SD323470

Feel the Power Between Vault and PowerShell

Jeffrey Fishman
Autodesk, Inc.

Markus Koechl
Autodesk, Inc.

Learning Objectives

- Learn how to instantiate Vault objects with PowerShell and the Vault API
- Learn how to perform Vault operations with PowerShell and the Vault API
- Learn how to interact with Vault objects, such as vaulted files, numbering schemes, items, and user/group information
- Learn how to bring all of these concepts together to create a quick Vault data reporting utility

Description

The Vault client in itself is a powerful data utility, but just imagine there's so much more that can be done with the Vault software development kit (SDK) software. Learn how to use the versatility and flexibility of Microsoft PowerShell, and bring your logic to new levels of automation. Learn how to instantiate Vault objects, call Vault Web Service methods, and manipulate the data stored within Vault in a lightweight scripted .NET language.

Class Speakers

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.

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.

Introduction

Target audiences and instructional methods

This class aims to introduce simple workflows, the Vault objects, and the Vault Web Services that can be utilized to construct reports in a lightweight fashion. The target demographic is System Administrators and Vault Developers looking to create a quick, light script to query and export select Vault data in a readable report using Autodesk API.

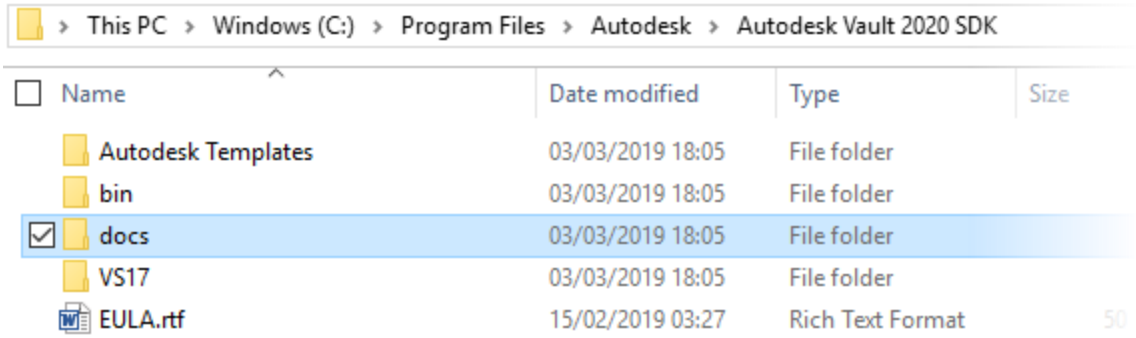
This class will demonstrate a ground-up approach to setting up a development environment, followed by scripting a simple data reporting utility with PowerShell.

Setting up the SDK environment

Installing the Vault SDK

Each Vault Client installation includes a full SDK that contains API help documents, code samples in C#, VB, and C++, as well as templates that can be used to build custom extensions, jobs, or other 3rd party applications for Vault. As a quick note, the Autodesk Vault API doesn't require an additional license in general. The Vault API and SDK also include 3rd party UI binaries from DevExpress. These binaries are redistributable and free to use within the context of the Vault API. However, direct use of these binaries requires a developer license – in this case, obtain a license 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***.



Installing Visual Studio 2017 / 2019

Visual Studio 2017 and 2019 are available as free (Community) and commercial (Professional, Enterprise) editions, which can be downloaded from Microsoft. All the material here will be scripted with PowerShell, so we'll want to have the PowerShell Tools for Visual Studio extension installed as well. As a free, open-source alternative, Visual Studio Code with the PowerShell extension can be used as well.

