

1. Installation	2
2. Getting started	2
3. Activation and Trial limitations	3
4. Features	4
4.1 Executing jobs from the client	4
4.2 Executing jobs via lifecycle changes	5
4.3 The sample jobs	6
5. Configuration	7
5.1 Configure Pdf creation for AutoCAD Dwgs	7
6. Customization	10
6.1 Preparing your environment	11
6.2 IDEs for powershell	11
6.3 Powershell, scripts and modules	12
6.4 Structure of a powerJobs script	13
6.5 Error Handling	17
6.6 Adding jobs	18
6.7 Environment Variables	19
6.8 Code Reference	20
6.8.1 powerJobs	20
6.8.1.1 getPublisher	21
6.8.1.2 Job	21
6.8.1.3 Log	22
6.8.2 Publisher	23
6.8.2.1 add_OnBeginPublish	24
6.8.2.2 OnBeginPublish	24
6.8.2.3 Open	25
6.8.2.4 Publish	25
6.8.3 Vault	26
6.8.4 VaultConnection	28
6.8.5 vaultExplorerUtil	30
6.9 Code Snippets	30
6.9.1 Changing filestates, categories	31
6.9.2 Create a textfile via template	32
6.9.3 Create DWG from an IDW	35
6.9.4 Creating PDFs with their parents' properties	35
6.9.5 Export Vault Data to XML	36
6.9.6 General powershell codesnipptes	37
6.9.6.1 Convert office documents to Pdf	38
6.9.6.2 Copy file in a directory	39
6.9.6.3 Print something with the windows default printer	39
6.9.6.4 Send queries to a sql server	39
6.9.6.5 Set default printer	40
6.9.7 Print via Inventor	40
6.9.8 Retrive the user that queued the job	41
6.9.9 Selected files to ZIP	41
6.9.10 Send email via jobserver	42
7. Tutorials	44
7.1 Adding the file revision to the PDF name	44
7.2 Converting PDF to PDF/A	45
7.3 Convert PDF to DWF	48
7.4 PDF on Item Lifecycle Transition	49
7.5 How to handle differently sized AutoCAD DWGs	50
8. FAQ	52
8.1 Where is the documentation for the Vault API?	52
9. Troubleshooting	52
9.1 Logging Levels	52
9.2 DLL not found in Powergui	53
9.3 Jobprocessor freezes	53
9.4 Resetting the toolbars	54
9.5 Cannot Check in Inventor files if a Pdf is attached to the file	55
10. Change logs	55
11. What's new in powerJobs2014	55

Installation

Requirements

- Vault Workgroup Client 2014
- Vault Professional Client 2014
- powershell 2.0 or higher



Windows PowerShell 2.0 is part of your Windows 7. For earlier Windows versions, please follow this link for downloading the according PowerShell version: <http://support.microsoft.com/kb/968929>

After the Setup

After the setup of powerJobs, you have to **reset all toolbars**. You can do that with clicking on the Tools menupoint in Autodesk Vault and there click on customize. Now you should see all toolbars(Main menu, Help, Standard and son on) and reset every toolbar you have. Now you can see the powerJobs menupoint in the standard toolbar. In case any of the toolbars gets visual bugs restart vault after the reset.

Install Locations

powerJobs is installed in the following locations on your system:

- All program libraries and executable files are placed in **C:\ProgramData\Autodesk\Vault 2013\Extensions\coolOrange.PowerJobs** and **C:\ProgramData\Autodesk\Vault 2013\Extensions\coolOrange.PowerJobs.Handler**
- All job definition files, e.g. scripts and module libraries, are placed in **C:\ProgramData\coolOrange\powerJobs**
- Shortcuts to open the Vault Jobprocessor and to the powerJobs Configuration directory are placed in the **start menu** and on your **desktop**.
- A shortcut to the powerJobs documentation on the coolOrange Wiki is placed in the startmenu.

Updates

To install a newer version of powerJobs just execute the setup file of the new version. This will automatically update the files in the existing installation.

Uninstall

In case you want to remove powerJobs from your computer you can

- either execute the setup file again. This will give you the option to repair or remove powerJobs. Click on "Remove" to uninstall the program.
- or you can go to "Control Panel - Programs and Features", find "coolOrange powerJobs 2014" and run "Uninstall

Getting started

Creating a PDF from a Vaultfile

With powerJobs you can easily do different things, for instance creating a PDF-file from a Vaultfile (iam, ipt, idw or dwg). This is very simple because powerJobs offers you the possibility to do this by using a menubutton powerJobs -> Create PDF. So all you have to do is:

- 1 Creating a PDF from a Vaultfile
- 2 Select a Vaultfile
- 3 Create PDF
- 4 Job Queue
- 5 Job Processor

Select a Vaultfile

In this example we have selected the "**Catch Assembly.iam**". Afterwards the **powerJobs** button will be enabled.

Create PDF

Click on the **powerJobs** button and select "**create PDF**".

If you don't see this button then you have to reset the toolbars: Tools->Customize and reset all of them, then close the window.

Job Queue

If you open the job queue now, you will see your job in here.

You can find the job queue in the menu at **Tools -> Job Queue**.

Job Processor

In order to execute the job you have to start the job processor. You can either use the shortcut on the desktop or start it directly.

The jobprocessor is located at "**C:\Program Files\Autodesk\Vault Professional 2014\Explorer\Jobprocessor.exe**"

Result

After the jobprocessor has finished turn back to your Vault client and refresh it either via "**F5**" or the refresh button. Your Pdf should be next to your engineering document.

You can also try the other jobs we deliver. If you click on "**powerJobs -> Folder > PDF**" while you have selected a folder you can create PDFs for every engineering document in this folder. Also you could try the Dwf or Dxf job through "**powerJobs -> Job Dialogue**" or customize the jobs or even create your own if the default jobs doesn't suit your needs.

If you want to do this a good start would be the [customization](#) part of our wiki.

In case you want to execute jobs on lifecycle changes take a look at this [article](#).

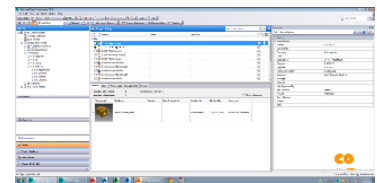
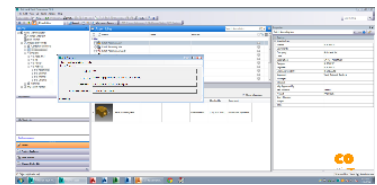
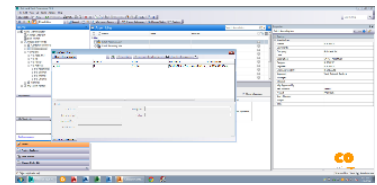
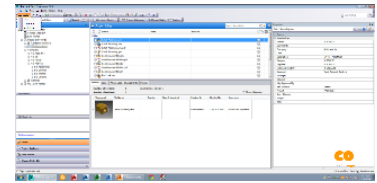
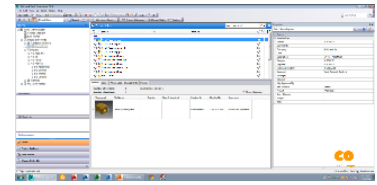
Activation and Trial limitations

Trial limitations

There is no difference in functionality between the trial version and the fully licensed product.

Trial duration

After the installation our products are available as a trial version for 30 days.



Use Trial

If you click **Later** in the Activate dialog, the product will continue in trial mode.

Order

With the button **Order** you will be referred directly to the coolOrange order page. Once there, follow the instructions to order the product.

Activate

Once you have received a S/N from coolOrange you can click the button **Activate**. In the following dialog you have three different options to activate your product:

Online Activation

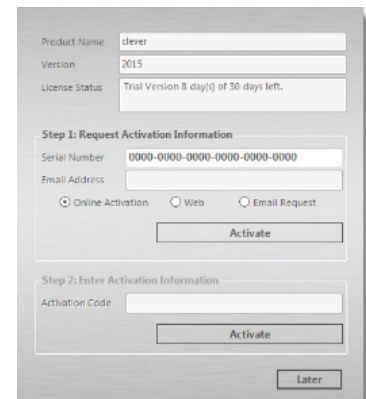
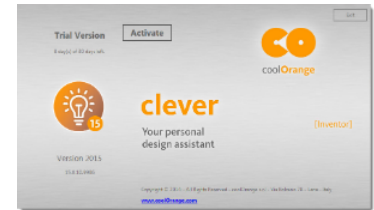
This is the preferred method but might not work if you use a proxy server to access the internet.

Web

This opens the activation website. If you fill the form you'll get an activation code.

Email Request

With "Email Request" an email gets send to register@coolOrange.net and you get your activation code via email. Before you can click the Activate button you need to enter your email address. This can be slow sometimes so the first two options should be preferred if possible.



Features

Executing jobs from the client

Executing jobs via lifecycle changes

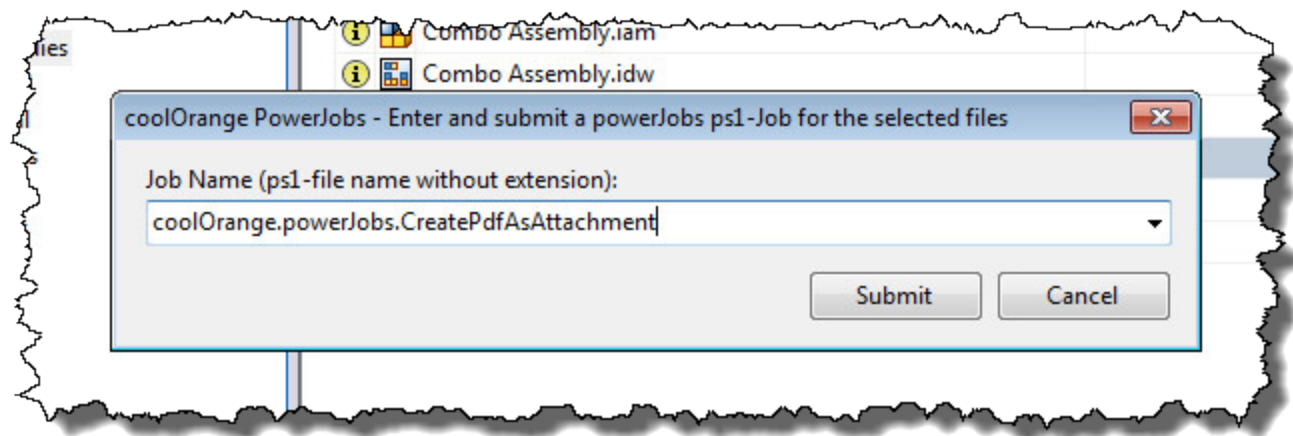
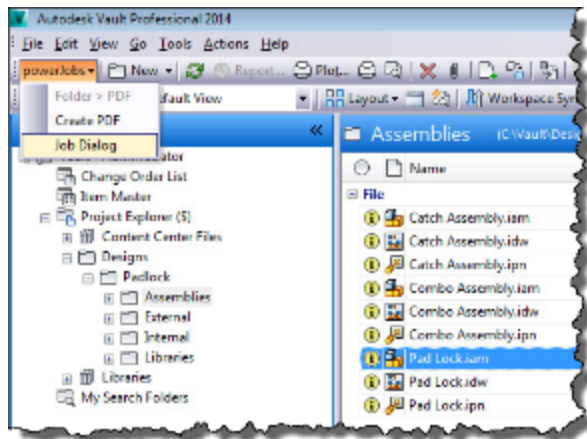
The sample jobs

Executing jobs from the client

To execute a job from the vault client you have to open the job dialogue, which can be found in the powerJobs menu.



The job dialogue item is only enabled if you have selected a file!



Executing jobs via lifecycle changes

i You will need the "lifecycle event editor" to configure the lifecycle transitions. It is part of the Vault sdk. The SDK installer is located in "C:\Program Files\Autodesk\Vault Professional 2014\SDK"

After you installed the sdk you can find the editor in "C:\Program Files (x86)\Autodesk\Autodesk Vault 2014\SDK\util\LifecycleEventEditor"

Example

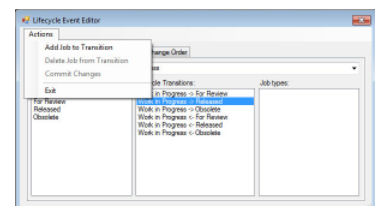
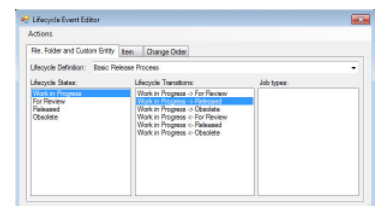
You want to add a the job during the transition from **"Work in Progress"** to **"Released"**. Our example job is **myCompany.myJob.ps1**

Activate the tab **"File.Folder and CustomEntity"**

Select **"Work in Progress"** in the "Lifecycle states list

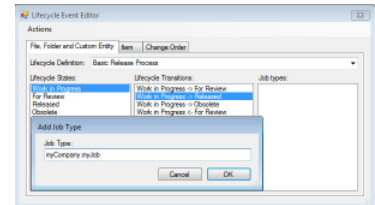
Select **"Work in Progress -> Released"** in the Lifecycle Transitions list

Open the **"Add Job to Transition"** dialogue



Enter the name of your job in the text field.

i Do not forget to commit your changes. Otherwise your work is gone.



