

1. Installation	3
2. Getting started	4
3. Activation and Trial limitations	5
4. Features	6
4.1 Executing jobs from the client	7
4.2 Executing jobs via lifecycle changes	7
4.3 Extended Jobprocessor UI	8
4.4 Variable inspector	9
4.5 powerVault Cmdlets	9
4.6 File format conversion	9
4.7 Integration to powerGate	10
4.8 Time triggered jobs	11
4.9 AutoCAD Engine	13
4.10 The sample jobs	15
4.11 Trace Window	16
4.12 Jobs with multilingual Vault support	17
5. Configuration	18
5.1 Configure Pdf creation for AutoCAD Dwgs	18
6. Customization	22
6.1 Code Reference	22
6.1.1 Cmdlets	22
6.1.1.1 Add-Log	23
6.1.1.2 Add-VaultFile	24
6.1.1.3 Get-VaultFile	25
6.1.1.4 Get-VaultFileAssociations	26
6.1.1.5 Get-VaultFiles	27
6.1.1.6 Open-VaultConnection	27
6.1.1.7 Save-FileAs	28
6.1.1.8 Show-Inspector	29
6.1.1.9 Update-VaultFile	29
6.1.2 powerJobs	31
6.1.2.1 getPublisher	32
6.1.2.2 Job	32
6.1.2.3 Log	33
6.1.3 Publisher	34
6.1.3.1 add_OnBeginPublish	35
6.1.3.2 OnBeginPublish	36
6.1.3.3 Open	36
6.1.3.4 Publish	37
6.1.4 Vault	37
6.1.5 VaultConnection	39
6.1.6 vaultExplorerUtil	41
6.2 Code Snippets	42
6.2.1 Changing filestates, categories, properties	42
6.2.2 Create a textfile via template	43
6.2.3 Create DWG from an IDW	44
6.2.4 Creating PDFs with their parents' properties	44
6.2.5 Export Vault Data to XML	45
6.2.6 General powershell codesnipptes	45
6.2.6.1 Convert office documents to Pdf	46
6.2.6.2 Copy file in a directory	47
6.2.6.3 Print something with the windows default printer	47
6.2.6.4 Send queries to a sql server	47
6.2.6.5 Set default printer	48
6.2.7 Get all users of a certain group	48
6.2.8 Print via Inventor	49
6.2.9 Retrive the user that queued the job	49
6.2.10 Selected files to ZIP	50
6.2.11 Send email via jobserver	50
6.3 Preparing your environment	52
6.4 IDEs for powershell	52
6.4.1 Adding a new IDE	55
6.5 Powershell, scripts and modules	57
6.6 Structure of a powerJobs script	57
6.7 Error Handling	59
6.8 Adding jobs	60
6.9 Environment Variables	61
6.10 powerJobs file object	62
6.11 Customizing the file conversion	63
7. Tutorials	66
7.1 Adding the file revision to the PDF name	66
7.2 Converting PDF to PDF/A	66

7.3 Convert PDF to DWF	72
7.4 How to handle different sized AutoCAD DWGs	72
7.5 Pdf on Item Lifecycle Transition	73
8. FAQ	74
8.1 How to get a filename without extension?	74
8.2 Where is the documentation for the Vault API?	75
8.3 Where is the powerJobs2013 documentation?	75
9. Troubleshooting	75
9.1 Logging Levels	75
9.2 DLL not found in Powergui	76
9.3 Jobprocessor freezes	76
9.4 The powerJobs button doesn't show up in the menu	76
9.5 powerVault deadlocks in powershell	78
9.6 AutoCAD COM exceptions	79
9.7 powerJobs doesn't start with Vault Workgroup	80
9.8 Startup-Oeration: 'RestrictionChecks' failed!	81
9.9 PDF is created for the wrong file version	82
9.10 DWG/DXF Export of 2D Inventor files uses the wrong configuration file	82
10. Change logs	82
10.1 15.0.32	83
10.2 15.1.49	83
10.3 15.1.67	84
10.4 15.1.73	84
10.5 15.1.87	84
10.6 15.2.6	85
10.7 15.2.41	85
10.8 15.2.66	85
11. What's new in powerJobs2015	85

Installation

Requirements

- Vault Workgroup Client 2015/2015R2
- Vault Professional Client 2015/2015R2
- powershell 2.0 or higher ([Download PS4](#))
- According CAD Applications such as Inventor and DWGTrueView in order to create PDF or other formats out of DWG, IDW and the like.
- As powerJobs is an extension to the Vault JobProcessor, the system requirements defined by Autodesk leads. For more information on the Vault system requirements, use this link: <http://knowledge.autodesk.com/support/vault-products/troubleshooting/caas/sfdcarticles/sfdcarticles/System-requirements-for-Autodesk-Vault-products.html>



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>



We recommend to install the Vault JobProcessor and powerJobs on a dedicated workstation. As CAD applications and potentially other applications shall run on such machine, a workstation is more appropriate from a hardware prospective. License for the used applications must be available either as single license or dedicated network license.

powerJobs and CAD

	Vault 2014 + pJ 2014	Vault 2015 + pJ 2015	Vault 2015R2 + pJ 2015R2
Inventor 2014	x	x	x
Inventor 2015		x	x
TrueView 2014	x	x	x
TrueView 2015		x	x

x... supports

Windows permissions

In order to execute and synchronize the jobs properly, the current windows user needs the following rights on:

Path	Required Rights
C:\ProgramData\coolOrange\powerJobs	"List folder / read data" and "Read extended attributes"
C:\ProgramData\Autodesk\Vault2015\Extensions\coolOrange.PowerJobs.Handler\PowerJobs.vcet.config	"Write"

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**

2015\Extensions\coolOrange.PowerJobs and **C:\ProgramData\Autodesk\Vault 2015\Extensions\coolOrange.PowerJobs.Handler**

- All job definition files, e.g. scripts and module libraries, are placed in **C:\ProgramData\coolOrange\powerJobs**
- The **powerJobs.exe** and all related files are placed in **C:\Program Files\coolOrange\powerJobs 2015**
- Shortcuts to open the powerJobs 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 2015" and run "Uninstall"

Getting started

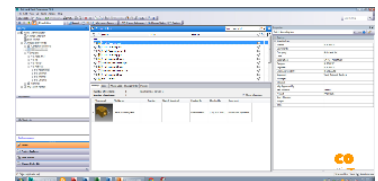
Creating a PDF from a Vaultfile

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](#)
- 6 [Result](#)

Select a Vaultfile

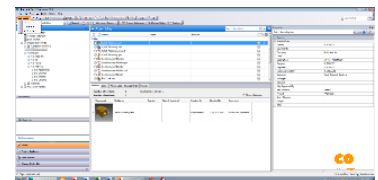
In this example we have selected the "**Catch Assembly.iam**". Afterwards the **powerJobs button** will be enabled.



Create PDF

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

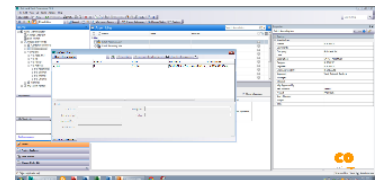
If you don't see this button then you have to reset the toolbars: Tools->Customize and reset all of them, then close the window.