Setting up for PowerShell

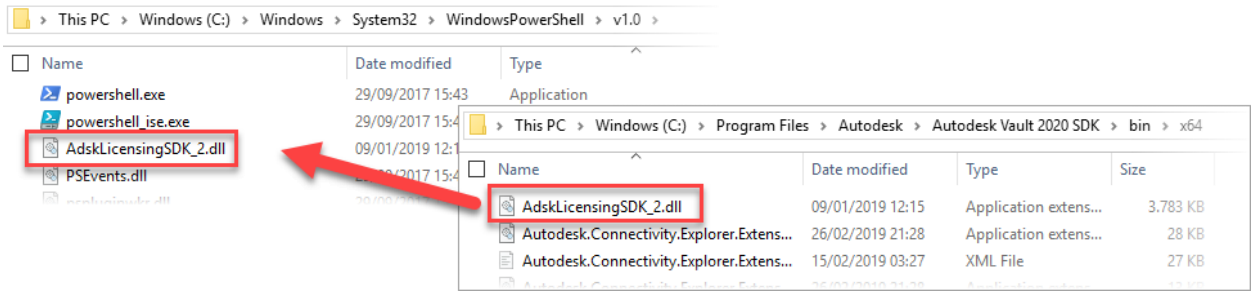
Before you start using PowerShell with the Vault API, ensure that the execution policy allows running unsigned scripts on the client. You can do that with the command listed below – please note, this required administrator rights!

```
Set-ExecutionPolicy RemoteSigned
```

Required binaries

You'll want to load the Autodesk.Connectivity.WebServices binary in your script from the SDK installation's ***bin*** directory. Ensure that the licensing binary is present in the directory you're loading from as well – without it, you'll receive an error when trying to log in! You can also copy the required Autodesk.Connectivity.WebServices and Autodesk.Connectivity.WebServices.* binaries to the same directory as the executing script / file.

Alternatively, you can copy the licensing binary to the same directory as where PowerShell.exe is located, at ***C:\Windows\System32\WindowsPowerShell\v1.0***.



Available SDK binaries:

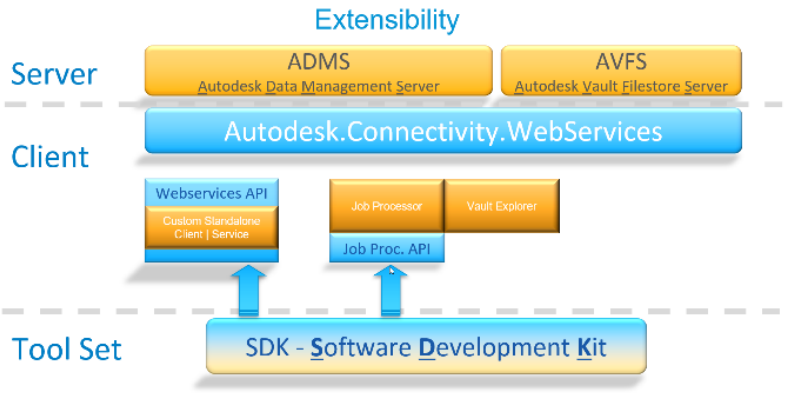
- Licensing Binary (The gate-keeper)
  - AdskLicensingSDK_2
- Connectivity Binaries (The meat and potatoes)
  - Autodesk.Connectivity.WebServices
  - Autodesk.Connectivity.WebServices.WCF
  - Autodesk.Connectivity.WebServices.Interop
- Extensibility Binaries (The extra goodies)
  - Autodesk.Connectivity.Explorer.Extensibility
  - Autodesk.Connectivity.Explorer.ExtensibilityTools
  - Autodesk.Connectivity.Explorer.Extensibility.Framework
  - Autodesk.Connectivity.JobProcessor.Extensibility
  - Autodesk.DataManagement.Server.Extensibility
- Framework Binaries (The handy helpers)
  - Autodesk.DataManagement.Client.Framework
  - Autodesk.DataManagement.Client.Framework.Forms
  - Autodesk.DataManagement.Client.Framework.Vault
  - Autodesk.DataManagement.Client.Framework.Vault.Forms

If using any GUI components, then the additional DevExpress binaries from the SDK's bin directory should also be referenced or copied to the same directory as the executing script / file.