The sample jobs

- Jobs
 - CreatePdfAsAttachment
 - CreateDwfxAsAttachment
 - CreateInventorDwg
 - CreatePdfForAllFilesInFolder
 - SaveLocalAsSheetMetalDxf
- Modules
 - CadHelper
 - VaultHelper

Jobs

1. coolOrange.powerJobs.CreatePdfAsAttachment.ps1
2. coolOrange.powerJobs.CreateDwfxAsAttachment.ps1
3. coolOrange.powerJobs.CreateInventorDwg.ps1
4. coolOrange.powerJobs.CreatePdfForAllFilesInFolder.ps1
5. coolOrange.powerJobs.SaveLocalAsSheetMetalDxf.ps1


CreatePdfAsAttachment

The job coolOrange.powerJobs.CreatePdfAsAttachment creates PDF Visualization Attachments for the following types:

- dwg
- idw
- iam
- ipt

If the job is executed for different types nothing will happen.

You can execute this job from the Vault client when you highlight the file that you want to create a PDF for and then click the powerJobs->Create PDF menu command. This will trigger the queueing of the coolOrange.powerJobs.CreatePdfAsAttachment job. Alternatively you can add this job to a lifecycle state transition via the LifeCycleEvent Editor. It will create a PDF file with the same filename as the cad file

 The job's script can be edited to allow other extensions and to create different filenames.

CreateDwfxAsAttachment

coolOrange.powerJobs.CreateDwfxAsAttachment creates dwfx Attachments for the following file types:

- ipt
- iam

The dwfx files will have the same name as the CAD file but with the extension dwfx. They are placed beside the CAD files but have no links to them.

CreateInventorDwg

coolOrange.powerJobs.CreateInventorDwg creates an Inventor Dwg file for following file types:

- ipt
- iam

It is meant as an example to demonstrate how one can use the SaveAs function in Inventor to create different file formats.

CreatePdfForAllFilesInFolder

- dwg
- idw

SaveLocalAsSheetMetalDxf

The outputfile will be written in "C:\TEMP"

Modules

- # CadHelper

VaultHelper

Configuration

Configure Pdf creation for AutoCAD Dwg

These locations are:

- Vault
- TrueViewSetup.dwg
- Dwg TrueView options
 - Plot Stamp Settings...
- Job
- See also

Vault

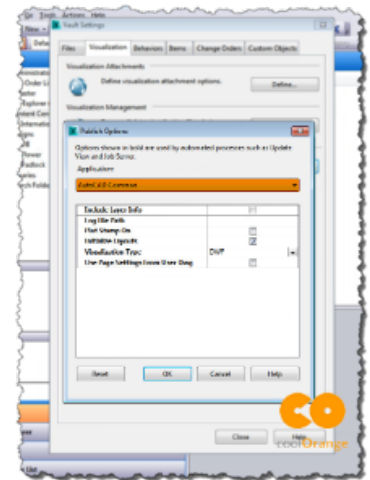
There are two important categories:

AutoCAD Common



Only the **bold** entries are used by the Jobprocessor

Include Layer Info	Includes the layer information in the Pdf so that you can select the different layers
Plot Stamp On	If it is checked a plot stamp will be made on the Pdf
Initialize Layouts	Must be checked or you won't get any Pdfs
Use Page Settings From User Dwg	When checked the Jobprocessor will ignore powerJobs' TrueViewSetup.dwg. Instead it will use the settings from the DWG itself.

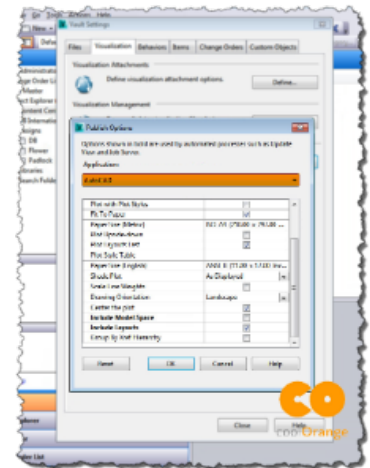


and AutoCAD



AutoCAD has higher priority than the more specialized categories like **AutoCAD Mechanical**. If you uncheck both of these settings in **AutoCAD** and check them in **AutoCAD Mechanical** you still won't get any content into your Pdf.

Include Model Space	Include/Exclude the model space of the dwg.
Include Layouts	Include/Exclude the layouts of the dwg.



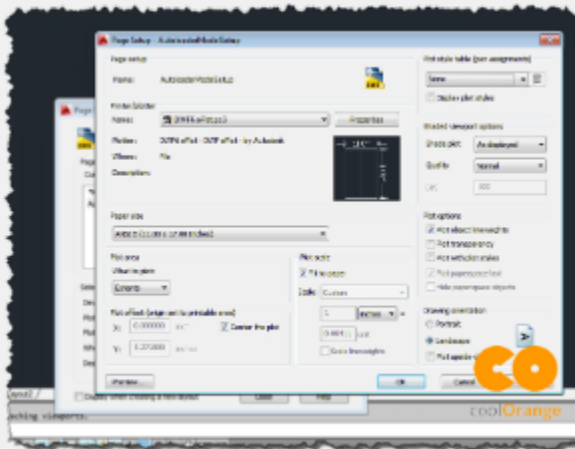
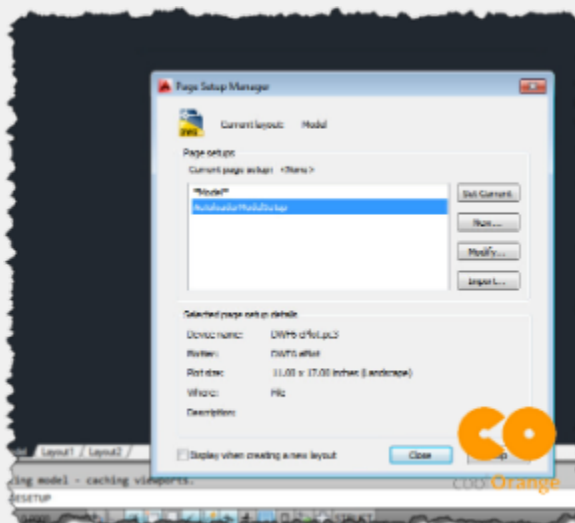
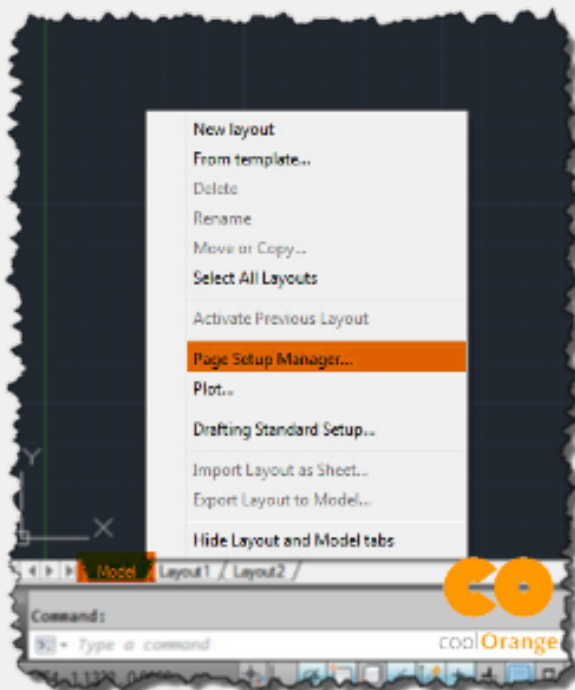
TrueViewSetup.dwg

PowerJobs comes with its own TrueViewSetup.dwg.



Make a backup of the TrueViewSetup.dwg before you make changes to it.

The **TrueViewSetup.dwg** is located at "**C:\ProgramData\Autodesk\Vault 2014\Extensions\coolOrange.PowerJobs.Handler**"



In order to edit the settings rightclick on your model or layout tab and choose "**Page Setup Manager...**". Select the **Autoloader ModelSetup** or **AutoloaderLayoutSetup** and click on "**Modify...**".



The tab you wish to edit has to be selected.

Dwg TrueView options

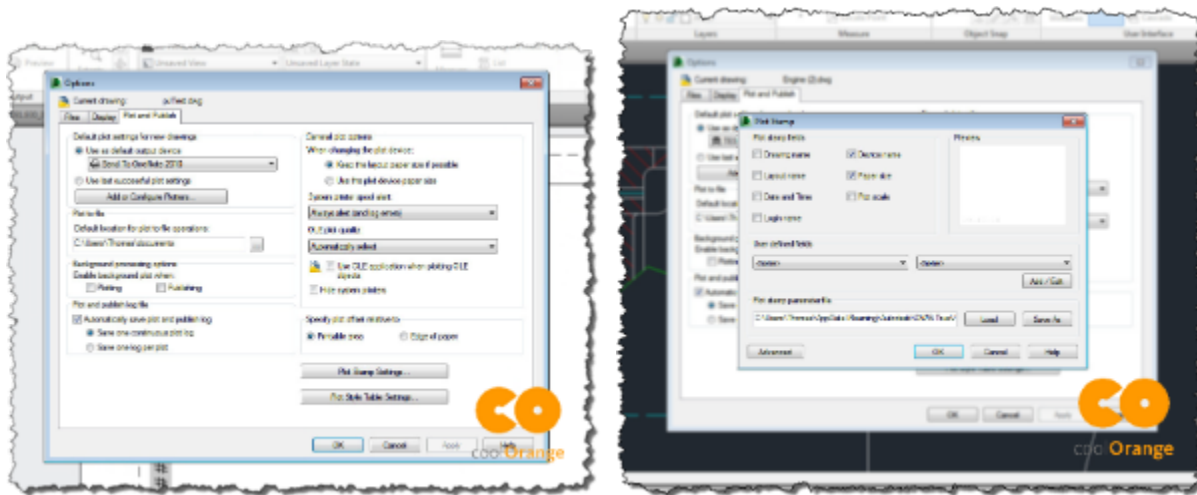
Some things cannot be configured in Vault or the TrueViewSetup.dwg. Stuff you would normally configure in AutoCAD has to be done in DwgTrueView, because it is used for the Pdf generation instead of AutoCAD.



To be able to access the DwgTrueView options you have to open a file with it. Otherwise the options are not visible.

Plot Stamp Settings...

"**Plot Stamp Settings...**" is the only thing in here, which is used by powerJobs Pdf generation. It is located in the "**Plot and Publish**" Tab of the TrueView options. For detailed information look up the [Autodesk knowledgebase](#) please.



Job

The job **coolOrange.powerJobs.CreatePdfAsAttachment.ps1** contains a settings region. There you can config if the Pdf should be visible in vault, how its name is generated and which file types are eligible for Pdf creation.

```
#region Settings
$showPDF = $true #change this setting to $true or $false for showing/hiding the PDF
$PDFfileName = $file.EntityName + ".pdf" #define here the file name for the
generated PDF
$inventorExtensions = @(".idw",".dwg",".iam",".ipt") #Edit the list to restrict PDF
generation to certain file types
#endregion
```

See also

- [Page Setup Manager](#)
- [Vault Publish Options](#)
- [Plot Stamp Settings](#)

Customization

- [Preparing your environment](#)

- IDEs for powershell
- Powershell, scripts and modules
- Structure of a powerJobs script
- Error Handling
- Adding jobs
- Environment Variables
- Code Reference
- Code Snippets

Preparing your environment

Before you can begin to create your own jobs you have to prepare your development environment.