Job Queue

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

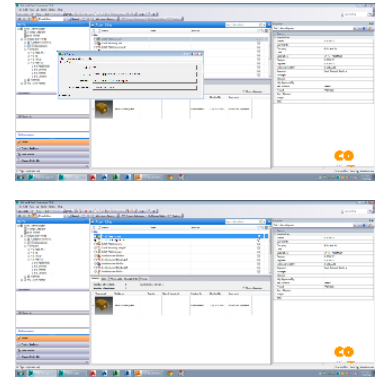
You can find the job queue in the menu at **Tools -> Job Queue**.



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 2015\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

Where can i find the activation dialog ?

After installing powerJobs you should run the JobProcessor and execute a Job.

When the Job is being executed the splashscreen should appear where you can activate the product.

Trial limitations

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

Trial duration

After the installation powerJobs is 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.

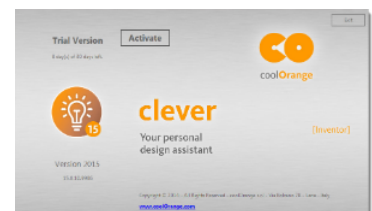
Order

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



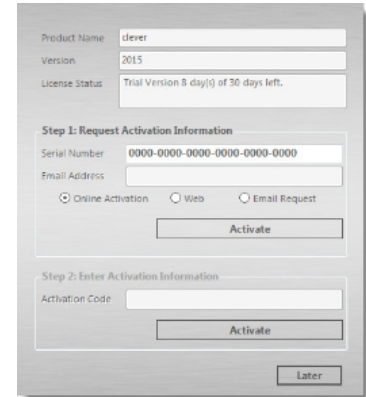
This is the preferred method but might not work if you use a proxy server to access the internet.

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.

The image shows a software activation window with a light gray background. At the top, there are three input fields: 'Product Name' with 'clever' entered, 'Version' with '2015' entered, and 'License Status' with 'Trial Version 8 day(s) of 30 days left.' entered. Below these is 'Step 1: Request Activation Information'. It contains a 'Serial Number' field with '0000-0000-0000-0000-0000-0000' entered, an 'Email Address' field, and three radio buttons: 'Online Activation' (selected), 'Web', and 'Email Request'. An 'Activate' button is at the bottom right of Step 1. Below Step 1 is 'Step 2: Enter Activation Information', which has an 'Activation Code' field and an 'Activate' button. At the very bottom right is a 'Later' button.

Features

AutoCAD Engine

PowerJobs internally has couple of engines that are used to handle your files. The AutoCAD engine is one of three engines used by powerJobs. The other two are the Inventor and the TrueView engine.

Executing jobs from the client

Executing jobs via lifecycle changes

Extended Jobprocessor UI

File format conversion

Per default powerJobs is able to use Inventor, AutoCAD or TrueView to convert files. It is possible to use other applications but you have to implement it yourself.

Integration to powerGate

Jobs with multilingual Vault support

The SYSPROPS variable is a language neutral substitute for system properties.

powerVault Cmdlets

The powerVault cmdlets are a collection of Vault related cmdlets. Their purpose is to reduce the amount of code and produce better maintainable code.

The sample jobs

Time triggered jobs

You can trigger a specific job at a certain time.

Trace Window

The trace window is a developer tool in powerJobs. It gives you the possibility to log debug information without the need to open the log file every time. With 15.1 the Log Window was added as a part of the PowerJobsProcessor.

Variable inspector

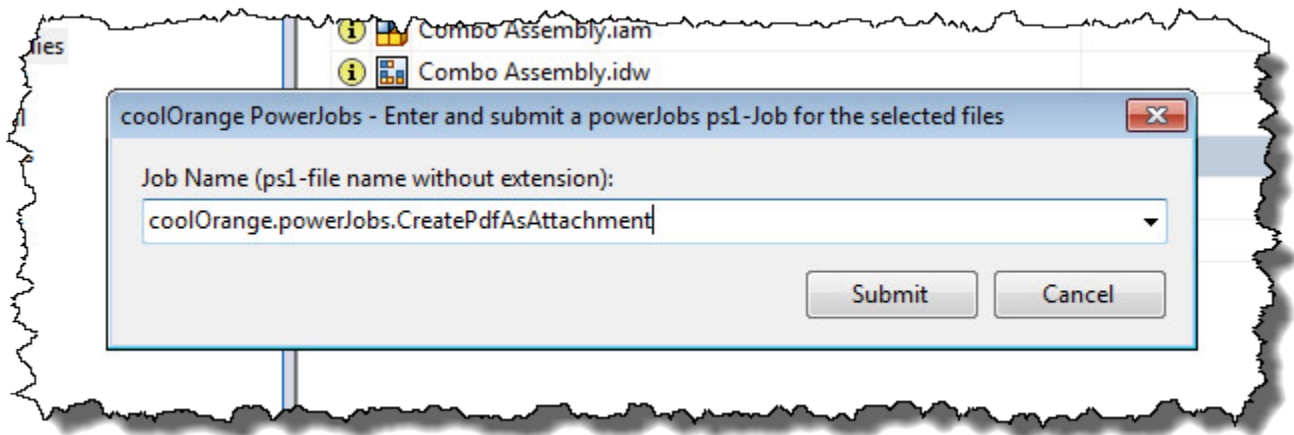
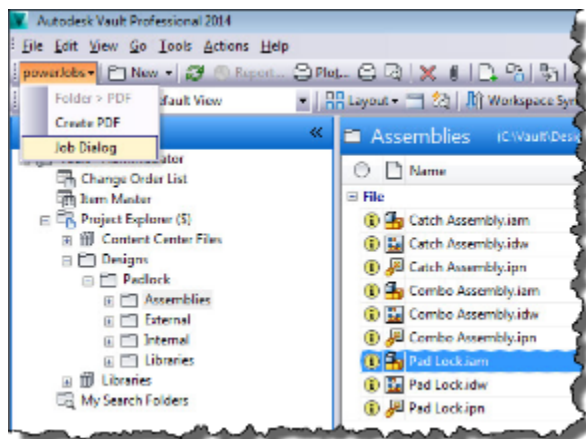
The inspector is a tool to show all variables of your current powershell console and their values.

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



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

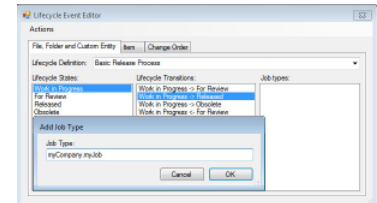
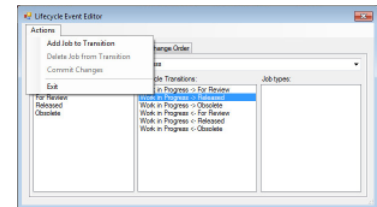
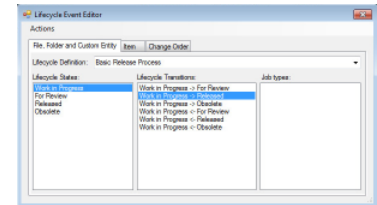
You want to add a the job during the transition from **"Work in Progress"** to **"Released"**. Our example job is **myCompany.myJob.ps1**

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.