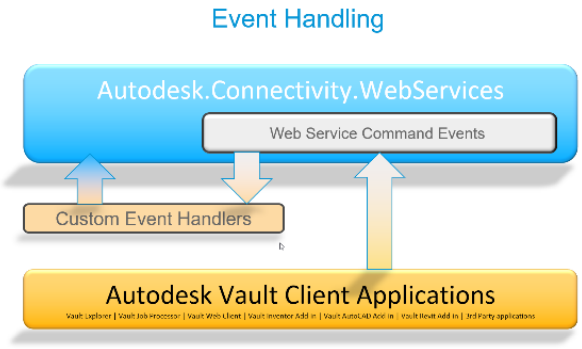
Autodesk Vault SDK Overview

API structure

Vault has a client / server architecture. The SDK accesses the server information using the client's WebServices API, which is the meat and potatoes of the SDK. Every data call will require this. The Web Services API lives in the Autodesk.Connectivity.WebServices assembly.



In order to utilize the SDK, a Vault client or server installation is required. In addition, the SDK allows you to create custom job handlers and custom event handlers.



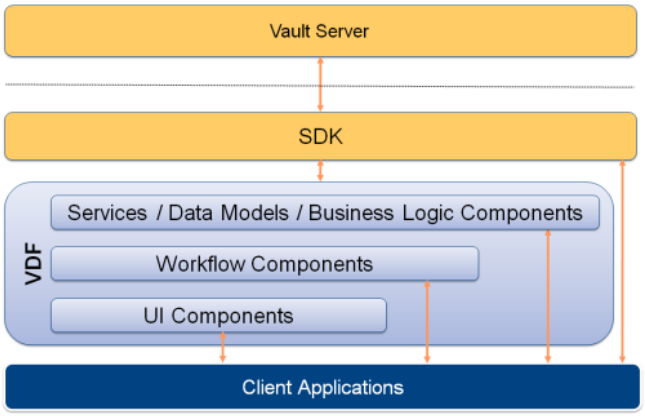
Included tools and resources

Documentation for all classes and functions are available as part of the SDK installation. In addition, sample applications in a variety of .NET languages can be found, including select project templates that can be added to your Visual Studio installation. For every new product year, a change log is also included.

The Vault Developer Framework (VDF)

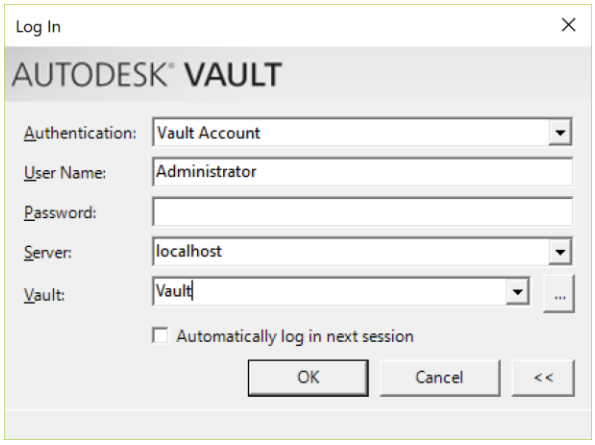
What is it?

VDF is a high-level framework that sits on top of the existing Vault API, and is included as part of the SDK. Its primary goal is to ease access to general API functionality with light-weight data management components!



What can be done with it?

This framework can be a primary means for accessing web services, data management business logic, and GUI components. VDF components will not be covered as part of this lecture, but they certainly are handy to have around when building a client, extension, or add-in and can help reduce development time required.



How can VDF be installed?

You can find and run the installation executable in the same location as where the Vault Client is installed:

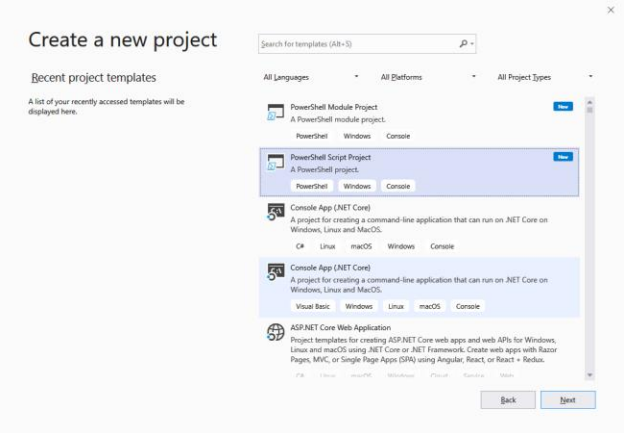
- C:\Program Files\Autodesk\Vault Professional 20xx\SDK\Setup.exe