1. Choose a IDE and install it if needed. ([powershell IDEs](#))
2. Enter your Vault connection data in **function PrepareEnvironment**. The function is part of **coolOrange.powerJobs.VaultHelper.psm1**.

```
#(Loginname, Password, Vaultserver, Vaultname)
[coolOrange.PowerJobs.PowerShellVaultProxy]::Instance.Login("Administrator", "", "localhost", "Vault")
```

IDEs for powershell

- PowerGUI
- Powershell 2.0 ISE

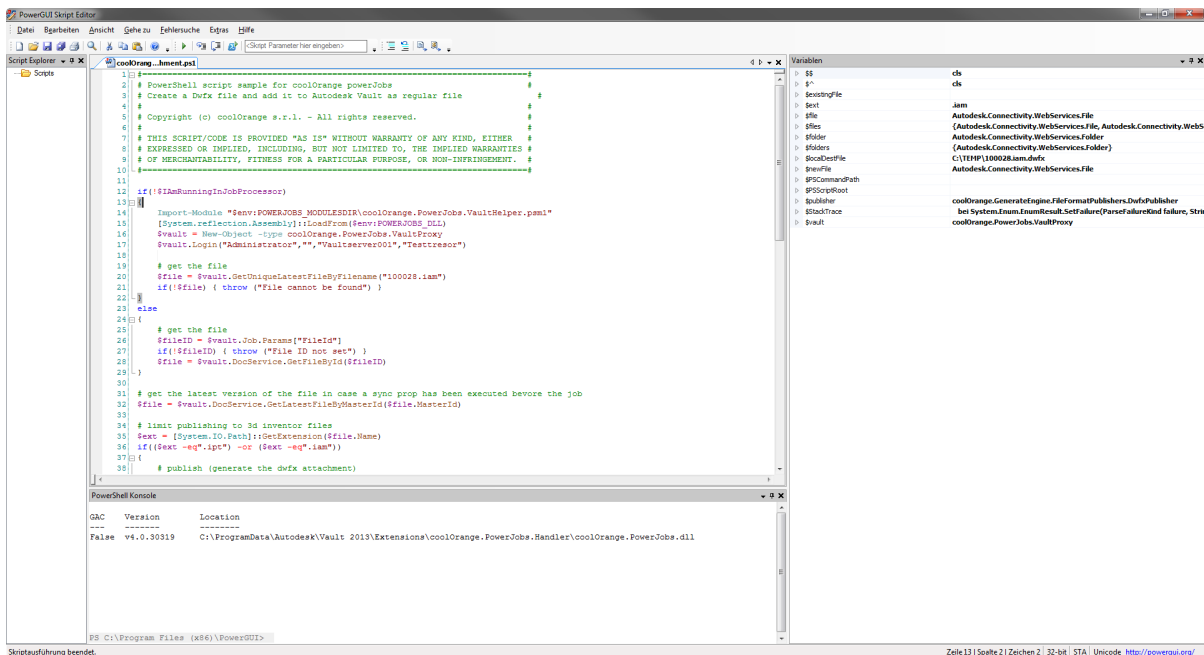


If you have a 64 bit Vault client installed, you have to use a 64 bit IDE. If you have 32 bit Vault client installed, you have to use a 32 bit IDE.

There are several IDEs for powershell available. Of course you can use whichever you like. We present the following two, because they are free of charge.

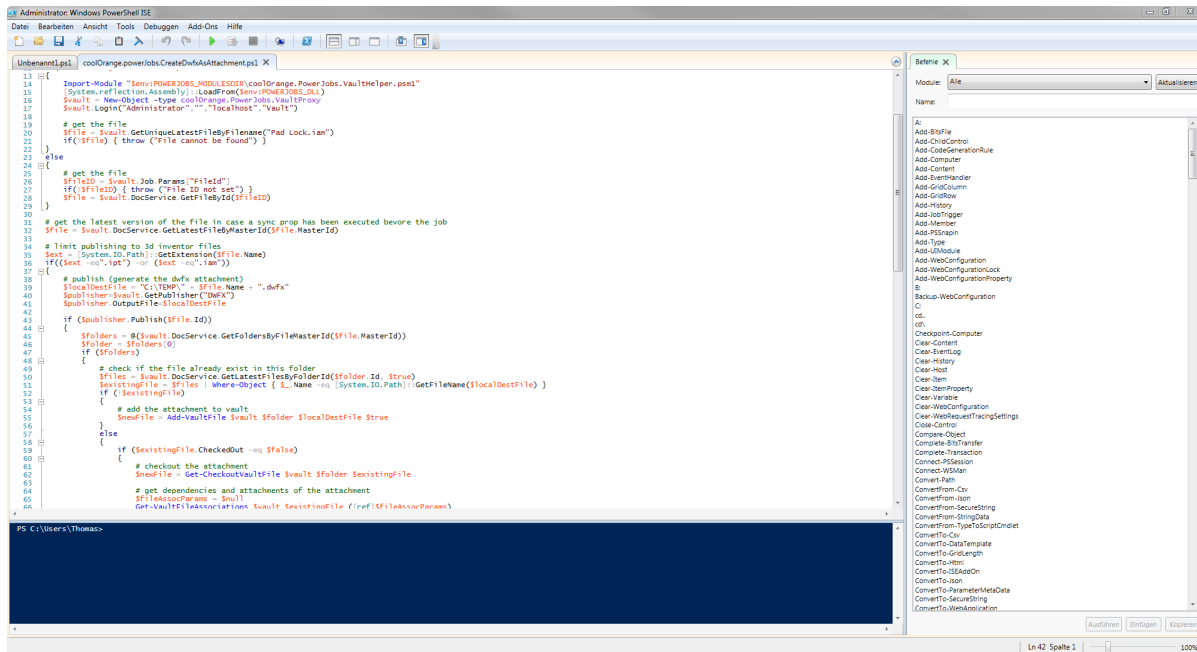
PowerGUI

Another powershell development environment is PowerGUI. It is also free and you can download it from <http://www.powergui.org>.



Powershell 2.0 ISE

With PowerShell 2.0 you get a free development environment, called PowerShell ISE.



Support for .net 4

Out of the box, the powershell ISE is configured to use the .net 2.0 runtime. However, to run and debug scripts for powerJobs 2013 the ISE must be configured to use the .net 4 runtime. You need to provide a config file for that. Just create a textfile named powershell_ise.exe.config and paste the following lines into it:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0.30319" />
  </startup>
</configuration>
```

Then save and copy the file besides the powershell_ise.exe.

Powershell, scripts and modules

Powershell

Windows PowerShell® is a task-based command-line shell and scripting language designed especially for system administration. Built on the .NET Framework, Windows PowerShell helps IT professionals and power users control and automate the administration of the Windows operating system and applications that run on Windows.



More information and learning resources can be found on microsoft TechNet [Getting Started with Windows PowerShell](#)

Source: [Microsoft TechNet](#)

Scripts

A script is a list of commands which are executed by a scripting engine. Contrary to normal source code scripts don't require to be compiled. The engine will interpret them and do the rest. This allows for an easy and flexible workflow, even if you have no programming skills.

Scripts have the file extension **ps1**.

Modules

PowerShell modules provide an efficient, manageable and production-oriented way to package code.

E.g. In one job you create a function for sending e-mails with a PDF attachment, but you don't want to copy this function every time you need it in a new job. Instead you can save the function in a module and import it every time you need the function. Modules are very similar to classes in object oriented programming languages.

Modules have the file extension **psm1**.

PowerJobs modules are located by default in the folder: C:\ProgramData\coolOrange\powerJobs\Modules

Structure of a powerJobs script

- [Preperation](#)
- [Execution](#)
- [Template for a publishing job](#)

PowerJobs scripts consists of two parts. The preperation and the execution of the script.

Preperation

With the comand "**PrepareEnvironment**" you can prepare your debug environment. It grants access to the objects "**\$vault**", "**\$vaultConnec**
tion", "**\$vaultExplorerUtil**" and "**\$powerJobs**".

```
PrepareEnvironment
```

On the top of script the **VaultHelper** module should be imported. This looks like this.

```
Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
```

"**PrepareEnvironmentForFile**" gets the right file for your environment. If the job is executed in the jobprocessor, the file is taken from the "**\$powerJobs.Job**" parameter. If it's executed in debug mode, the file will be searched by name in the first parameter.

In the second parameter you can specify if you want the latest version of the file or not.

```
$file = PrepareEnvironmentForFile "Catch Assembly.idw" $true
```



With "**\$powerJobs.Log.xxxx**" you have direct access to powerJobs logging functionality.

It is a good practice to define a settings region in the preperation part of your script. This way you can easily change things like filepaths, filenames, filetype restrictions and much more.

```
#region Settings
$showPDF = $true #change this setting to $true or $false for showing/hiding the PDF
$PDFfileName = $file.EntityName + ".pdf" #define here the file name for the
generated PDF
$inventorExtensions = @(".idw",".dwg",".iam",".ipt")
#endregion
```

The whole preperation part in our CreatePdfAsAttachment example looks like this.

```
Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
# creating the $vault,$vaultConnection,$powerJobs and $file object for the Debugger
and for the PowerJobs environment
$file = PrepareEnvironmentForFile "Catch Assembly.idw" $true
$powerJobs.Log.Info("Starting job 'Create PDF as attachment' ...")
#region Settings
$showPDF = $true #change this setting to $true or $false for showing/hiding the PDF
$PDFfileName = $file.EntityName + ".pdf" #define here the file name for the
generated PDF
$inventorExtensions = @(".idw",".dwg",".iam",".ipt")
#endregion
# limit publishing to inventor files
$ext = [System.IO.Path]::GetExtension($file.EntityName).ToLower()
if( $inventorExtensions -contains $ext )
```

Execution

In the execution part of the script is the actual logic of your job located. Here is the whole work done. Basically you can do everything you want in here.

```
{
    # publish (generate the pdf attachment)
    ...
    $powerJobs.Log.Warn("Files with extension: '$ext' are not supported");
}
```

Template for a publishing job

The *coolOrange.powerJobs.DownloadVaultFile.psm1* must be in the powerJobs modules folder.

[More infos about modules](#)

```

fu
    Param(
        [Parameter(Mandatory=$true, Position=0)]
        $file,
        [Parameter(Mandatory=$true, Position=1)]
        [String]$origpath
    )
    try{
        #region DownloadSettings
        #Generate a Settingsobject. It is necessary for the new AcquireFiles method
        $settings = New-Object
Autodesk.DataManagement.Client.Framework.Vault.Settings.AcquireFilesSettings($vaultc
onnection, $false)
        $settings.AddEntityToAcquire($file)
        #Temporary path for the downloaded files
        $settings.set_LocalPath($origpath)
        #"Checkout", "Download", "NoAction"are possible values
        $settings.set_DefaultAcquisitionOption("Download")
        #What should be included?

$settings.OptionsRelationshipGathering.FileRelationshipSettings.set_IncludeChildren(
"FALSE")

$settings.OptionsRelationshipGathering.FileRelationshipSettings.set_RecurseChildren(
"FALSE")

$settings.OptionsRelationshipGathering.FileRelationshipSettings.set_IncludeAttachmen
ts("FALSE")
        #endregion
        #region create folders
        #It creates a new folder even though it exist
        if( !(Test-Path -Pathtype Container -Path $origpath) ){
            (New-Item -Path $origpath -ItemType Directory) > $null
        }
        #endregion
        #region Downloading the file
        $allDownloadedFiles = $vaultConnection.FileManager.AcquireFiles($settings)
        $allDownloadedFiles.FileResults | where-object {$_.File.EntityIterationId -eq
$file.EntityIterationId }
        #endregion
    }
    catch{
        throw("Failed to aquire files")
    }
}

```

```

Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psml"
Import-Module
"$env:POWERJOBS_MODULESDIR\coolOrange.powerJobs.DownloadVaultFile.psml"
# creating the $vault,$vaultConnection,$powerJobs and $file object for the Debugger
and for the PowerJobs environment
$file = PrepareEnvironmentForFile "Catch Assembly.idw" $true
#region Settings

```

```

$showPDF = $true #change this setting to $true or $false for showing/hiding the PDF
$PDFfileName = $file.EntityName + ".pdf" #define here the file name for the
generated PDF
$downloadFolder = "C:\TEMP\VaultDownload"
#endregion
$localDestFile = "C:\TEMP\" + $PDFfileName
$checkInAsHidden = !$showPDF

$downloadedFile = Download-VaultFile $file $downloadFolder
# here you can convert: $downloadedFile.LocalPath to the PDF file: $localDestFile
# ...
$folder=$vaultConnection.EntityOperations.GetParent($file)
if ($folder)
{
    # check if the file already exist in this folder
    $existingFile = Get-VaultFile $folder
    ([System.IO.Path]::GetFileName($localDestFile))
    if (!$existingFile)
    {
        $powerJobs.Log.Info("A new Pdf file will be added ...");
        $newFile = Add-VaultFile $folder $localDestFile $checkInAsHidden
    }
    else
    {
        if ($existingFile.IsCheckedOut -eq $false)
        {
            $powerJobs.Log.Info("Starting checkout of existing file ...");
            $newFile = Get-CheckoutVaultFile $existingFile

            $powerJobs.Log.Info("Reading file associations...");
            $fileAssocParams = Get-VaultFileAssociations $existingFile

            $powerJobs.Log.Info("Starting checkin of the pdf file ...");

            $powerJobs.Context.Log("Get-CheckinVaultFile",[Autodesk.Connectivity.JobProcessor.Ex
tensibility.MessageType]::eInformation);
            $newFile = Get-CheckinVaultFile $existingFile $localDestFile $fileAssocParams
            $checkInAsHidden
        }
        else
        {
            {
                throw("Destination file to be updated is checked out.")
            }
        }
    }

    # add generated file as design vizualization attachment
    Add-VaultDesignVizualizationFile $file $newFile
}
else
{
    throw("Getting the parent-folder failed.")
}

```