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

Extended Jobprocessor UI

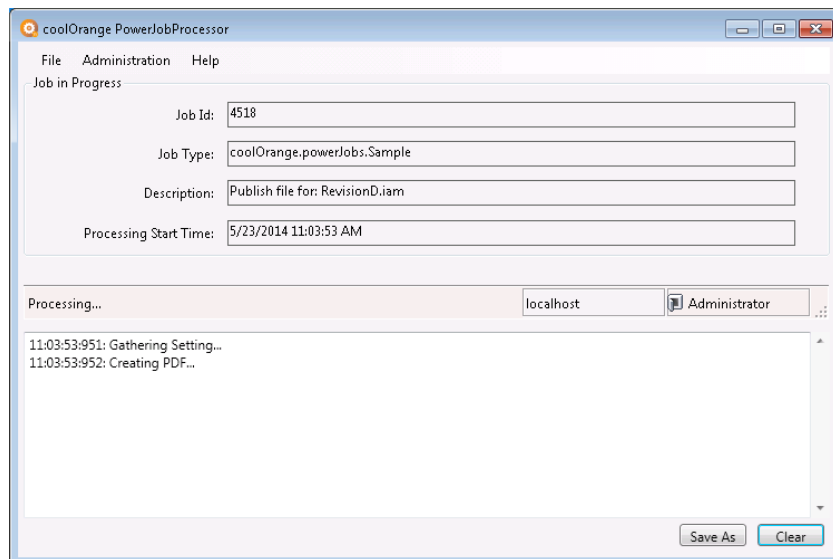
With powerJobs version 15.1 powerJobs got its own executable. It automatically loads the JobProcessor and provides additional features.

You can still start the jobprocessor through the jobprocessor.exe but we recommend to use the new powerjobs.exe.



Following features are only supported with the PowerJobsProcessor

- [Job Synchronization](#)
- [Trace Window](#)
- [Job Triggers](#)



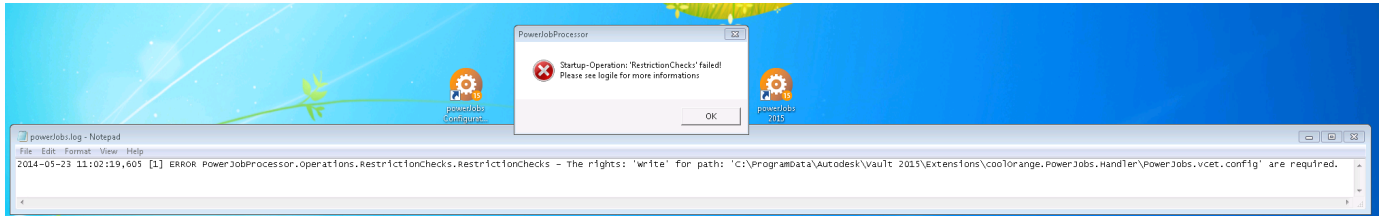
Job Synchronization

At every start of the application the Jobs placed in the "C:\ProgramData\coolOrange\powerJobs" folder are getting synchronized with the PowerJobs.vcet.config file to support the Jobs with the JobProcessor. For further informations see [Adding jobs](#).

Restriction Checks

At every execution of PowerJobsProcessor.exe, it checks if the current user has enough permissions to execute and synchronize jobs properly. If the user has not enough rights then powerJobs will not be started. In the log file will be written in detail what rights are missing on which path.

For further details check: [Windows rights](#)



Variable inspector

The inspector is a tool to show all variables of your current powershell console and their values.

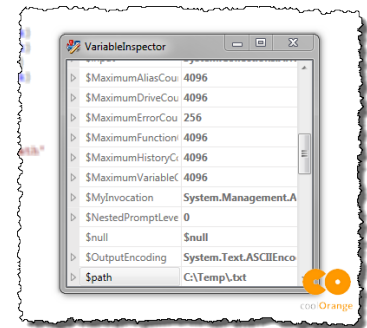
How to use it

You can open the inspector with **Show-Inspector**. If you put it somewhere in your job the inspector will show up at this point and pause the job. So you can find variables, which are not set correctly. Its a lot more convenient than the classical way to write them out into a textfile.

Example

Admittedly the error in the example is obvious, but it's just intended to show a possible purpose of the inspector. In a more complex script you can check your variables with this method without writing them out to a textfile.

```
function Get-Path{
    param($switch)
    switch($switch){
        a{$path = "C:\Temp\a.txt";break}
        b{$path = "C:\Temp\b.txt";break}
        c{$path = "C:\Temp\.txt";break}
        d{$path = "C:\Temp\d.txt";break}
    }
    return $path
}
$path = Get-Path -switch 'c'
Show-Inspector
Get-Content $path
```



powerVault Cmdlets

The powerVault cmdlets are a collection of Vault related cmdlets. Their purpose is to reduce the amount of code and produce better maintainable code.

Use of powerVault Cmdlets

If you don't know what a cmdlet is you may want to take a look at [this page\(MSDN\)](#). A complete list of them can be found [here](#).

File format conversion

Per default powerJobs is able to use Inventor, AutoCAD or TrueView to convert files. It is possible to use other applications but you have to implement it yourself.

Most of the functionality used to convert files can be found in the module 'C:\ProgramData\coolOrange\powerJobs\Modules\coolOrange.powerJ

obs.Publish.psm1'.

This module is already configured to allow for different file conversions, but you can customize it as needed.



For the moment Acad Engine is disabled by default for the PDF generation. To use Acad instead of TrueView, remove the flag -UseTrueView from the function SaveAs_PDF

Possible format conversions

Inventor, AutoCAD, TrueView: Source format

Target format: destination format

Valid extensions: Possible extensions for the target format

Inventor	AutoCad	TrueView	Target format	Valid extensions
iam, ipt, idw, dwg	dwg	dwg	PDF	.pdf
iam, ipt, idw, dwg	dwg	dwg	DWF	.dwf / .dwfx
ipt (sheet metal), idw, dwg	dwg		DXF	.dxf
iam, ipt, idw			DWG	.dwg
	dwg		IGES	.iges / .igs
iam, ipt	dwg *		STEP	.stp / .step
iam, ipt, idw, dwg	dwg		JPEG	.jpg
iam, ipt, idw, dwg	dwg		BMP	.bmp
iam, ipt, idw, dwg			GIF	.gif
iam, ipt, idw, dwg	dwg		TIFF*	.tiff / .tif
iam, ipt, idw, dwg	dwg		PNG	.png

*STEP creation for AutoCAD files needs AutoCAD Mechanical

*TIFF creation via Inventor supports only file extension '.tiff' and via AutoCAD only '.tif' is supported