Creating and Manipulating Vault Objects With PowerShell

Getting started

Like creating Vault objects with other .NET languages such as C#, VB.Net, and Visual C++, a majority of Vault objects need to be instantiated (with the *'New-Object'* qualifier for PowerShell), such as a log-in credential, lifecycle definitions and states, numbering schemes, property definitions, etc.

To kick things off, let's start up Visual Studio and create a new PowerShell project.



We'll then load up the Autodesk.Connectivity.WebServices binary from our SDK's ***bin*** directory:

```
[System.Reflection.Assembly]::LoadFrom('C:\Program Files\Autodesk\Autodesk Vault ' +
    $VaultVersion +' SDK\bin\x64\Autodesk.Connectivity.WebServices.dll')
```

From there on, we can begin creating our desired Vault objects.

Creating a simple credentials object for logging in

When using the Web Service layer directly to log in, a UserPasswordCredentials object can to be created. This holds the server identities object, Vault name, account name, account password, and optionally the connection type for the license.

First, we'll create our server identities object – this is passed to the UserPasswordCredentials object as the first parameter.

```
$serverIdentities = New-Object Autodesk.Connectivity.WebServices.ServerIdentities
```

After we've created our identities object, we'll populate its DataServer and FileServer properties with the appropriate server addresses. For simple installations, these values will be identical.

```
$serverIdentities.DataServer = "127.0.0.1"
$serverIdentities.FileServer = "127.0.0.1"
```

Now we'll specify how we want to connect to our servers with the LicensingAgent enumeration. As of the 2019 API version, we can select a (Client, Server, or None) license type. Please note, if you choose to use to use the Client type, your local machine will require an installation of the Vault client to be present and activated.

```
$licenseType = [Autodesk.Connectivity.WebServices.LicensingAgent]::Server
```

Finally, we'll create a UserPasswordCredentials object and pass in our identities as our first parameter. The other parameters are ordered as follows: *Vault name, user name, password, and connection type flag / read-only boolean*. We'll log in with an administrator account for now.

```
$userPasswordCredentials = New-Object Autodesk.Connectivity.WebServicesTools.UserPasswordCredentials(
    $serverIdentities, "Vault", "Administrator", "", $licenseType)
```

Once we've constructed our credentials object, we'll pass it to a new WebServiceManager object, which serves as our gateway to the Web Services and their methods.

```
$serviceManager = New-Object Autodesk.Connectivity.WebServicesTools.WebServiceManager($userPasswordCredentials)
```

Once we get here, we can place a breakpoint on the web service manager to test for proper connectivity. If all is good, we can now utilize the available web services to query data!

Calling Vault Web Services

Calling the Admin web service for user information

Now that we've established a connection, we'll query the server for some detailed user information. We'll begin by getting an array of User objects, which tells us who the user is, what kind of account it is (Local Vault or Active Directory), contact info, and if the account is active, returned as a User object:

```
$users = $serviceManager.AdminService.GetAllUsers()
```

The primary information we want to glean from this is the user ID, we'll use it to collect more information about the user's groups, roles, and Vault associations. Let's go ahead and pull out the user IDs now, and store them in an array:

```
$userIds = $users | ForEach-Object { $_.Id }
```