```
}
```

Error Handling

General information

Powerjobs handles errors in scripts differently from the normal PowerShell execution. When a script is executing in **powershell**, the default behavior is to **continue** the execution of the script when an error occurs in one of the commands. In **powerjobs** the errorhandling is changed so that an **exception** is thrown if an error occurs.



If the exception is not caught in your script the script will be cancelled and the exception will be caught in the powerJobs dll. The job will fail and an error message will be written to the log.

If you want to handle errors yourself in your script you can use PowerShell's built in exception mechanism and use **try/catch** blocks, or you can change the error handling behavior of PowerShell by changing the value of the **\$ErrorActionPreference** Variable.

\$ErrorActionPreference options:

Identifier	Numeric Value	Description
Continue	2	This is the default preference setting. The error object is written to the output pipe and added to \$error, and \$? is set to false. Execution then continues at the next script line.
SilentlyContinue	0	When this action preference is set, the error message is not written to the output pipe before continuing execution. Note that it is still added to \$error and \$? is still set to false. Again, execution continues at the next line.
Stop	1	This error action preference changes an error from a non-terminating error to a terminating error. The error object is thrown as an exception instead of being written to the output pipe. \$error and \$? are still updated. Execution does not continue.
Inquire	3	Prompts the user requesting confirmation before continuing on with the operation.. At the prompt, the user can choose to continue, stop or suspend the operation.



In powerJobs it is not recommended to use "Inquire" or numeric value 3.

Changing \$ErrorActionPreference value

To change the value of the \$ErrorActionPreference e.g. to "Continue" you can use the following statement:

```
$ErrorActionPreference = "Continue"
```

You can use the numeric value as well

```
$ErrorActionPreference = 2
```

Logging in jobs

We recommend to use powerJobs log object to add logging to your jobs. It is a member of the powerJobs object.

Example

```
try{
    $files = $vaultConnection.FileManager.AcquireFiles($settings)
}
catch{
    $powerJobs.Log.Error("Failed to aquire files")
}
```

Adding jobs

- Adding jobs to the directory
- Windows rights
- Adding jobs to the configuration
 - Enhanced job processor
- Verify that the job gets found

Adding jobs to the directory

Every job file has to be placed in "**C:\ProgramData\coolOrange\powerJobs\Jobs**"

Every module file has to be placed in "**C:\ProgramData\coolOrange\powerJobs\Modules**"

You can access this directory by double clicking on the powerJobs configuration Icon on your desktop, by using the powerJobs Configuration shortcut from the Start Menu or through your windows explorer.



If you add a job while the jobprocessor is running you have to restart it.



If you make changes to a script while the jobprocessor is running, the changes will be recognised the next time the job is executed.

Windows rights

Your windows user needs at least the following rights for the folder "**C:\ProgramData\coolOrange\powerJobs**"

"List folder / read data" and "Read extended attributes"

Adding jobs to the configuration



Change the config file always with full administration rights or without! Never swap between them, afterwards there could be an issue, where the Jobprocessor shows other job types than the current config file.

Open the file "**C:\ProgramData\Autodesk\Vault 2014\Extensions\coolOrange.PowerJobs.Handler\PowerJobs.vcet.config**"

You have to add your file to the element **<extension>**

This should look like this **<setting key="some unique entry" value="name of ps1 file without extension"/>**

E.g. <setting key="JobType6" value="coolOrange.PowerJobs.PowerJobHandler"/>

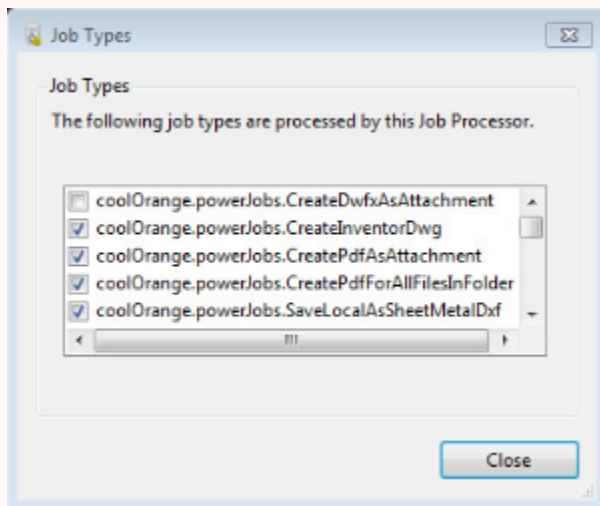
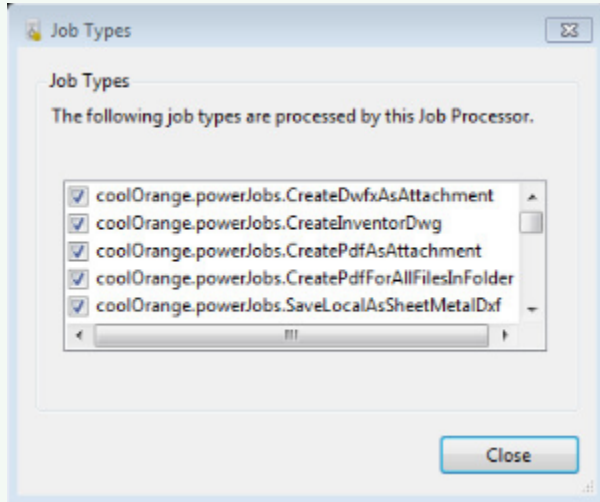
The adding of jobs to **PowerJobs.vcet.config** is new in 2014. Alternatively you can add your jobs to "**C:\Program Files\Autodesk\Vault Professional 2013\Explorer\JobProcessor.exe.config**", but we do not recommend it, cause it's prone to failure.

Enhanced job processor

The enhanced jobprocessor has its own configfile in which you have to add the powerJobs jobs. In the standard installation you gonna find it under "**C:\ProgramFiles\Autodesk\VaultProfessional####JobProcessor**".

Verify that the job gets found

Start your jobprocessor. If it is already running restart it. Open the "Job Types" dialogue. You will find it at "Administration" -> "Job Types". Scroll through the list and look for your job. If there is an entry for your job, but it is not checked, the job processor probably cannot find your ps1 file. Check if the file really exists and for spelling errors in the configuration.



If you want to test a new job, but you don't want to wait 10 min till the Jobprocessor begins to work, just click in the dialog file-> pause and then file->resume. Now all your queued jobs will start.

Environment Variables

PowerJobs sets three environment variables when it's installed. They contain the paths to the **jobs** and **modules** folder and the **powerJobs.dll**. PowerJobs loads the dll automatically, but if you want to debug your jobs you have to load it yourself.

```
[System.reflection.Assembly]::LoadForm($env:POWERJOBS_DLL)
```

The three environment variables are:

- POWERJOBS_DLL
- POWERJOBS_JOBSDIR
- POWERJOBS_MODULESDIR



To get a list of your environment variables type the following in powershell:

```
cd env:
dir
```

Code Reference

powerJobs

Contains the job and log object and other helpful functions.

Syntax




```
$powerJobs.Member
```

Members




powerJobs has these types of members:


- [Properties](#)
- [Methods](#)

Properties

	Type	Name	Description	Access type
	Context	Context	Simulates an IJobProcessorServices interface	read-only
	Job	Job	Simulates an IJob interface	read-only
	Log	Log	Grants access to powerJob's internal errorlogger.	read-only

Methods

	Name	Description
	getPublisher	Initializes a publisher.
	initialize	
	GetLatestItemByNumber	Gets the latest Revision for an Item.

	GetUniqueLatestFileByFilename	Gets the latest file paths for a set of files, by file name
---	-------------------------------	---

getPublisher

Initializes a publisher.

Syntax

```
$powerJobs.getPublisher( "PublishFormat" )
```

Parameters

publishFormat [input]

Type	Value	Description
String	PDF	Returns a PdfPublisher
String	DWF	Returns a DwfPublisher
String	DWFX	Returns a DwfxPublisher
String	SHEETDXF	Returns a DxfPublisher

Return value

Returns a [publisher object](#) depending on the input or NULL if no valid input is passed to the function.

Job

A Job from from the queue.

Syntax





```
$powerJobs.job.Member
```



Members

Job has these types of members:

- [Properties](#)

Properties

	Type	Name	Description	Access type
	String	Description	Gets the description of the job.	Read-write
	Long	Id	Gets the unique ID.	Read-write
	String	JobType	Gets the job type.	Read-write
	Dictionary< String , String >	Params	Gets the set of parameters on the job.	Read-only

	Int	Priority	Gets the priority of the job. Lower number means higher priority. 1 is the highest priority. Less than 1 is not allowed.	Read-only
	String	VaultName	Gets the Vault that the Job references.	Read-write

Log

Grants access to powerJob's internal errorlogger.

Syntax






```
$powerJobs.Log.Member
```

Members











Log has these types of members:

- [Properties](#)
- [Methods](#)

Properties

	Type	Name	Description	Access type
	Bool	IsDebugEnabled		Read-only
	Bool	IsErrorEnabled		Read-only
	Bool	IsInfoEnabled		Read-only
	Bool	IsWarnEnabled		Read-only
	Logger	Logger		Read-only

Methods

	Name	Description
	Debug	Logs a message with loglevel debug .
	DebugFormat	
	Error	Logs a message with loglevel error .
	ErrorFormat	
	Fatal	Logs a message with loglevel fatal .
	FatalFormat	
	Info	Logs a message with loglevel info .
	InfoFormat	
	Warn	Logs a message with loglevel warn .
	WarnFormat	

Examples

```
try{
    $files = $vaultConnection.FileManager.AcquireFiles($settings)
}
catch{
    $powerJobs.Log.Error("Failed to aquire files")
}
```

See also

For further information about Log4Net you can look at <http://logging.apache.org/log4net/>

Publisher

Contains the functionality to create PDF, DWF, DWFX and DXF files

Syntax





```
$powerJobs.getPublisher("PublishFormat")
```

Members

Publisher has these types of members:



- [Properties](#)
- [Methods](#)
- [Events](#)

Properties


	Type	Name	Description	Access type
	Bool	IgnoreCheckOutCheck	Currently not used	read-write
	String	OutputFile	Sets Filepath + Filename + Extension for the published file	read-write
	Int	GeneratorEnginePublish Count	Currently not used	read-write
	String	Options	This is part of the inventor-api. For a full description take a look at the inventor-api help at "HTML/DataIO_Sample.htm Translate - Sheet Metal to DXF API Sample "	read-write

Methods

	Name	Description
	Open	Opens a File without publishing it.
	Publish	Publishes a file.

	add_OnBeginPublish	Subscribes code to the OnBeginPublish Event.
	remove_OnBeginPublish	

Events

	Name	Description
	OnBeginPublish	This event gets raised right before the document gets published.

add_OnBeginPublish


Subscribes code to the OnBeginPublish Event.

Syntax

```
add_OnBeginPublish(  
{  
    param($publisher, $PublishEventArgs)  
    %do something with $document or $application...  
})
```

Parameters

Delegate [input]

 In powershell a delegate is declared by using the {} brackets without further keywords.

Type	Value	Description
CommonPublisher	\$publisher	Conains the current publisher object
PublishEventArgs	\$document, \$application	<p>\$document: The object passed to this variable depends on the files you are working with. See also Remarks.</p> <p>\$application: If you are working with Inventor files powerJobs will pass an Inventor object to this variable. You can access it with "\$document.application."</p>

Remarks

If the publisher is working with an Inventor file, the \$document variable will contain a reference to Inventor API Document instance of the current document. From here you can use the Inventor API o do all sorts of things.

If the publisher is working with TrueView, the \$docment variable will be a string containing the filename of the DSD file used to create the TrueView output. Here you can use the powershell text manipulation facilities to modify this DSD file to your needs.

OnBeginPublish

This event gets raised right before the document gets published.

Remarks

See [add_OnBeginPublish](#) for further information.

Open

Opens a File without publishing it.

Syntax

```
$publisher.Open(fileID)
```

Parameters

fileID [input]

Type	Value	Description
Long	fileID	The file id is part of the file object. Its property name is EntityIterationId .

Return value

Type	Value	Description
Bool	True	On success
Bool	False	On error

Remarks

You can add events like OnBeginPublish to this function.

Examples

```
$publisher.Open($file.EntityIterationId)
```

Publish

Publishes a file.

Syntax

```
$publisher.Open(fileID)
```

Parameters

fileID [input]

Type	Value	Description
Long	fileID	The file id is part of the file object. Its property name is EntityIterationId .

Return value

Type	Value	Description
------	-------	-------------

Bool	True	On success
Bool	False	On error

Vault

Syntax