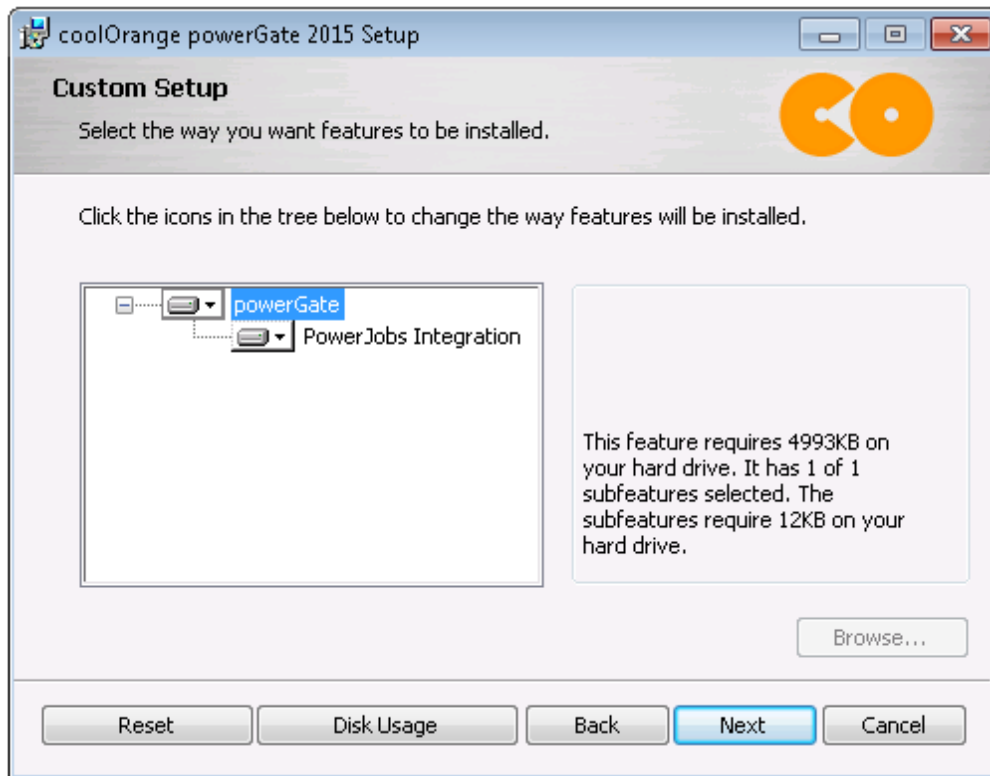


You can [customize](#) the default functionality of powerJobs to your needs.

Integration to powerGate

First step is to install [powerGate](#) after you have installed powerJobs.

When you run the setup, powerGate will ask you if you want to install the powerJobs integration:



The powerJobs integration should be selected as default. Just complete the installation.

Now you should find a new job into the jobs directory of powerjobs called: **coolOrange.powerGate.CreateAndUploadPDFtoSAP.ps1**

CreateAndUploadPDFtoSAP Job

This job can be triggered for IDW files and creates a PDF out of it.

By default the SAP-system <https://sap.coolorange.com> will be connected with the default user credentials.

To upload the PDF file to SAP, a unique identifier is required. Therefore you have two possibilities:

1. The file property 'SAP Documentnumber' is used as the unique identifier
2. The property does not exist, and a new number will be generated depending on the existing SAP-items

With this information a new entry in the entity 'DocumentInfoRecordDataCollection' will be created, which is linked to the entity 'DocumentInfoRecordOriginalCollection', where the file is uploaded.

The new entry for the entity 'DocumentInfoRecordDataCollection' is structured like this:

```
Documenttype = DEM
Documentnumber = UNIQUE_NUMBER
Documentversion = 01
Documentpart = 000
Description = FILENAME
```

The same data will be updated on the vault document UDP's:

```
SAP Documenttype = DEM
SAP Documentnumber = $sapDocumentNumber
SAP Documentversion = 01
SAP Documentpart = 000
```

Time triggered jobs

You can **trigger** a specific job at a certain **time**.

For example you can configure, that a job is executed every Friday in January and June at 02:30 pm. You only have to make a **settings file**

for your job.

Setting up a trigger

In the folder C:\ProgramData\coolOrange\powerJobs\Jobs create a text file with exactly the same name as your job, but with the file extension .SETTINGS instead of .PS1.



For example for the job coolOrange.powerJobs.CreatePdfAsAttachment.ps1 your settings file has to be called coolOrange.powerJobs.CreatePdfAsAttachment.settings

The settings file is written in [json](#).

- **TimeBased:** Must be in [cron](#) syntax
- **Vault:** The vault name, where the job should be triggered
- **Priority:** Jobs with higher priority will be executed first. Number between 1 and 99 are valid. The lower the number the higher the priority.
- **Description:** Description of the job.

Example

```
{
  "Trigger": {
    // This is a cron syntax expression. If you are not
    // familiar with cron, please see: http://www.cronmaker.com/
    // Here are some common cron expressions:
    // every minute: 0 0/1 * 1/1 * ? *
    // every weekday at 8th am: 0 0 8 ? *
    MON,TUE,WED,THU,FRI *
    "TimeBased": "0 0 8 ? * MON,TUE,WED,THU,FRI *",
    // This is the name of the Vault you want to trigger
    // the job
    "Vault": "Vault",
    // And this two parameters are optional and self
    // explaining:
    "Priority": 10,
    "Description": "This job is triggered weekdays at
    8:00 am"
  }
}
```



If some issues appear, then compare your settings with the [sample](#) which is delivered with the setup.



PowerJobs triggers a Job only if the same job isn't already pending in the job queue.

Example

You configured your job to be triggered every minute from 08:00 pm to 06:00 am. Your job processor is looking every 5 minutes for new jobs

At 08:00 the job will be triggered the first time and is pending in the job queue

At 08:01 the job will be triggered the second time but the first job is still pending

The second job at 08:01 won't be created

Samples

We recommend to people without deep knowledge on cron, to use online platforms like [CronMaker](#) to generate special cron expressions.

Here you can find some often used expressions:

Description	Cron expression
every 5 seconds	<code>* / 5 * * 1 / 1 * ? *</code>
always full minute and at 30 seconds	<code>0,30 * * 1 / 1 * ? *</code>
working days except on Saturday at 8:00 am	<code>0 0 8 ? * MON,TUE,WED,THU,FRI *</code>
every hour	<code>0 0 0 / 1 1 / 1 * ? *</code>

AutoCAD Engine



AutoCAD engine is still in technical preview mode! One of the known issue is described [here](#).

PowerJobs internally has couple of engines that are used to handle your files. The AutoCAD engine is one of three engines used by powerJobs. The other two are the Inventor and the TrueView engine.

Functionality

The AutoCAD engine can be used the same way as the TrueView engine but in addition it grants access to the AutoCAD API like the Inventor engine does for the Inventor API.



Samples

The "[SaveAs_OtherFormat](#)" function(`colorange.powerJobs.publish.psm1`) provided with powerJobs uses the AutoCAD API.

How to use the AutoCAD API in a job



event subscription

In .NET, you can subscribe or unsubscribe to an event by calling `add_` and `remove_` before the event name. e.g. `add_OnBeginPublish` to subscribe and `remove_OnBeginPublish` to unsubscribe the `OnBeginPublish` event.

AutoCAD API is exposed through the event called `OnBeginPublish` of the [Publisher](#) object which gets called after the document is opened and before it gets published. Look at the [OnBeginPublish](#) event for more details. When a user subscribes to this event through the powershell scripts, he can use the AutoCAD API to do all sort of things to the document before it gets published.

Example to Export DWG file to JPG file.

JPG sample

```
$publisher = $powerJobs.getPublisher("PDF")
$publisher.add_OnBeginPublish({
    param($publisher, $eventArgs)
    $originalValue = $eventArgs.Document.GetVariable("FILEDIA")
    $eventArgs.Document.SetVariable("FILEDIA", 0)
    $eventArgs.Document.SendCommand("_JPGOUT
C:\Temp\File.jpg$([System.Environment]::NewLine)")
    $$eventArgs.Document.SetVariable("FILEDIA", $originalValue)
})
$publisher.Open($file.id)
```

Creating PDFs, DWFs and DWFx without AutoCAD

If there is no AutoCAD installed DwgTrueView will be used instead. Other file conversions are not possible without AutoCAD as only these three formats are supported by DwgTrueView.