We can then pass this array to the GetUserInfosByUserIds() method, which returns UserInfo objects, which in turn stores User, Role, KnowledgeVault, and Group objects or object arrays:

```
$userDetails = $serviceManager.AdminService.GetUserInfosByUserIds($userIds)
```

userinfo	Autodesk.Connectivity.WebServices.UserInfo[]	Autodesk.Connectivity....
[0]	Autodesk.Connectivity.WebServices.UserInfo	Autodesk.Connectivity....
User	Autodesk.Connectivity.WebServices.User	Autodesk.Connectivity....
Roles	Autodesk.Connectivity.WebServices.Role[]	Autodesk.Connectivity....
[0]	Autodesk.Connectivity.WebServices.Role	Autodesk.Connectivity....
Id	1	System.Int64
Name	Administrator	System.String
IsSys	True	System.Boolean
Descr	Full control over all folders and administrative server privileges	System.String
Vaults	Autodesk.Connectivity.WebServices.KnowledgeVault[]	Autodesk.Connectivity....
[0]	Autodesk.Connectivity.WebServices.KnowledgeVault	Autodesk.Connectivity....
Id	1	System.Int64
Name	Vault	System.String
CreateDate	8/3/2009 1:15:50 PM	System.DateTime
CreateUserId	2	System.Int64
[1]	Autodesk.Connectivity.WebServices.KnowledgeVault	Autodesk.Connectivity....
Id	4	System.Int64
Name	Vault2	System.String
CreateDate	10/23/2019 11:03:23 AM	System.DateTime
CreateUserId	2	System.Int64
Groups	Autodesk.Connectivity.WebServices.Group[]	Autodesk.Connectivity....
[0]	Autodesk.Connectivity.WebServices.Group	Autodesk.Connectivity....
Id	1	System.Int64
Name	Everyone	System.String
EmailDL		System.String
CreateUserId	0	System.Int64
CreateDate	8/3/2009 5:15:25 PM	System.DateTime
IsActive	True	System.Boolean
IsSys	True	System.Boolean
Auth	Vault	Autodesk.Connectivity....

Collecting all Items

Now that we've collected users, we can go a little more advanced and collect all item information with the Item Service. The quickest and most efficient way to do so is to use a search using the FindItemRevisionsBySearchConditions method! We'll begin by constructing our search condition object with the following:

```
$searchCondition = New-Object Autodesk.Connectivity.WebServices.SrchCond
```

We'll need to specify a few things for the search condition to match what we want – specifically, we don't want the search to filter out any Item objects, so we'll have it include everything. To do so, we'll need to direct the search type to find all properties (*AllProperties*). Since we have this search type, and as listed within the SDK help documentation for constructing the search condition, the search operator must also be set to Contains (*1*). For the search rule, there are three values that can be selected: Must, May, and May Not. These can be interpreted as And,

Or, and Not, respectively. We'll specify Must for this search. At this point, we'll be searching with the equivalent of (*All Properties Must Contain "*"):

```
$searchCondition = New-Object Autodesk.Connectivity.WebServices.SrchCond
$searchCondition.PropTyp = [Autodesk.Connectivity.WebServices.PropertySearchType]::AllProperties
$searchCondition.SrchRule = [Autodesk.Connectivity.WebServices.SearchRuleType]::Must
$searchCondition.SrchOper = 1
$searchCondition.PropDefId = 0
```