```
$vault.Membername
```

Members

Vault has these types of members:

- [Properties](#)
- [Methods](#)



The members of the vault object are part of the vault api. You can look them up in the VaultVDF documentation at **Autodesk.Connectivity.WebServices Namespace**

Properties

	Type	Name	Description	Access type
		AdminService	Contains methods for manipulating users and groups.	read-only
		AuthService	Contains methods for authenticating to the Vault server.	read-only
	Bool	AutoTransferOwnership	Gets or sets the automatic ownership behavior on all the services in the WebServiceManager.	write-only
		BehaviorService	Contains methods for manipulating behaviors.	read-only
		CategoryService	Contains methods for manipulating categories.	read-only
		ChangeOrderService	Contains methods for creating and manipulating change orders.	read-only
		ContentService		read-only
		CustomEntityService	A collection of methods related to the Custom Entity entity type.	read-only
		DocumentService	Contains methods for manipulating files and folders within a vault.	read-only
		DocumentServiceExtensions	Contains more methods for manipulating files and folders within a vault.	read-only

		FilestoreService	Contains methods for uploading and downloading binary file data.	read-only
		FilestoreVaultService	Contains methods to determine information about the Knowledge Vaults.	read-only
		ForumService	Contains methods for posting messages.	read-only
		InformationService	Contains methods to determine information about the server, such as the version and product level.	read-only
		ItemService	Contains methods for manipulating items.	read-only
		JobService	Contains methods for manipulating the job queue.	read-only
		KnowledgeLibraryService		read-only
		KnowledgeVaultService	Contains methods for getting information about the vaults on the server.	read-only
		LifeCycleService	Contains methods related to the lifecycle behavior.	read-only
		PackageService	Contains methods for importing and exporting item data.	read-only
		PropertyService	Contains methods for manipulating properties on Entities.	read-only
		ReplicationService	Contains methods for transferring ownership between workgroups.	read-only
	Bool	ReSignIn	Gets or sets the re-sign in behavior on all the services in the WebServiceManager.	write-only
		RevisionService	Contains methods for manipulating revision values and schemes for Entities.	read-only
		SecurityService	Contains methods for logging into and out of vaults and setting security on specific Entities.	read-only
		SharePointService		read-only
		WebServiceCredentials		read-write
		WinAuthService	Contains methods for logging into and out of vaults using Windows credentials.	read-only

Methods

	Name	Description
	Clone	
	GetSyncedClone	

VaultConnection

Syntax

```
$VaultConnection.Membername
```

Members









VaultConnection has these types of members:






- [Properties](#)
- [Methods](#)





The members of the vault object are part of the vault api. You can look them up in the VaultVDF documentation at **Connection Class Members**


Properties

	Type	Name	Description	Access type
		CategoryManager	Gets an object which encapsulates all access to Vault Categories	read-only
		ChangeOrderManager	Gets an object which encapsulates all access to Vault Change Orders	read-only
		ConfigurationManager	Gets an object which manages configuration data on the vault server.	read-only
		CustomObjectManager	Gets an object which encapsulates all access to Vault Custom Objects	read-only
		EntityOperations	Gets an object which provides a set of workflows that can be applied to any entity object.	read-only
		FileManager	Gets an object which encapsulates all access to Vault Files	read-only
		FolderManager	Gets an object which encapsulates all access to Vault Folders	read-only
	String	IdentityKey	Gets a string which uniquely identifies this connection.	read-only

	Bool	IsAnonymousConnection	Gets a value which identifies whether or not we have an anonymous connection to the server.	read-only
	Bool	IsConnected	Gets a values which tells if this connection object has an active connection. If a logout occurs, then the connection will essentially be invalid.	read-only
	Bool	IsReadOnly	Gets a value which identifies whether or not this is a read only connection to the server	read-only
	Bool	IsServerOnlyConnection	Gets a value which identifies whether or not we have a connection to a server but not to a specific vault.	read-only
	Bool	IsWindowsAuthenticated Connection	Gets a value which identifies whether or not the connection to the server used windows authentication.	read-only
		ItemManager	Gets an object which encapsulates all access to Vault Items	read-only
		LinkManager	Gets an object which encapsulates all access to Links	read-only
	Long	PartSizeInBytes	Gets the password for the authenticated user.	read-write
		PersistableIdManager	Gets an object which encapsulates all interaction with persistable ids	read-only
		PropertyManager	Gets an object which encapsulates all access to vault property definitions and property values.	read-only
	String	Server	Gets the name of the server that we are connected to.	read-only
	String	Ticket	Gets an encrypted ticket that represents the unique connection to the server.	read-only
	Long	UserID	Gets the unique identity of the authenticated user on the vault server.	read-only
	String	UserName	Gets the name of the authenticated user.	read-only
	String	Vault	Gets the name of the vault that we are connected to.	read-only

		WebServiceManager	Gets the low level Web Service Manager which stores the underlying physical connection to the server. This can be used to make direct calls to the web service layer for any functionality that is not provided by the Framework.	read-only
		WorkingFoldersManager	Gets an object which encapsulates all access to a vaults working folders	read-only

Methods

	Name	Description
	ClearCache	Clears the specified cached data from the connection
	Equals	Tests if obj is equal to this connection object.

vaultExplorerUtil

Syntax


```
$vaultExplorerUtil.Membername
```

Members

vaultExplorerUtil has these types of members:

- [Methods](#)

Methods

	Name	Description
	UpdateFileProperties	Updates a set of properties for a file.

Code Snippets

Changing filestates, categories

Create a textfile via template

Create DWG from an IDW

Creating PDFs with their parents' properties

Export Vault Data to XML

General powershell codesnipptes

- [Convert office documents to Pdf](#) — Convert Word, Excel or Powerpoint files to PDF
- [Copy file in a directory](#) — Copy a file in a directory and create the directory if it does not exist
- [Print something with the windows default printer](#)
- [Send queries to a sql server](#) — Connecting and sending queries to a sql server via powershell
- [Set default printer](#) — Setting the windows default printer via powershell.

Print via Inventor

Retrive the user that queued the job

Selected files to ZIP

Send email via jobserver

Changing filestates, categories



If you want to change, filestates/categories with the jobserver the jobserver user needs sufficient permissions. Additionally you can only change things you could change manually as well.

Changing state of master files

```
Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
$file = PrepareEnvironmentForFile "Part2.ipt" $true
$powerJobs.Log.Info("Starting job 'Release_File' ...")
#Path where the .txt file and then the .xml file should be placed
$path = "C:\${$file.EntityName}"
#All LifeCycleDefinitions get hooked, you need them to set a filestate
$def = $vault.DocumentServiceExtensions.GetAllLifeCycleDefinitions()
#In this case we take the "Flexible Release Process"
$FlexibleReleaseProcess = $def | Where-Object {$_.DispName.Equals("Flexible Release Process")}
#From the "FlexibleReleaseProcess"-object you can take your favorite state to set
#in this case "Released"
$releaseState = $FlexibleReleaseProcess.StateArray | Where-Object {$_.Name.Equals("Released")}
#The masterfilestate gets changed to "Released"
$vault.DocumentServiceExtensions.UpdateFileLifeCycleStates(@($file.EntityMasterId),@($releaseState.Id),"Released from PS")
```

Changing state of child files

```

Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
$file = PrepareEnvironmentForFile "Assembly1.iam" $true
$powerJobs.Log.Info("Starting job 'change_state_child' ...")

#All LifecycleDefinitions get hooked, you need them to set a filestate
$def = $vault.DocumentServiceExtensions.GetAllLifecycleDefinitions()
#In this case we take the "Flexible Release Process"
$FlexibleReleaseProcess = $def | Where-Object {$_.DispName.Equals("Flexible Release Process")}
#From the "FlexibleReleaseProcess"-object you can take your favorite state to set
$releaseState = $FlexibleReleaseProcess.StateArray | Where-Object {$_.Name.Equals("Released")}

#Get all childfiles
$assocs = $vault.DocumentService.GetFileAssociationsByIds(@($file.EntityIterationId),
[Autodesk.Connectivity.WebServices.FileAssociationTypeEnum]::None,$false,
[Autodesk.Connectivity.WebServices.FileAssociationTypeEnum]::All,$true,$false,$false)
if($assocs[0].FileAssocs -ne $null){
    $stateIds = @()
    $childfileIds = @()
    foreach($f in $assocs[0].FileAssocs){
        $childfileIds += $f.CldFile.MasterId
        $stateIds += $releaseState.Id
    }
}
#The childfilesstates get changed to "Released"
$vault.DocumentServiceExtensions.UpdateFileLifecycleStates($childfileIds,$stateIds,"Released from PS")

```

Changing categories

```

Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
$file = PrepareEnvironmentForFile "Part2.ipt.pdf" $true
$powerJobs.Log.Info("Starting job 'change_category' ...")

#All category-definitions get hooked
$defc = $vault.CategoryService.GetCategoriesByEntityClassId("FILE",$true)
#In this case we want to set the filecategory to "Engineering"
$category = $defc | Where-Object {$_.Name.Equals("Engineering")}
#Change Category
$vault.DocumentServiceExtensions.UpdateFileCategories(@($file.EntityMasterId),$category.Id,"Category Change by PS")

```

Create a textfile via template

To use the script, create a file "C:\template.txt". Write your desired text and the variables for the properties in the file.

The syntax is as following:

```
Text text text text {Propertiename};
```

The format is flexible. You could also define a html page.


```
<html>
<table border="1">
  <tr>
    <td>{Name};</td>
    <td>{Classification};</td>
    <td>{Created By};</td>
  </tr>
</table>
</html>
```

```

Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"

$file = PrepareEnvironmentForFile "Assembly1.idw" $true

$powerJobs.Log.Info("Starting job 'Create_txt' ...")

#Paths:
#TEMPLATE-PATH
$pathTemp = "C:\template.txt"
$FilePath = "C:\file.txt"

#Propertydefinitions
$propDefs = $vault.PropertyService.GetPropertyDefinitionsByEntityClassId("FILE")

#Writes the Propertydefinitionids in a System.Array to use a specific method
$sids = @()
foreach($f in $propDefs){
    $sids+=$f.Id
}
#gets the file-properties
$props =
$vault.PropertyService.GetProperties($file.EntityClass.Id,@($file.EntityIterationId),$
ids)

#Define RegularExpression, to get the propertynames from the template-file
$regex = [regex]"\"{+(?i)\b[A-Z\s]+\b}\""
$template = Get-Content -Path $pathTemp
$header = $regex.Matches($template)

#Get the values
foreach($match in $header)
{
    $literalFieldName = $match.Value.TrimStart("{").TrimEnd("}")
    $propId = 0
    foreach($pd in $propDefs)
    {
        if($pd.DisplayName.Equals($literalFieldName))
        {
            $propId=$pd.Id
        }
    }
    if($propId -gt 0)
    {
        foreach($pi in $props)
        {
            if($pi.PropDefId.Equals($propId))
            {
                $template = $template -replace $match.Value, $pi.Val
            }
        }
    }
}

#Writes the values in a file
$template |Out-File $FilePath -Append

```

Create DWG from an IDW

This code converts an IDW into an DWG. At line 20 you can choose other filetypes as well. But keep in mind you can only convert what is supported by Inventors "Save as"

```
Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"

$file = PrepareEnvironmentForFile "Assembly1.idw" $true

$powerJobs.Log.Info("Starting job 'Create DWG' ...")

# limit publishing to idw files
$ext = [System.IO.Path]::GetExtension($file.EntityName).ToLower()
$inventorExtensions = @(".idw")
if($inventorExtensions -contains $ext)
{
    # publish (generate the dwfx attachment)
    $localDestFile = "C:\TEMP\" + $file.EntityName + ".dwg"
    $publisher=$powerJobs.GetPublisher("PDF")
    $publisher.OutputFile=$localDestFile
    #Eventhandler in which you can create other file formats
    $publisher.add_OnBeginPublish(
    {
        param($publisher, $document)
        $document.Document.SaveAs($localDestFile,$true)
    })

    #The Eventhandler gets called
    if(!$publisher.Open($File.EntityIterationId))
    {throw "The .dwg-translation failed!"}
}
```

Creating PDFs with their parents' properties

```
#region Get Properties
$EntityClassId = $file.EntityClass.get_Id()
$propDefs =
$vault.PropertyService.GetPropertyDefinitionsByEntityClassId($EntityClassId)
$UpdDefs = $propDefs | Where { $_.IsSys -eq $false }
$UpdIds = @()
$UpdDefs | ForEach-Object { $UpdIds += $_.Id }
$props = $vault.PropertyService.GetProperties($EntityClassId,
@($file.EntityIterationId), @($UpdIds))
$pIds= @()
$pVals= @()
$props| ForEach-Object { $pIds+=$_.PropDefId; $pVals+=$_.Val }
#endregion
#Checkout PDF
#...
#update the properties
$vault.DocumentService.UpdateFileProperties(@($newPdfFile.EntityMasterId), $pIds, $vals)
#Checkin PDF
#...
```

Export Vault Data to XML

With this code you can export data from Vault into a xml file.

```

Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"

$file = PrepareEnvironmentForFile "Assembly1.idw" $true

$powerJobs.Log.Info("Starting job 'Create_xml' ...")