Explicitly using DwgTrueView instead of AutoCAD

You can do it by making a change in the PowerShell publish module sample which we have provided with the powerJobs. Open the 'C:\ProgramData\coolOrange\powerJobs\Modules\coolOrange.powerJobs.Publish.psm1' and you will find the Publish-VaultFile Cmdlet with a parameter switch called \$UseTrueView which is set to false by default. Pass the parameter *-UseTrueView* to not to use AutoCAD and use DwgTrueView instead.

Publish-VaultFile Cmdlet

```
...
#region Internals
function Publish-VaultFile {
PARAM(
[Parameter(Mandatory=$True,Position=1)]
[PSCustomObject]$File,
[STRING]$ToFile,
[STRING]$Format="PDF",
[STRING]$Options=$null,
[SWITCH]$Open = $false,
[SWITCH]$UseTrueView = $false,
[ScriptBlock]$OnBegin = $null
)
    $publisher=$powerJobs.GetPublisher($Format)
    $publisher.OutputFile = $ToFile
    $publisher.UseAutoCad = !($UseTrueView.ToBool())
    if($OnBegin -ne $null) {
        $publisher.add_OnBeginPublish({
            param($publisher, $eventArgs)
            $OnBegin.Invoke($publisher, $eventArgs)
        })
    }
    if($Options -ne $null) {
        $publisher.Options= $Options
    }
    if(!$Open) {
        return $publisher.Publish($File.Id)
    } else {
        return $publisher.Open($File.Id)
    }
}
...
```

And an example client script below forces to use DWGTrueView application instead of AutoCAD.

using DWGTrueView

```
Publish-VaultFile -File "C:\Temp\fileToConvert.dwg" -ToFile
"C:\Temp\ConvertedFile.pdf" -Fromat "PDF" -UseTrueView
```

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

coolOrange.powerJobs.CreatePdfForAllFilesInFolder expects a folder id as job parameter. It can be executed in the Vault Client through powerJobs->Folder > PDF, but only if you have highlighted a folder in Vault. It works only for the following types:

- dwg
- idw

The job will create a separate coolOrange.powerJobs.CreatePdfAsAttachment job for each document contained in the Vault folder. You can

use this job to create PDFs for a large number of documents.

SaveLocalAsSheetMetalDxf

coolOrange.powerJobs.SaveLocalAsSheetMetalDxf creates dxf files from Inventor sheet metal ipt files. In order to generate dxf files Inventor requires additional information. You can use this sample to learn how to do this with powerJobs.

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

Modules

1. coolOrange.powerJobs.CadHelper.psm1
2. coolOrange.powerJobs.VaultHelper.psm1

CadHelper

Contains functions to check if the necessary CAD applications are installed.

VaultHelper

Contains functions to simplify the access to Vault folders and files.

Trace Window

The trace window is a developer tool in powerJobs. It gives you the possibility to log debug information without the need to open the log file every time. With 15.1 the Log Window was added as a part of the [PowerJobsProcessor](#).

How to use it

The Trace Window is using the PowerJobs.dll.log4net file to log informations from the PowerJobs.Handler Add-In. What is getting logged is configurable in the log4net file from PowerJobs.

By default Info, Error and Fatal messages are logged.

 More Information about logging level!

```
</appender>
<appender name="MsgAppender"
→   type="powerJobs.Interfaces.MsgSenderAppender, powerJobs.Interfaces" ->
→   <layout type="log4net.Layout.PatternLayout">
→     <conversionPattern value="[%-5level] - - %message" -/>
→   </layout>
→   <filter type="log4net.Filter.LevelRangeFilter">
→     <levelMin value="INFO" -/>
→     <levelMax value="FATAL" -/>
→   </filter>
</appender>
```

To manually log information during script execution you can use the **Add-Log** command-let in your script to fill the window with the debug information you need.

Example



Adding stuff to the log window

```
function Write-Log{
    param($switch)
    switch($switch){
        a{
            Add-Log -Text "a"
        }
        b{
            Add-Log -Text "b"
        }
        c{
            Add-Log -Text "c"
        }
        d{
            Add-Log -Text "d"
        }
    }
}
Write-Log -switch @('a','c')
```

Add logging information

Add-Log

Jobs with multilingual Vault support

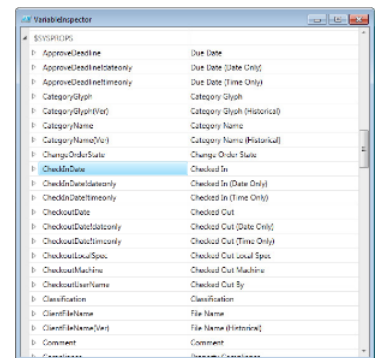
The \$SYSPROPS variable

The SYSPROPS variable is a language neutral substitute for system properties.

The substitute will translate to a system property's name in the currently used Vault's language. It consists of a list of key-value pairs. The key will be used in your scripts and is constant. The values are the system property names. These are generated dynamically depending on the Vault's language.



The keys are actually objects not strings. They contain all the members of the class "*Autodesk.DataManagement.Client.Framework.Vault.Currency.Properties.PropertyDefinition*".



How to use \$SYSPROPS

Syntax

```
$SYSPROPS.KEY
```

How it works

First powershell resolves the \$SYSPROPS variable. The result of this depends on your Vault's language. The result of the \$SYSPROPS variable will be used to further resolve the script.

The left script will translate to something like on the right side.

```
$file.($SYSPROPS.ClientFileName)
```

```
#English Vault  
$file.'File Name'  
#German Vault  
$file.Dateiname
```

Advanced uses

Like mentioned before every key contains the members from its respective webservice object. This allows for example to get all the entity classes that use a certain property.

```
$SYSPROPS.CategoryName.AssociatedEntityClasses
```



\$SYSPROPS becomes initialized when calling one of the PrepareEnvironment functionalities of powerJobs (PrepareEnvironment, PrepareEnvironmentForFile or PrepareEnvironmentForFolder)

This variable is created only if you first use Open-VaultConnection to connect to your Vault.

Configuration

Configure Pdf creation for AutoCAD Dwgs

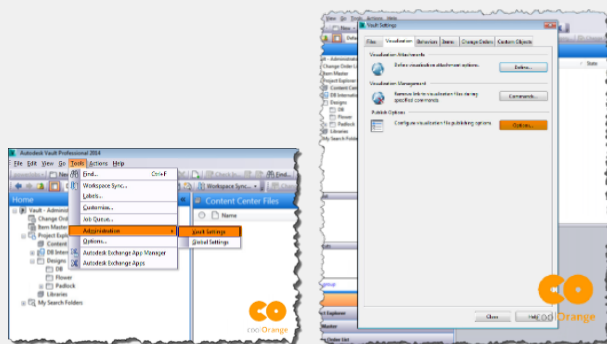
PowerJobs uses Autodesk technology for the actual PDF creation. Therefore we cannot cover every detail of the possible settings. But we try to give you a good overview of the most common settings. There are four locations where you can configure the pdf creation.