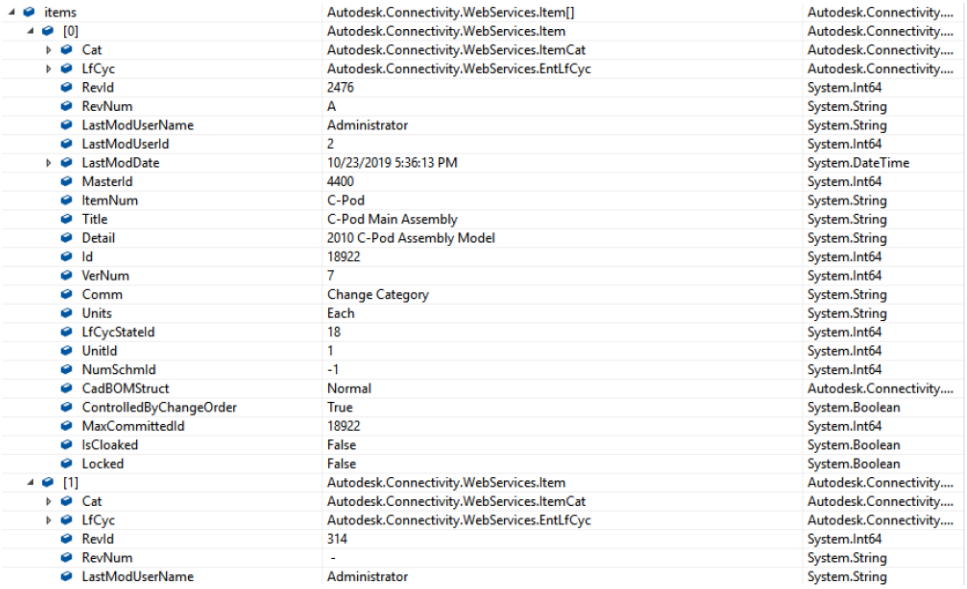
We're not really worried about sorting our results right now, nor about the search status, so we can create some empty new objects for those to pass along to our FindItemRevisionsBySearchConditions method. We're also not getting a very large Item set, so we'll pass along an empty bookmark. If the Item set is too large to get in one go, however, you'll want to use the returned bookmark in your next search call.

```
$searchSort = New-Object Autodesk.Connectivity.WebServices.SrchSort
$searchStatus = New-Object autodesk.Connectivity.WebServices.SrchStatus
$bookmark = ""
```

Now that we've set everything we've needed up for our call to retrieve Items, we can go ahead and pass them in to our method:

```
$items = $serviceManager.ItemService.FindItemRevisionsBySearchConditions(
    $searchCondition,
    $searchSort,
    $true,
    [ref]$bookmark,
    [ref]$searchStatus
)
```

Our results from this call will be an array of all items that could be retrieved from that one call!



items	Autodesk.Connectivity.WebServices.Item[]	Autodesk.Connectivity....
[0]	Autodesk.Connectivity.WebServices.Item	Autodesk.Connectivity....
Cat	Autodesk.Connectivity.WebServices.ItemCat	Autodesk.Connectivity....
LfCyc	Autodesk.Connectivity.WebServices.EntLfCyc	Autodesk.Connectivity....
RevId	2476	System.Int64
RevNum	A	System.String
LastModUserName	Administrator	System.String
LastModUserId	2	System.Int64
LastModDate	10/23/2019 5:36:13 PM	System.DateTime
MasterId	4400	System.Int64
ItemNum	C-Pod	System.String
Title	C-Pod Main Assembly	System.String
Detail	2010 C-Pod Assembly Model	System.String
Id	18922	System.Int64
VerNum	7	System.Int64
Comm	Change Category	System.String
Units	Each	System.String
LfCycStateId	18	System.Int64
UnitId	1	System.Int64
NumSchmId	-1	System.Int64
CadBOMStruct	Normal	Autodesk.Connectivity....
ControlledByChangeOrder	True	System.Boolean
MaxCommittedId	18922	System.Int64
IsCloaked	False	System.Boolean
Locked	False	System.Boolean
[1]	Autodesk.Connectivity.WebServices.Item	Autodesk.Connectivity....
Cat	Autodesk.Connectivity.WebServices.ItemCat	Autodesk.Connectivity....
LfCyc	Autodesk.Connectivity.WebServices.EntLfCyc	Autodesk.Connectivity....
RevId	314	System.Int64
RevNum	-	System.String
LastModUserName	Administrator	System.String

Light-weight Reporting Utility criteria

- Getting all Vault Users
- Getting Vault User Group, Role, and Membership info
- Getting Vault Items
- Getting Vault ECOs

Closing notes

Thank you for your interest and patience in this course and the information presented in it! Markus owns the complimentary course SD321955 - Autodesk Vault 2020—Programming 101, which contains quite a bit of info on how to utilize the Autodesk Vault API with various Vault workflows not covered here, such as creating custom Jobs, leveraging Vault event hooks, and giving a more in-depth look at the PowerShell samples provided as part of the SDK.