#Path where the .txt file and then the .xml file should be placed
$path = "C:\${$file.EntityName}"

#Propertydefinitions
$propDefs =
$vault.PropertyService.GetPropertyDefinitionsByEntityClassId($file.EntityClass.Id)
#Writes the Propertydefinitionids in a System.Array to use a specific method
$sids = @()
foreach($f in $propDefs){
    $sids+=$f.id
}
#Gets the file-properties
$props = $vault.PropertyService.GetProperties("FILE",@($file.EntityIterationId),$sids)

$values = @()
$names = @()
#Gets the exact name and value of the properties
for($i = 0; $i -lt $propDefs.Count; $i+=1){
    for($j = 0; $j -lt $props.Count; $j+=1){
        if($propDefs[$i].Id -eq $props[$j].PropDefId -and $props[$j].ValTyp -ne
"Image"){
            $values += $props[$j].Val
            $names += $propDefs[$i].DispName
        }
    }
}

#Writes the properties first in a .txt-file
#"root"-tag at the beginning and the end of the xml-file is necessary
"<root>" | Out-File "$($path).txt"
for ($i = 0;$i -lt $props.Length-1; $i+=1){
    $xmltags = "<${$names[$i]}>${$values[$i]}</${$names[$i]}>"
    #The String have to be manipulated to make a xml-file, because characters as space
or parentheses are forbidden in tags
    $xmltags.Replace(" ", "").Replace("(","").Replace(")","") + "`n" | Out-File
"$($path).txt" -Append
}
"</root>" | Out-File "$($path).txt" -Append

#Writes the .txt-file content into the xml-file
Get-Content "$($path).txt" | Out-File "$($path).xml"
#Remove the txt-file
Remove-Item "$($path).txt"

```

General powershell codesnipptes

Convert office documents to Pdf

Convert Word, Excel or Powerpoint files to PDF

Copy file in a directory

Copy a file in a directory and create the directory if it does not exist

Print something with the windows default printer

Send queries to a sql server

Connecting and sending queries to a sql server via powershell

Set default printer

Setting the windows default printer via powershell.

Convert office documents to Pdf

Requirements

- MS Office2010 or higher

Word

```
$word = New-Object -ComObject Word.Application
Sleep -Seconds 10
$process = Get-Process winword -ErrorAction SilentlyContinue
$word.Visible = $false
$doc = "C:\Temp\Document.docx"
$saveaspath = "C:\Temp\Document.pdf"
#to fix language pack problems
$ci = [System.Globalization.CultureInfo]'en-US'
#Opens the wordfile to save as pdf-file
$openDoc = $word.documents.PSBase.GetType().InvokeMember('Open',
[Reflection.BindingFlags]::InvokeMethod, $null,$word.documents,$doc, $ci)
#Creates the pdf-file
$openDoc.ExportAsFixedFormat($saveaspath ,
[Microsoft.Office.Interop.Word.WdExportFormat]::wdExportFormatPDF)
$openDoc.Close()
$word.Quit()
```

Excel

```
#Creates a Excel-Object
$excel = New-Object -ComObject Excel.Application
$excel.Visible = $false
$formatPDF = 17
$saveaspath = "C:\TEMP\WorkBook.pdf"
$workbook = $excel.Workbooks.Open("C:\TEMP\WorkBook.xlsx")
#Creates the pdf-file
$workbook.SaveAs($saveaspath , $formatPDF)
$workbook.Close()
$excel.Quit()
```

Powerpoint

```
$powerpnt = New-Object -ComObject PowerPoint.Application
$doc = "C:\Temp\Presentation.pptx"
$saveaspath = "C:\Temp\Presentation.pdf"
$openDoc =
$powerpnt.Presentations.PSBase.GetType().InvokeMember('Open',[Reflection.BindingFlags]
::InvokeMethod,$null,$powerpnt.Presentations,$doc, $ci)
$openDoc.SaveAs($saveaspath ,
[Microsoft.Office.Interop.PowerPoint.PpSaveAsFileType]::ppSaveAsPDF,[Microsoft.Office.
Core.MsoTriState]::msoFalse)
$openDoc.Close()
```

Copy file in a directory

Copy a file in a directory and create the directory if it does not exist

```
#Path to file
$sourceFile = "C:\<YourFolder>\<YourFile>"

#Path to your directory, if it doesnt exist, it creates a new one
$targetDirectory = "C:\<YourFolder>"

#Test if path is existing
if(!(Test-Path $targetDirectory)){mkdir $targetDirectory}

#file get copyed to directory
Copy-Item -Path $sourceFile -Destination $targetDirectory

#The Copy-Item cmdlet contains a lot more intresting options, for instance -Force
#to force overwriting, or -Recurse to copy a complete folderstructure
```

Print something with the windows default printer

```
#Write here the file-path
$document = "C:\<YourFolder>\<YourFile>"

#Prints the document content and waits until the process ends
Start-Process -FilePath $document -Verb Print -Wait
```

Send queries to a sql server

Requirements

- You need a SQL Server named MySQLServer
- You should be able to connect to it via integrated security
- The server should have a Database called TestDB
- TestDB should contain a Table called Table1 with columns matching the insert statement below

```

#settings
$DataSource = 'MySQLServer'
$DbName = 'TestDB'

#create connection object
$conn = New-Object System.Data.SqlClient.SqlConnection("Data Source=$DataSource;
Initial Catalog=$DbName; Integrated Security=SSPI")
#open database connection
$conn.Open()

#get a command object
$cmd = $conn.CreateCommand()

#define the insert statement
$cmd.CommandText = "INSERT INTO Table1 VALUES ('testtext1', 'testtext2', 123)"

#execute the command
$cmd.ExecuteNonQuery()

#cleanup
$cmd.Dispose()

#close the connection
$conn.Close()

#cleanup
$conn.Dispose()

```

Set default printer

Setting the windows default printer via powershell.

```

#get default printer
$olddefaultprinter=Get-WmiObject -Class Win32_Printer -Filter "Default=True"

#set new default printer
$newdefaultprinter=Get-WmiObject -Class Win32_Printer -Filter "DeviceID='Prntername'"
$newdefaultprinter.SetDefaultPrinter()

#write here the actions to be done with new default printer

#set old default printer, if needed
$olddefaultprinter.SetDefaultPrinter()

```

Print via Inventor


```

Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
$file = PrepareEnvironmentForFile "Catch Assembly.idw" $true
$powerJobs.Log.Info("Starting job 'PrintWithInventor' ...")

#region Settings
# limit publishing to 2d inventor files
$validExtensions = @(".idw")
#endregion

$ext = [System.IO.Path]::GetExtension($file.get_EntityName()).ToLower()
if($validExtensions -contains $ext)
{
    $publisher=$powerJobs.GetPublisher("PDF")
    $publisher.OutputFile="c:\temp\dummy.pdf"
    $publisher.add_OnBeginPublish(
    {
        param($publisher, $PublishEventArgs)
        $printManager = $PublishEventArgs.Document.PrintManager

        $printManager.GetType().InvokeMember("Printer",[Reflection.BindingFlags]::SetProperty,
        $null, $printManager, "PDF Report Writer")
        $printManager.ScaleMode = [Inventor.PrintScaleModeEnum]::kPrintBestFitScale
        $printManager.PrintRange = [Inventor.PrintRangeEnum]::kPrintAllSheets
        $printManager.PaperSize = [Inventor.PaperSizeEnum]::kPaperSizeA0
        $printManager.SubmitPrint()
    })
    if (!$publisher.Open($file.EntityIterationId))
    {
        throw ("Open failed")
    }
}

```

Retrive the user that queued the job

```

#retrieve all jobs in the Vault jobqueue
$jobs = $vault.JobService.GetJobsByDate(1000,[DateTime]::MinValue)
#select the job I am (me)
$currentJob = $jobs | Where-Object {$_.Id.Equals($powerJobs.Job.Id)}
#Gets user informations from the Job-creator
$userinfo = $vault.AdminService.GetUserByUserId($currentJob.CreateUserId)
#get the email address of the user that queued the job
$email = $userinfo.Email
#place here you additional code
#For instance, send an E-Mail to the user
#The following code writes the userdata in a textfile.
$userinfo >> 'C:\usertest.txt'

```

Selected files to ZIP

Select files in Vault and let them zip together via jobserver. The resulting zip file could be sent via email, stored into Vault, saved in a custom folder, uploaded somewhere, etc

```
#To test this script, just copy&paste the content above into a new powershell file,
for instance called FilesToZip.ps1, and save it into the powerJobs folder.
#Edit the JobProcessor.exe.config to declare your new job. For queueing the job, you
might use the LifecycleEventEditor and configure your job on a given lifecycle change.

Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
$file = PrepareEnvironmentForFile "Assembly1.idw" $true
$powerJobs.Log.Info("Starting job 'Zip_Structure' ...")

#Path for the final zipfile
$zipout = "C:\myZips\"
#region DownloadSettings
#Generate a Settingsobject. It is necessary for the new AcquireFiles method
$settings = New-Object
Autodesk.DataManagement.Client.Framework.Vault.Settings.AcquireFilesSettings($vaultcon
nection, $false)
$settings.AddEntityToAcquire($file)
#Temporary path for the downloaded files
$settings.set_LocalPath("C:\TEMP\zip")
#"Checkout", "Download", "NoAction"are possible values
$settings.set_DefaultAcquisitionOption("Download")
#What should be included in the zip?
$settings.OptionsRelationshipGathering.FileRelationshipSettings.set_IncludeChildren("T
RUE")
$settings.OptionsRelationshipGathering.FileRelationshipSettings.set_RecurseChildren("T
RUE")
$settings.OptionsRelationshipGathering.FileRelationshipSettings.set_IncludeAttachments
("TRUE")
#endregion
#Downloading the files
$files = $vaultConnection.FileManager.AcquireFiles($settings)

#It creates a new folder even though it exist
New-Item -Path $settings.get_LocalPath() -Force -ItemType Directory
New-Item -Path $zipout -Force -ItemType Directory

#Create Zip-File
$zipFileName = "$($zipout)$($file.EntityName).zip"
set-content $zipFileName ("PK" + [char]5 + [char]6 + ("$([char]0)" * 18))
$ZipFile = (new-object -com shell.application).NameSpace($zipFileName)
$fileToBeZipped = Get-ChildItem $settings.get_LocalPath()

$fileToBeZipped | ForEach-Object {
    #The method .MoveHere() is running asynchronous, so we have to wait a few seconds
before moving the next file into zip
    #In this case: 2 seconds
    Start-Sleep -Seconds 2
    #Write files into the zipfile
    $ZipFile.MoveHere($_.FullName,1024)
}
```

Send email via jobserver

```

Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
$file = PrepareEnvironmentForFile "Part2.ipt" $true
$powerJobs.Log.Info("Starting job 'send_email' ...")

#region Config Mail
    $from = "powerjobs@yourdomain.com" #Required
    $to = "user@targetdomain.com" #Required
    $subject = "The document $($file.EntityName) has been processed" #Required
    $body = "Write here your email text and use variables to add additional
information" #Optional
#endregion
#region Config SMTP
    $smtp = "yourSMTPServer"
    $passwd = ConvertTo-SecureString -AsPlainText "YourPassword" -Force
    $cred = new-object Management.Automation.PSCredential $from, $passwd
#endregion
#region Config Attachment
    $sendattachment = $true
#endregion
#region Send Message
    if($sendattachment -eq $true){
        Send-MailMessage -To $to -From $from -Subject $subject -Body $body -SmtpServer
$smtp -Credential $cred -Attachments "C:\TEMP\$($file.EntityName)"
    }
    else{
        Send-MailMessage -To $to -From $from -Subject $subject -Body $body -SmtpServer
$smtp -Credential $cred
    }
#endregion

```

Get address from a vault property

In case you like to send the email to persons in a more dynamic way, and let's suppose that you have the email address of the person stored in a user defined property, here are some more rows that might help you to retrieve the email address.

So, the first line pulls all the property definitions from Vault. The second filters all the property definitions for the one property you are looking for, so change the "SendTo" string to the name of the property you are looking for. The third line pulls the value for our file for the specified property. And finally the last line just stores the value from the first property into the variable \$emailTo, which you can now use with your Send-MailMessage command.

```

$propDefs =
$vault.PropertyService.GetPropertyDefinitionsByEntityClassId($file.EntityClass.Id)
$propDef = $propDefs | Where-Object { $_.DispName -eq "SendTo" }
$prop =
$vault.PropertyService.GetProperties($file.EntityClass.Id,@($file.EntityIterationId),@
($propDef.Id))
$emailTo = $prop[0].Val

```

Send notification via email

```

Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
$file = PrepareEnvironmentForFile "Catch Assembly.idw" $true
$powerJobs.Log.Info("Starting job 'email' ...")