These locations are:

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

Vault

You can find the "**Publish options**" dialogue at "**Tools**" -> "**Administration**" -> "**Vault Settings**" -> "**Visualization**" -> "**Option S...**"



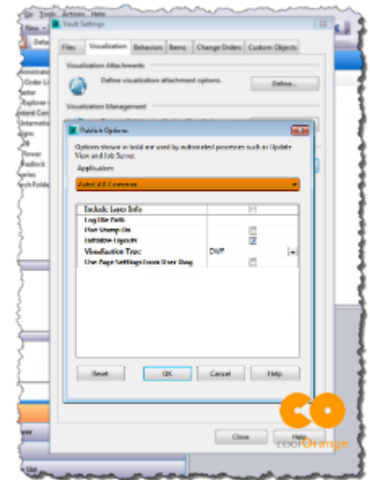
There are two important categories:

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 a file within the %appdata% folder. It's recommended to uncheck this setting.

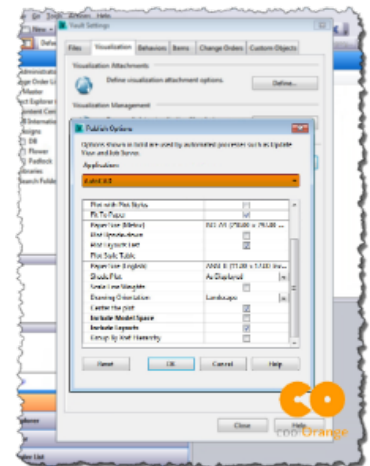


and AutoCAD



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.



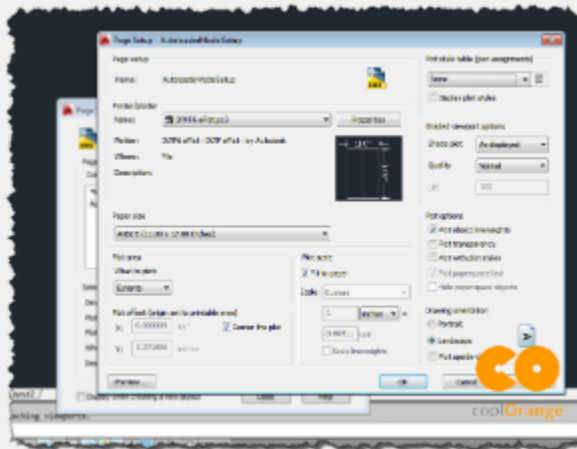
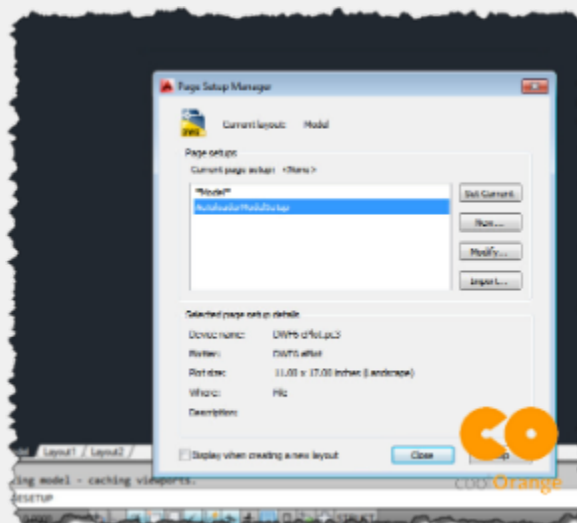
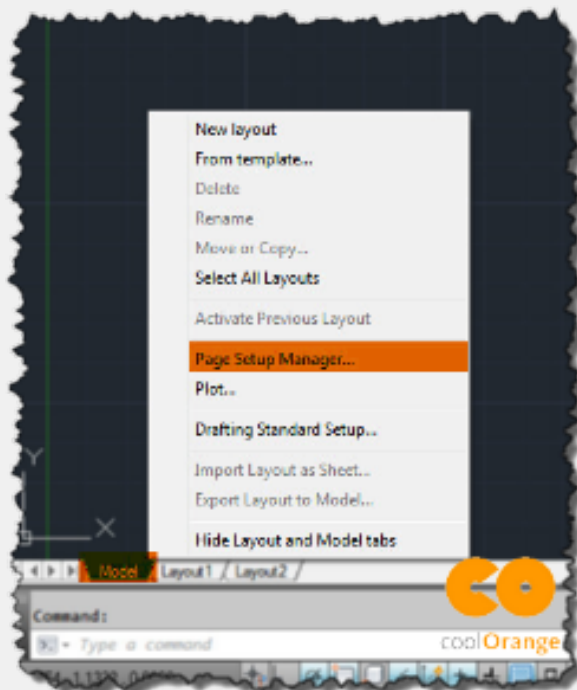
TrueViewSetup.dwg

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\coolorange\powerJobs\Modules\Publish\TrueViewSetup.dwg**"



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...**".

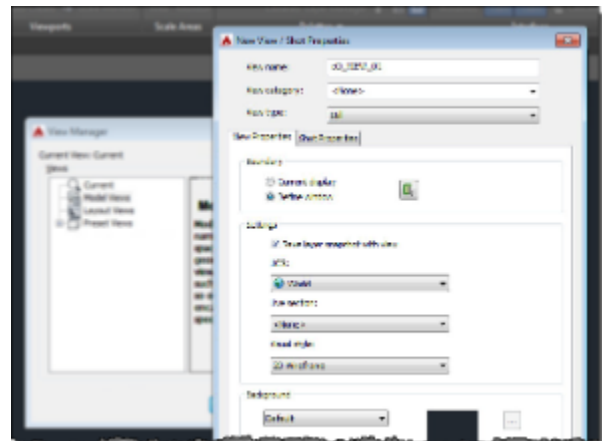
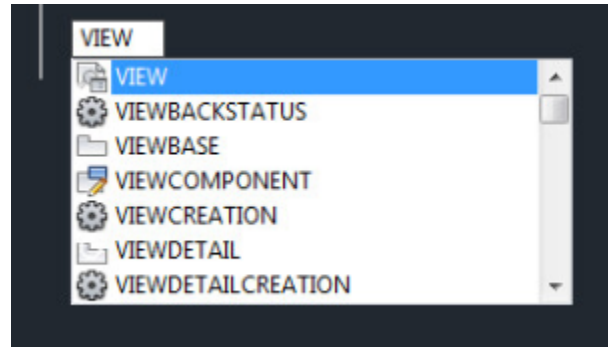
✓ The tab you wish to edit has to be selected.

Plotting Views

In order to plot certain views you have to create a view with the same name in your *TrueViewSetup.dwg* and set the Plot area to "View".

To do so open your *TrueViewSetup.dwg*, type VIEW in your AutoCAD command line and create a new view.

