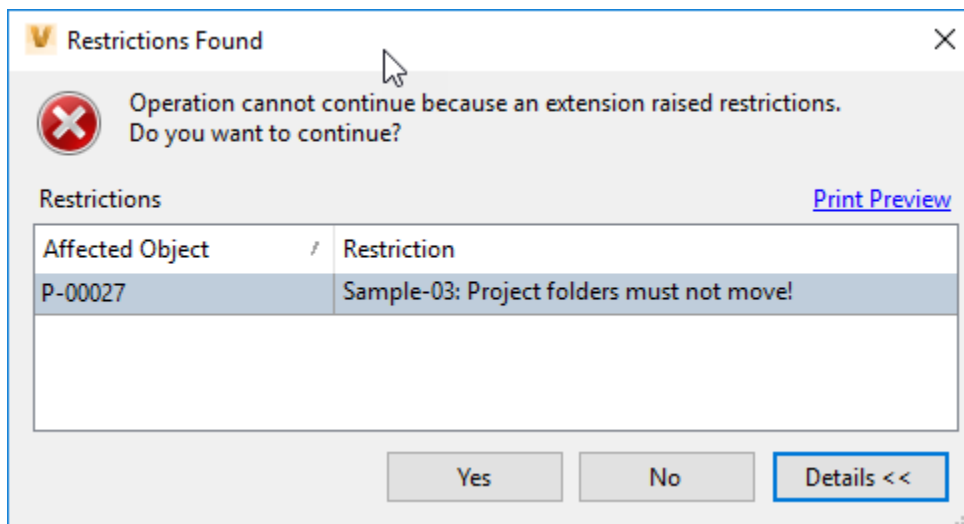
The use case's main target was to share a persistent link to view documents of a project in Thin Client.

Due to the persistent Id, the Thin Client access will persist for ever – even if the entity moved.

In Vault Explorer, aka Thick Client, the move of folders is not appreciated, and administrators often ask to prevent it. Our Event Handler easily can add a restriction to the folder move event.

The Dialog listing the object invoking the restriction is part of the API and allows consistent restriction feedback to the user for Autodesk Vault built-in ones as well as our custom ones.

Implementing restrictions is straightforward, as the event argument offers a method to add a restriction by name and user message.



As a sample, we now implement that folders in Vault no longer move. For that enable the restriction event for folder move first:

```
//DocumentService.DeleteFolderEvents.Pre += new EventHandler<DeleteFolderEventArgs> (e) => {
//DocumentService.DeleteFolderEvents.Post += new EventHandler<DeleteFolderEventArgs> (e) => {
DocumentService.MoveFolderEvents.GetRestrictions += new EventHandler<MoveFolderEventArgs> (e) => {
//DocumentService.MoveFolderEvents.Pre += new EventHandler<MoveFolderEventArgs> (e) => {
//DocumentService.MoveFolderEvents.Post += new EventHandler<MoveFolderEventArgs> (e) => {
```

Adding a restriction is a single line call handing over the source object's name and the text explaining the reason, why we stop the running process at this point:

```
private void MoveFolderEvent_GetRestrictions(object sender, MoveFolderCommandEventArgs e)
{
    //the sender is DocumentService; cast the object
    DocumentService mDocService = (DocumentService)sender;

    //get the new folder's object
    Autodesk.Connectivity.WebServices.Folder mFolder = mDocService.GetFolderById(e.FolderId);

    if (mFolder.Cat.CatName == "Project")
    {
        e.AddRestriction(new ExtensionRestriction(mFolder.Name, "Project folders must not move!"));
    }
}
```

The more complicated task implementing restrictions usually is to define and analyze all pre-requisites and conditions that have to match before the restriction adds. For our scenario we performed an easy one, just checking for the folder's category against the "Project" name.

4.3 Use Case 3 | Summary

Event handlers are extensions like job handlers. The significant difference is that they react to events in general, whereas job handlers respond to specific job types added to the queue. In other words, also a job might fire an event that the handler will pick up. It is powerful, and even more as it allows to restrict the execution easily. Not any task is best execute during events and better run on job processor. Consider circular iterations or dependencies; events called as a sub-event fire another loop before the calling event finished. Job handling easier allows splitting a complicated job into sub-jobs, which perform subsequently. Job processing allows putting sub-sequent jobs to the queue. If you limit a particular job type to a single processor, the calling job will have finished before the new initiated one starts.

And last but not least, consider that events are handled and performed by the user logged in to Vault; so, this user's role and permissions determine the ability to execute.

5 Outlook

You walked through a variety of API samples and use cases! Thank you for your interest and resilience in reaching this last short chapter.

Do you already face tasks to be solved by custom programming? Are you thinking about “which extension type handles my necessary actions at best?”

We have some criteria and arguments that should help to decide quickly on the path to go.

	Standalone	Job Proc. Extension	Event Handler
Vault client installation required?	Yes – Computer license No – Server licensing	Yes	Yes
Task size	Small . . large	Small . . large	Small
Execution – point of time	Individual	<ul style="list-style-type: none"> • One time queue submission + priority • Time interval queue submission + priority 	Immediate