#region config link
$vaultname = "Vault"
$vaultserver = "Servername"
$folders = $vault.DocumentService.GetFoldersByFileMasterId($file.get_EntityMasterId())
$folder = $folders[0]
$folderpath = $folder.FullName
$folderpath = $folderpath.Replace("$", "%24").Replace("/", "%2f")
$fullPath = $folderpath + "%2f" + $file.EntityName.Replace(" ", "+")
$link =
"http://$( $vaultserver )/AutodeskDM/Services/EntityDataCommandRequest.aspx?Vault=$( $vaultname )&ObjectId="+$fullPath+"&ObjectType=File&Command=Select"
#endregion
#region config lifecycle
$lfcTransId = $powerJobs.Job.Params["LifeCycleTransitionId"]
$lfcTrans =
$vault.DocumentServiceExtensions.GetLifeCycleStateTransitionsByIds(@($lfcTransId))
$lfcS =
$vault.DocumentServiceExtensions.GetLifeCycleStatesByIds(@($lfcTrans[0].FromId,
$lfcTrans[0].ToId))
$oldstate = $lfcS[0].DispName
$newstate = $lfcS[1].DispName
#endregion
#region Config Mail
    $from = "jobserver@yourcompany.com"
    $to = "receiver@yourcompany.com"
    $receivername = "receiver"
    $subject = "The file $($file.get_EntityName()) has changed from $oldState to
$newState"
    $body = "Dear $($receivername), if you like to view the related document, just
follow this link: $link"
#endregion
#region Config SMTP
    $smtp = "yourSMTPServer"
    $passwd = ConvertTo-SecureString -AsPlainText "YourPassword" -Force
    $cred = new-object Management.Automation.PSCredential $from, $passwd
#endregion
#region Send Message
    Send-MailMessage -From $from -To $to -Subject $subject -Body $body -SmtpServer
$smtp
#endregion

```

Tutorials

- [Adding a watermark to PDFs](#)
- [Adding the file revision to the PDF name](#)
- [Converting PDF to PDF/A](#)
- [Convert PDF to DWF](#)
- [PDF on Item Lifecycle Transition](#)
- [How to handle differently sized AutoCAD DWGs](#)

Adding the file revision to the PDF name

The following line saves the **Revisionlabel** in the variable **\$revision**. The revisionlabel is what you see in Vault. E.g. "A" or "C"

```
$revision = $file.RevisionInfo.get_RevisionLabel()
```

Afterwards you can add this variable to your filename string like this

```
$PDFfileName = $file.EntityName + "$($revision).pdf"
```



The outcome would be something like this:

4711.idwA.pdf or just **4711.idw.pdf** if you don't have a revision.



Of course you can further customize the filename of your PDFs. Powershell brings powerful tools for string modification. <http://technet.microsoft.com/en-us/library/ee692804.aspx>

Converting PDF to PDF/A

- First things first
 - [Check the script](#)
 - [Calling the function](#)
- [Automated Conversion for existing PDFs](#)
- [Modify the default PDF job to create PDF/A](#)
- [Validation](#)

First things first

In order to use the below module you need "**Ghostscript**". You can download it on this site: [DOWNLOAD](#)



You must use the 64 bit version of ghostscript! There may be issues with the 32 bit version, because the conversion process needs a lot of memory.



You can download our sample PDF module [HERE](#)

Check the script

Now make sure the install location in the script is correct.

```
function Format-PdfToPdfalb($original,$converted)
{
    $convertedparam = "-sOutputFile=" + $converted
    $oldloc = Get-Location
    #Ghostscript's Install Path
    cd "C:\Program Files\gs\gs9.05\bin\"
    .\gswin64c.exe -sDEVICE=pdfwrite -q -dNOPAUSE -dBATCH -dNOSAFAFER -dPDFA -dUseCIEColor
    -sProcessColor=DeviceCMYK $convertedparam $original
    cd $oldloc.Path
}
```

Calling the function

You can call the function like this:

```
$org = "C:\TEMP\original.pdf" $conv = "C:\TEMP\conv.pdf"
Format-PdfToPdfalb $org $conv
```



More Information about the ghostscript syntax can be found in its documentation. [Ghostscript_Documentation](#)

Automated Conversion for existing PDFs

```
Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
Import-Module "$env:POWERJOBS_MODULESDIR\cO_PDF-Helper.psm1"

# creating the $vault,$vaultConnection,$powerJobs and $file object for the Debugger
and for the PowerJobs environment
$file = PrepareEnvironmentForFile "Catch Assembly.iam.pdf" $true

$powerJobs.Log.Info("Starting job 'Pdf Conversion' ...")

#region PDF Settings
    $ErrorActionPreference = "Stop"
    $origpath = "C:\TEMP\orig\"
    $destpath = "C:\TEMP\PDF_A\"
    $PdfFileName = $file.EntityName
    $origfile = $origpath + $PdfFileName
    $destfile = $destpath + $PdfFileName
    $ValidExtensions = @(".pdf")
    #If true then newer PDFs will be overwritten
    $OverrideNewerPDF = $true
#endregion

#region DownloadSettings
    #Generate a Settingsobject. It is necessary for the new AcquireFiles method
    $settings = New-Object
Autodesk.DataManagement.Client.Framework.Vault.Settings.AcquireFilesSettings($vaultcon
nection, $false)
    $settings.AddEntityToAcquire($file)
    #Temporary path for the downloaded files
    $settings.set_LocalPath($origpath)
    #"Checkout", "Download", "NoAction"are possible values
    $settings.set_DefaultAcquisitionOption("Download")
    #What should be included?

$settings.OptionsRelationshipGathering.FileRelationshipSettings.set_IncludeChildren("F
ALSE")

$settings.OptionsRelationshipGathering.FileRelationshipSettings.set_RecurseChildren("F
ALSE")

$settings.OptionsRelationshipGathering.FileRelationshipSettings.set_IncludeAttachments
("FALSE")
#endregion

#region create folders
#It creates a new folder even though it exist
if( !(Test-Path -Pathtype Container -Path $origpath) ){
    New-Item -Path $origpath -ItemType Directory
```

```

}
if( !(Test-Path -PathType Container -Path $destpath) ){
    New-Item -Path $destpath -ItemType Directory
}
#endregion

try{
    #region Downloading the file
    try{
        $files = $vaultConnection.FileManager.AcquireFiles($settings)
    }
    catch{
        $powerJobs.Log.Error("Failed to aquire files")
    }
    if(!(Test-Path -PathType Leaf $origfile)){
        $powerJobs.Log.Error("$origfile is missing")
    }
    if((Test-Path -PathType Leaf $destfile)){
        #Creating a .Net file object
        $oDestFile = Get-ChildItem -Path $destpath | Where-Object { $_.Name -eq $PdfFileName }
        if(!$OverrideNewerPDF){
            if($oDestFile.LastWriteTime -gt $file.get_CheckInDate()){
                $powerJobs.Log.Error("Destfile: $($destfile) is newer than
Source: $($origfile)")
                exit 1
            }
        }
    }
}
#endregion

#region PDF conversation
#limit conversation to PDF files
$ext = [System.IO.Path]::GetExtension($file.EntityName).ToLower()
if( $ValidExtensions -contains $ext ){
    try{
        #Converting PDF
        $gserror = Format-PdfToPdfalb $origfile $destfile
        #Do something with the new PDF
    }
    catch{
        $powerJobs.Log.Error("PDF conversion failed: Ghostscript error:
$( $gserror )")
    }
    finally{
        #Uncomment the next line to delete $destfile. It`s commended out for
testing purposes
        #Remove-Item -Force $destfile
    }
}
else{
    $powerJobs.Log.Error("$( $ext ) is not a valid filetype")
}
#endregion
}

```

```
finally{
    Remove-Item -Force $origfile
}
```

Modify the default PDF job to create PDF/A

Overall you have to modify 5 lines.

The first thing is to import our PDF-Helper module.

```
...
Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
Import-Module "$env:POWERJOBS_MODULESDIR\cO_PDF-Helper.psm1"
...
```

A bit further down you need to add a new Variable **\$localOrigFile** and assign its value to **\$publisher.OutputFile**.

Then you have to add the **Format-PdfToPdfa1b** function directly after the PDF generation.

```
...
# publish (generate the pdf attachment)
$localOrigFile = "C:\TEMP\Orig" + $PDFfileName
$localDestFile = "C:\TEMP\" + $PDFfileName
$publisher=$powerJobs.GetPublisher("PDF")
$publisher.OutputFile = $localOrigFile
$checkInAsHidden = !$showPDF
$powerJobs.Log.Info("Creating Pdf ...");
if ($publisher.Publish($file.EntityIterationId) -eq $true)
{
    Format-PdfToPdfa1b $localOrigFile $localDestFile
    $folder=$vaultConnection.EntityOperations.GetParent($file)
}
...
```

The last thing to add is **Remove-Item \$localOrigFile** to delete the additional temporary file.

```
...
# delete the physical file
Remove-Item $localOrigFile
Remove-Item $localDestFile
...
```

Validation



There is no warranty that the Conversion will work always!

We tested the function and it worked with every pdf file we used, but it still may not work with special pdfs. Use one of the following PDF validators, to ensure that the PDF file conforms the PDF/a 1-b ISO standard.

<http://www.pdf-tools.com/pdf/validate-pdf-a-online.aspx>

<http://www.validatepdfa.com/online.htm>

Convert PDF to DWF

There is a tutorial about this on our blog written by [Martin Weiss](#).

<http://blog.coolorange.com/2013/08/23/convert-pdf-to-dwf-in-net/>

PDF on Item Lifecycle Transition

The standard PDF job that comes with powerJobs is prepared to create a PDF on a file lifecycle transition. When you perform a lifecycle transition in Vault, and you did configured Vault to queue a job, some default arguments will be passed to the job. One of the arguments is the File-Id. But if you configure Vault to queue a job on an item lifecycle transition, then you will get the ItemId as a parameter. As several files might be linked to your item, it's up to you to identify for which file attached to the item you like to create a PDF

Goal

After completing this tutorial you will know how to get the list of linked files to an item, how to find the file you are looking for and finally how to create a PDF out of it. It also shows you how to identify useful Vault API commands.

First Step

The first step ist to identify a convinient API call in order to get the files linked to an item. This blog post (<http://blog.coolorange.com/2013/03/09/vault-webservice-trace/>) describes how to activate the web service trace and how to filter

Second Step

Now we know that the function for getting the files associated to an item is called GetItemFileAssociationsByItemIds. Unfortunately the drawings associated to an item, could be of primary link if there is no model associated, or tertiary link if there is a model associated. Additoinal you may have a DWG and IDW associated to the same item. So, the logic to pick the rigt drawing might have to be adapted to your need. The code in the following lines will look for associated DWG or IDW. In case there only one hit, then we will take that one. In case there are multiple hits, then we will see if one is primary, otherwise we will just take the first. You have to insert it in the CreatePdfAsAttachment job on line two.

```

#Region Get File
$itemID = $powerJobs.Job.Params["ItemId"]
#$itemID = 1787
if($itemID)
{
    #collect all associations
    $fileItemAssoc =
    $vault.ItemService.GetItemFileAssociationsByItemIds(@($itemID),[Autodesk.Connectivity.
    WebServices.ItemFileLnkTypOpt]::Primary -bor
    [Autodesk.Connectivity.WebServices.ItemFileLnkTypOpt]::Secondary -bor
    [Autodesk.Connectivity.WebServices.ItemFileLnkTypOpt]::Tertiary)
    #search for associations of type IDW or DWG
    $drawings = $fileItemAssoc | Where-Object { $_.FileName -like '*.idw' -or
    $_.FileName -like '*.dwg' }
    if($drawings -is [System.Array]) #test if the result is an array or a single
    results
    {
        #if there are multiple results, lie a IDW or DWG, pick the one that is primary
        $primary = $drawings | Where-Object { $_.Type -eq 'Primary' }
        if($primary -ne $null) #if there is no primary, then just take the first one
        {
            $fileId = $drawings[0].CldFileId
        }
    }
    else
    {
        $fileId = $drawings.CldFileId
    }
    if(!$fileId) #if no file has been found, then quit the job with an entry in the
    log file
    {
        $item = $vault.ItemService.GetItemsByIds(@($itemID));
        $powerJobs.Log.Error("No drawing found for item "+$item.SyncRoot[0].ItemNum)
        return 1;
    }
    $powerJobs.Job.Params.Add("FileId","$($fileId)")
}
#Endregion

```

How to handle differently sized AutoCAD DWGs

TLDR

You define some templates and the script chooses the correct template based on a vault property.

Overview

Settings like **Paper size**, **Drawing Orientation** and **Plot area** are defined in powerJobs' own *TrueViewSetup.dwg*. The file is located in *C:\ProgramData\Autodesk\Vault 2014\Extensions\coolOrange.PowerJobs.Handler\TrueViewSetup.dwg* and the path cannot be changed. powerJobs only allows for one such a file and this file only has room for one paper size and one orientation.

The idea is to create a template folder with copies of the default *TrueViewSetup.dwg*. You name them something like *ISO_A4_land.dwg* or *ISO_A3_port.dwg*. Then you need a new vault property in which the filename is written for every file.

The template folder

My template folder is located at:

C:\ProgramData\Autodesk\Vault 2014\Extensions\coolOrange.PowerJobs.Handler\Templates

If you wish to use another folder you have to change it in the scripts.

It contains three templates. *Default.dwg*, *DIN_A4.dwg*, *DIN_A0.dwg*

Also I added a new property to my Vault called "Paper Size". In this property I wrote which template should be used by powerJobs. **DIN_A4** or **DIN_A0**.

Calling the script

Directly after the creation of my fileobject I call the Get-PropertyValue function. This function returns the exact value of the property of the file you passed to it.