Afterwards open your page setup manager and select your view



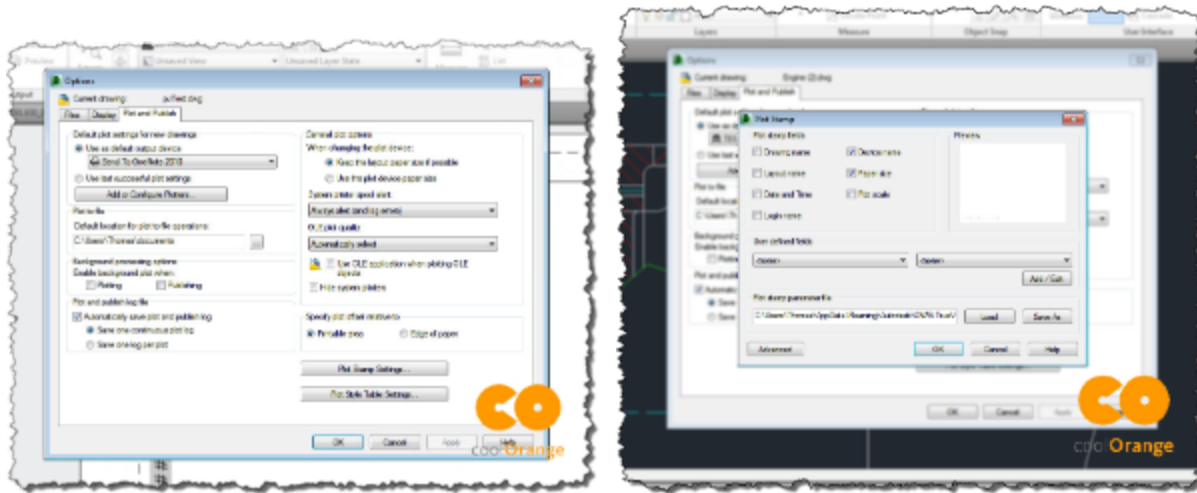
Dwg TrueView options

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

- [Code Reference](#)
- [Code Snippets](#)
- [Preparing your environment](#)
- [IDEs for powershell](#)
- [Powershell, scripts and modules](#)
- [Structure of a powerJobs script](#)
- [Error Handling](#)
- [Adding jobs](#)
- [Environment Variables](#)
- [powerJobs file object](#)
- [Customizing the file conversion](#)

Code Reference

Cmdlets

Add-Log

Writes content to the Trace window.

Add-VaultFile

Adds a new file to vault.

Get-VaultFile

Creates a powerJobs file object for the file, that meets your search criteria. It can be used to download the file to a local directory.

Get-VaultFileAssociations

Returns a collection of the file associations.

Get-VaultFiles

Returns an array of file objects that match your parameters.

Open-VaultConnection

Opens a connection to the vault that is stated in the parameters.

Save-FileAs

Publishes the vault file to different formats like PDF,DWF,DWFX,JPG,BMP,TIFF,STEP and others

Show-Inspector

Opens the inspector window. The inspector is a debugging tool, that shows all existing variables and their values.

Update-VaultFile

This function has a lot of utilities to update and manipulate files in vault.

Add-Log

Writes content to the Trace window.

Syntax

```
Add-Log -Text <string> [<CommonParameters>]
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
String	-Text	The message, that will be written to the Trace Window	input	true	

Return type

void

Remarks

The log window doesn't have to be open in order to add content to it.

Examples

The trace window is a developer tool in powerJobs. It gives you the possibility to log debug information without the need to open the log file every time. With 15.1 the Log Window was added as a part of the [PowerJobsProcessor](#).

Add-VaultFile

Adds a new file to vault.

Syntax

```
Add-VaultFile -From <string> -To <string> [-Force <bool>] [-Hidden <bool>]  
[-Comment <string>] [-FileClassification <string>] [<CommonParameters>]
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
string	From	Absolute Path to a local file.	input	true	
string	To	Absolute vault path. It has to include the filename.	input	true	
bool	Force	If the target directory doesn't exist it will be created.	input	false	true
bool	Hidden	If true the file will be hidden in vault.	input	false	false
string	Comment	A comment for the checkin.	input	false	Generated by coolOrange powerJobs
string	FileClassification	The classification of the file.	input	false	

Return type

powerJobs file object

Remarks

This function is only for **new** files. If you want to alter existing files use [Update-VaultFile](#).

If the target file already exists it will be checked out and the new file will be checked in.

Examples

Add File

```
$file = Add-VaultFile -From "C:\Temp\pJ_1.ipt" -To "$/PowerJobsTestFiles/pJ_7.test"
```


Add File with comment and classification

```
$file = Add-VaultFile -From "C:\Temp\pJ_1.dwf" -To "$/PowerJobsTestFiles/pJ_7.test"
-comment "Test" -FileClassification "DesignVisualization"
```

Get-VaultFile

Creates a **powerJobs file object** for the file, that meets your search criteria. It can be used to download the file to a local directory.

Syntax

```
Get-VaultFile [-File <string>] [-FileId <long>] [-Properties <hashtable>]
[-DownloadPath <string>] [<CommonParameters>]
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
string	File	Absolute vault path to a file.	input	false	
long	FileId	EntityIterationId or entityMasterId of a vault file.	input	false	
hashtable	Properties	This will give you a file with matching properties.	input	false	
string	DownloadPath	This is the location where the file and all it's references will be downloaded	input	false	

Return type

powerJobs file object

Remarks

If multiple files are possible the function returns the first file found. In case you want to get multiple files use [Get-VaultFiles](#).

Examples

```
$file = Get-VaultFile -properties @{"Description" = "TEMPLATE"; "Part Number" =
"Dial"}
```

```
$file = Get-VaultFile -File '$/PowerJobsTestFiles/pJ_6.idw'
```

Downloading the file

```
$file = Get-VaultFile -File '$/Assemblies/Catch Assembly.iam' -DownloadPath  
"C:\Temp\TestDownload"
```

If the directory "C:\Temp\TestDownload" does not exist, it will be created.

Attention: This does not mean that the file 'Catch Assembly.iam' is located directly under "C:\Temp\TestDownload"

When downloading the file, you have to remember that the file, and all its references will be downloaded into this directory. The goal of this functionality is, that the file can be opened without problems out from here.

The folder structure of the files in Vault is maintained, and they will not be renamed.

To retrieve the location of where the main file was downloaded, you can use the additional property: \$file.LocalPath

Get-VaultFileAssociations

Returns a collection of the file associations.

Syntax

```
Get-VaultFileAssociations -File <string> [-Attachments] [-Dependencies]  
[ <CommonParameters> ]
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
String	File	Absolute vault path to a file.	input	true	
switch	Attachments	If the switch is set only attachments will be returned.	input	false	false
switch	Dependencies	If the switch is set only dependencies will be returned.	input	false	false

Return type

psCustomObject

Remarks

The -Dependencies and -Attachments switches cannot be combined. At the moment only **file to file** associations are possible. For other things like **file to item** you have to use the vdf.

Examples

Get all associations

```
$allChilds = Get-VaultFileAssociations -File "$/PowerJobsTestFiles/pJ_5.iam"
```

Get dependencies only

```
$deps = Get-VaultFileAssociations -File "$/PowerJobsTestFiles/pJ_5.iam"
-Dependencies
```

Get attachments only

```
$attmnts = Get-VaultFileAssociations -File "$/PowerJobsTestFiles/pJ_5.iam"
-Attachments
```

Get-VaultFiles

Returns an array of file objects that match your parameters.

Syntax

```
Get-VaultFiles [-FileName <string>] [-Folder <string>] [-Properties <hashtable>]
[<CommonParameters>]
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
String	-FileName	Name of a file in vault.	input	false	
String	-Folder	Absolute path to a vault folder	input	false	
Hashtable	-Properties	A hashtable with one or multiple properties.	input	false	

Return type

System.Array

Remarks

If you want a single specific file use [Get-VaultFile](#).

Examples

```
$files = Get-VaultFiles -Folder '$/Designs/2014/03/0'

$files = Get-VaultFiles
Foreach($file in $files){
    $file.'File Name'
}
```

Open-VaultConnection

Opens a connection to the vault that is stated in the parameters.

Syntax

```
Open-VaultConnection [-Vault <string>] [-Server <string>] [-User <string>]  
[-Password <string>] [<CommonParameters>]
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
String	-Vault	The name of the vault for which the connection should be made	input	false	Vault
String	-Server	The Ip or hostname of the server on which the vault server is installed	input	false	localhost
String	-User	A valid user, that is able to login to the target vault	input	false	Administrator
String	-Password	The users password	input	false	""

Return type

String

Remarks



This function is only needed to debug a job. You don't need it to execute a job from the job processor. It creates the **\$powerJobs**, **\$vault**, **\$vaultconnection** and **\$vaultExplorerUtil** objects.