```
$file = PrepareEnvironmentForFile "DEM001.dwg" $true
$papersize = Get-PropertyValue 'Paper Size' $file
Change-TemplateTo $papersize
```

Get-PropertyValue

```
function Get-PropertyValue(){
param(
[string]$propertyName,
[psCustomObject]$file
)
    $propDef =
$vault.PropertyService.GetPropertyDefinitionsByEntityClassId($file.EntityClass.Id) |
Where { $_.DispName -eq $propertyName }
    $property = $vault.PropertyService.GetProperties( $file.EntityClass.Id,
@($file.EntityIterationId), @($propDef.Id) )
    if($property -eq $null){
        return $null
    }
    else{
        return $property[0].Val
    }
}
```

The Script

The script uses the template name you passed to it to choose the correct file.

The script is rather simple so I won't go into detail, but you should pay some attention to line 16. There is decided what happens if there is no template. In this version of the script a warning is written to the logfiles and nothing else happens. You may want to [cancel](#) the whole job or use the default.dwg as template.

Change-TemplateTo

```
function Change-TemplateTo(){
param(
[String]$newTemplate
)
$path = [system.IO.Path]::GetDirectoryName($env:POWERJOBS_DLL)
$setupDwg = $path + '/' + 'TrueViewSetup.dwg'
$template = $path + '/Templates/' + $newTemplate + '.dwg'

if([System.IO.File]::Exists($template)){
    if([System.IO.File]::Exists($setupDwg)){
        cp -Path $setupDwg -Destination "$($setupDwg).bak"
        rm -Path $setupDwg -Force
    }
    cp -Path $template -Destination $setupDwg
}
else{
    $powerJobs.Log.Warn("The file $template doesn't exist")
}
}
```

FAQ

Where is the documentation for the Vault API?

Question

Where is the documentation for the Vault API?

Answer

If you have installed the Autodesk Vault 2014, you can find the documentation local under this path "**C:\Program Files (x86)\Autodesk\Autodesk Vault 2014 SDK\docs**". For all previous versions you have perhaps only to change the version of Vault in the path.

You can also visit the [Autodesk Developer Network](#). There you will find a developer community and some tutorials for the Vault API.

Troubleshooting

- [Logging Levels](#)
- [DLL not found in Powergui](#)
- [Jobprocessor freezes](#)
- [Resetting the toolbars](#)
- [Cannot Check in Inventor files if a Pdf is attached to the file](#)

Logging Levels

Log4Net

In the file 'coolOrange.powerJobs.dll.log4net' you can configure powerJob's errorlogging. In the standardinstallation you can find the file

coolOrange.powerJobs.dll.log4net

at

C:\ProgramData\Autodesk\Vault 2014\Extensions\coolOrange.PowerJobs.Handler\

In the line

```
<level value="ERROR" />
```

you can configure the logginglevel.

In the line

```
<param name="File" value="C:\temp\powerJobs.log" />
```

you can configure the outputpath and name of the logfile.



For further information about Log4Net you can look at <http://logging.apache.org/log4net/>

Logginglevel

ALL	Everything is written to the logfile
DEBUG	Debuginformation is written to the logfile
INFO	Every error, warnings and infos are written to the logfile
WARN	Every error and warnings are written to the logfile
ERROR	Every error is written to the logfile
FATAL	Only critical errors are written to the logfile
OFF	No logging

Errorlog Paths

The standardpath for the errorlogs is C:\TEMP

The latest log is called 'powerjobs.log', the older ones 'powerjobs.log1', 'powerjobs.log2' etc.

DLL not found in Powergui

Problem

You get an error, that some ddls cannot be found while debugging in Powergui

Solution

Make sure, that you are using the right version of Powergui.

- For powerJobs2013 the 32bit version of Powergui is required
- For powerJobs2014 the 64bit version of Powergui is required

Jobprocessor freezes

Problem

Your jobserver hangs if you try to make PDFs or stops with an error

Solution

Make sure that you started DWGTrueView and Inventor at least once with adminprivileges.

Restart your PC afterwards.

If you start the jobprocessor with adminprivileges it can cause errors as well.

Related articles

Error rendering macro 'contentbylabel' : com.atlassian.confluence.macro.params.ParameterException: 'PRD' is not an existing space's key

Resetting the toolbars

▼ Symptoms

The powerJobs button doesn't show up in the menu.

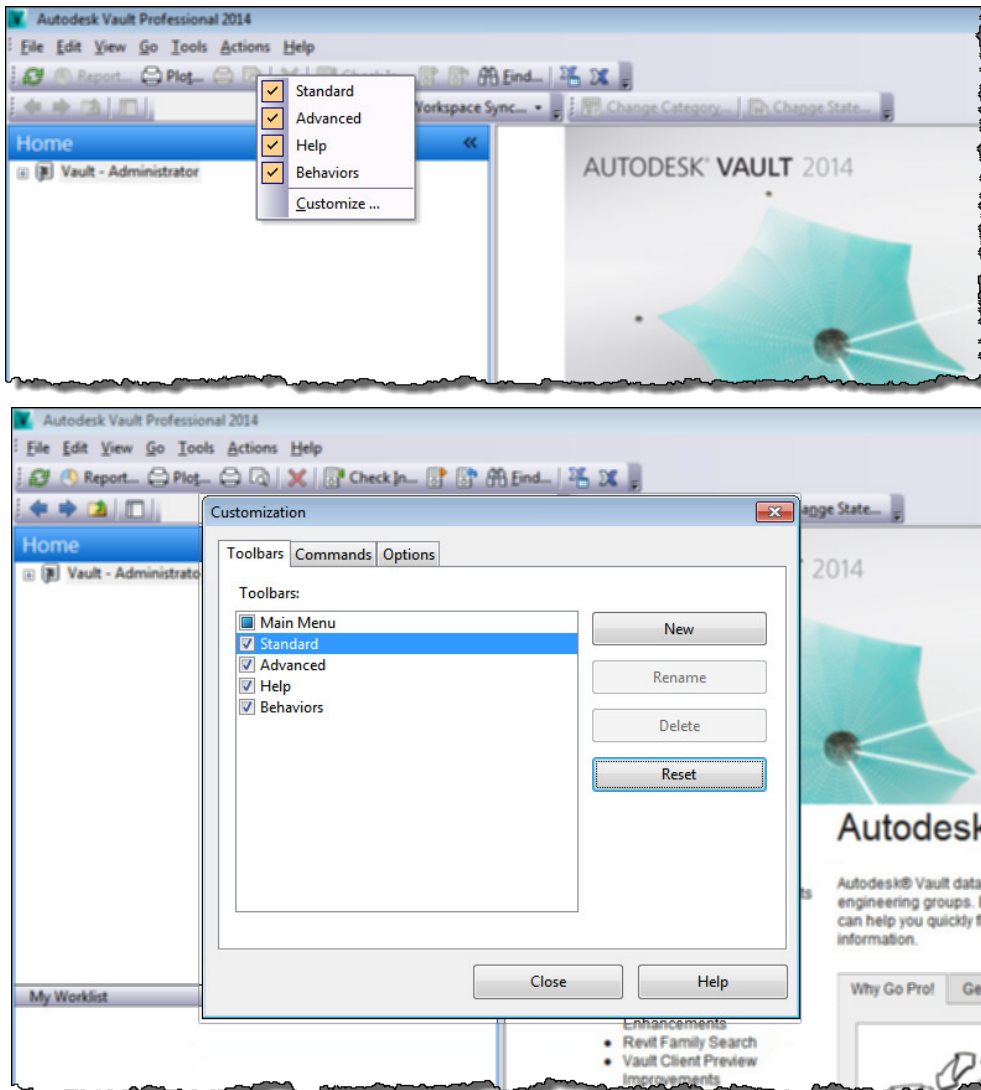
▼ Cause

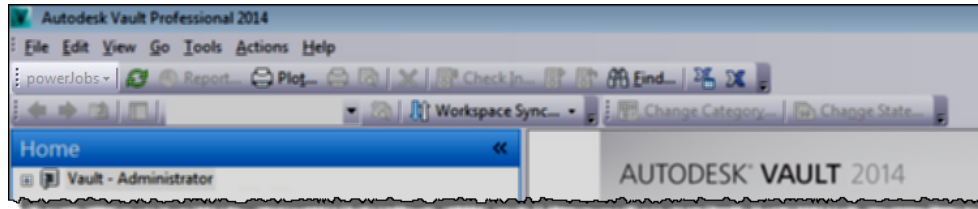
After a new installation of powerJobs the menu extension isn't loaded yet. Also sometimes the powerJobs button is disabled.

▼ Resolution

After the installation

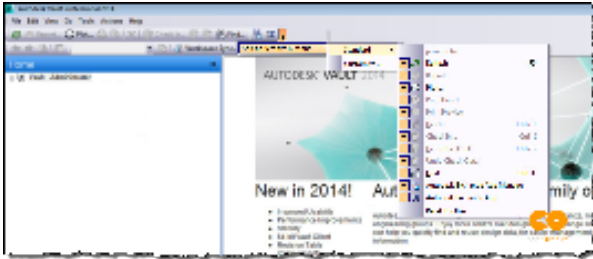
- Right click anywhere in the menu and click on "**Customize ...**"
- Select "**Standard**" and click on "**Reset**"





Button disabled

In case your button just disappeared then this solution should be preferred, because it doesn't reset your other customizations. Click on the small **arrow** to the **right** of the **standard toolbar**. Click on powerJobs to reactivate it.



More information

More information about the vault toolbars can be found in Autodesk's [knowledge base](#).

Cannot Check in Inventor files if a Pdf is attached to the file

Symptoms

Check in through Inventor returns an error if Dwf/Dwfx files are created during the check in

Cause

There is a bug in the Inventor API that crashes the check in if you want to generate a dwf/dwfx for a file, that already has a visualization attachment. Due to technical restrictions it isn't possible to change the classification of the pdfs. That's why it will crash on every file, that has a Pdf attached to it.

Resolution

This issue can be resolved through deactivating the Dwf/Dwfx generation during checkin in Inventor. Instead you can create them through the job processor.

Change logs

Current Version of powerJobs 2014 is 14.0.140

Date	Version	Description
24.09.2013	14.0.136	fixed bug multisheet for pdf creation of inventor drawings
	14.0.130	

What's new in powerJobs2014

What's new

- You will notice that we overall improved the look&feel of all products, and so the one of powerJobs, at least for those components that are visible, like the icon on your desktop.
 - In order to simplify the development of PowerShell scripts in relation to Vault, we introduced a function called *PrepareEnvironment*, resident in the *coolOrange.powerJobs.VaultHelper.psm1* module, which creates all the variables needed for the connection to Vault, like the \$vault, \$... and other. So, just by calling this function you are ready to write your code using those variables.
 - In this module you can find new functionality to create a.e. generic objects for working with the Vault API
 - As the PDF generation is the most used script, we further simplified, so in the *Settings region* you'll find the file extensions for which a PDF shall be generated.
 - We introduced new variables like the \$powerJobs, \$vaultConnection, \$vaultExplorerUtil in order to support the API changes in Vault 2014
 - You can use \$powerJobs.Log to have direct access to the powerJobs log4net logger
 - New Jobs has to be configured in the ..\Extensions\coolOrange.PowerJobs.Handler\PowerJobs.vcet.config file
 - PowerJobs is always enabling on first startup the JobQueue in your Vault and adding the PDF-sample Job to the lifecycle-state changed event of released states. Now you can choose if you want to add the PDF jobs to the lifecycle-state changed event. Its not obligatory any more.
-

What's changed

- In order to streamline the syntax across all PowerShell supported coolOrange products, we now expose the Autodesk WebserviceManager through the \$vault variable. So, the names for the webservices are the same as in the Autodesk API and identical to myView. This allows to easily use PowerShell code across the coolOrange products
- The special functions, like *GetUniqueLatestFileByFilename* which were part of the \$vault variable are now moved to a \$powerJobs. So the \$vault now is just the WebserviceManager, while the powerJobs specific members are now in the \$powerJobs variable.
- As mentioned, we introduced few more variable to support the new VDF, such as the \$vaultConnection, which represents the Vault connection object and the \$vaultExplorerUtil, which exposes the Explorer Utility objects. This way the full Vault API stays at your service from within powerJobs via PowerShell
- We removed the *downloadDependentFiles* function as such function is already provided by the VDF now. The *AcquireFiles* from the VDF is much more capable and flexible, so there is no point for providing our own function.