You can use Open-VaultConnection in the PrepareEnvironment function in the else part of **if(\$IAMRunningInJobProcessor -eq \$true)**, so you don't have to execute it every time you work on a job.

Save-FileAs

Pubilshes the vault file to different formats like PDF,DWF,DWFX,JPG,BMP,TIFF,STEP and others

Syntax

```
Save-FileAs $file "C:\Temp\Test.pdf "
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
------	------	-------------	--------------	-----------	---------------

PsObject	-File	The vault file that will be published (powerJobs file object)	input	true	
String	-LocalDestFile	The location of the required output file	input	true	

Return type

bool

When this function returns true, the \$file object was correctly exported to the local destination path. Otherwise false will be returned. In that case exceptions has to be retrieved from the logfile.

Examples

Export an assembly to different formats:

```
$file = Get-VaultFile -File "$/Designs/Catch Assembly.iam"
Save-FileAs $file "C:\Temp\Catch Assembly.pdf"
Save-FileAs $file "C:\Temp\Catch Assembly.dwf"
Save-FileAs $file "C:\Temp\Catch Assembly.tiff"
Save-FileAs $file "C:\Temp\Catch Assembly.step"
```

See Also

To get more details about what formats can be exported by default and how to change this functionality [read more about this here!](#)

Show-Inspector

Opens the inspector window. The inspector is a debugging tool, that shows all existing variables and their values.

Syntax

```
Show-Inspector [-ElementToHighlight <string>] [<CommonParameters>]
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
String	-ElementToHighlight	The inspector jumps to the passed variable. E.g. '\$file'	input	false	

Return type

void

Update-VaultFile

This function has a lot of utilities to update and manipulate files in vault.

Syntax

```
Update-VaultFile -File <string> [-Comment <string>] [-Category <string>] [-Status
<string>] [-LifecycleDefinition <string>] [-Revision <string>] [-RevisionDefinition
<string>] [-Properties <hashtable>] [-AddChilds <string[]>] [-Childs <string[]>]
[-RemoveChilds <string[]>] [-AddAttachments <string[]>] [-Attachments <string[]>]
[-RemoveAttachments <string[]>] [<CommonParameters>]
```

Parameters

Type	Name	Description	input/output	Mandatory	Default value
string	File	Full path to the file, that should be updated.	input	true	
string	Comment	Comment for the file history.	input	false	
string	Category	Sets the category of the file.	input	false	
string	Status	Sets the status of the file.	input	false	
string	LifecycleDefinition	Sets the lifecycle of the file.	input	false	
string	Revision	Sets the revision of the file.	input	false	
string	RevisionDefinition	Sets the revision definition of the file.	input	false	
hashtable	Properties	Changes the user defined properties of the file.	input	false	
string	AddChilds	Adds new childs to the file. An array with absolute vault paths is expected.	input	false	
string	Childs	Overrides all childs with the passed childs.	input	false	
string	RemoveChilds	Removes Childs from the file. An array with absolute vault paths is expected.	input	false	
string	AddAttachments	Adds new attachments to the file. An array with absolute vault paths is expected.	input	false	
string	Attachments	Overrides all attachments with the passed attachments.	input	false	
string	RemoveAttachments	Removes attachments from the file. An array with absolute vault paths is expected.	input	false	

Return type

powerJobs file object

Remarks



You can only make changes to the files you could do with the vault client. E.g. It is not possible to change the state of a file to something that isn't defined in the lifecycle definition.

Unlike in the vault vdf there are no lds needed for the Update-VaultFile parameters. The lds are handled by powerJobs.

- To refer to a file you need to pass an absolute vault path. E.g. "\$/Design/powerJobsTest/pJ_5.ipt"
- For categories, states, etc. you need to pass the full name. E.g. "Engineering", "Work in Progress"

Examples

```
$parent = Get-VaultFile -File "$/PowerJobsTestFiles/pJ_4.iam"
$child = Add-VaultFile -From "C:\Temp\pJ_1.ipt" -To
"$/PowerJobsTestFiles/pJ_4_3.ipt"
$parentUpdated = Update-VaultFile -File $parent.'Full Path' -Attachments
@($child.'Full Path') -Comment "Set one attachment only"
```

```
$parent = Get-VaultFile -File "$/PowerJobsTestFiles/pJ_2.iam"
$child1 = Get-VaultFile -File "$/PowerJobsTestFiles/pJ_2_1.ipt"
$child2 = Get-VaultFile -File "$/PowerJobsTestFiles/pJ_2_2.ipt"
$child3 = Get-VaultFile -File "$/PowerJobsTestFiles/pJ_2_3.ipt"
$parentUpdated = Update-VaultFile -File $parent.'Full Path' -RemoveChilds
@($child3.'Full Path') -Comment "Child 3 removed"
```

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
	Bool	UseAutoCad	Default value is true. When setting to false, and AutoCAD is installed on the machine, TrueView will be used instead of AutoCad for publishing	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	Removes code from the OnBeginPublish event.

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 $PublishEventArgs.Document or $PublishEventArgs.Application...
})
```

Parameters

Delegate [input]



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

Type	Value	Description
CommonPublisher	\$publisher	Contains the current publisher object

PublishEventArgs	\$PublishEventArgs	<p>\$PublishEventArgs.Document: The object passed to this variable depends on the files you are working with. See also Remarks.</p> <p>\$PublishEventArgs.Application: If you are working with Inventor files or AutoCAD files, powerJobs will pass an Inventor or AutoCAD object to this variable. You can access it with "\$PublishEventArgs.Application."</p>
------------------	--------------------	---

Remarks

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

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

If the publisher is working with TrueView, the \$PublishEventArgs.Document 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.Publish(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

\$Vault grants direct access to the vault api.

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

`$VaultConnection` grants direct access to the vault api.

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



powerJobs2013 documentation

From now on powerJobs2013 documentation is available as Pdf only. You can download it here: [Part1](#) [Part2](#)

Changing filestates, categories, properties

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.

Get all users of a certain group

Print via Inventor

Retrive the user that queued the job

Selected files to ZIP

Send email via jobserver

Changing filestates, categories, properties



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

```
$file = PrepareEnvironmentForFile "Catch Assembly.iam"

Update-VaultFile -File $file."Full Path" -Status "Released"
```

Changing state of child files

```
$file = PrepareEnvironmentForFile "Catch Assembly.iam"

$assoc = Get-VaultFileAssociations -File $file.'Full Path'
ForEach($file in $assoc){
    Update-VaultFile -File $file.'Full Path' -Status 'Work in Progress'
}
```

Changing categories

```
$file = PrepareEnvironmentForFile "Catch Assembly.iam"

Update-VaultFile -File $file.'Full Path' -Category 'Engineering'
```

Changing properties

```
Update-VaultFile -File '$/Padlock/Catch Assembly.iam' -Properties @{'Author' = 'Angela
Merkel'; 'Description' = 'Bundeskanzlerin'}
```

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>
```

```

$file = PrepareEnvironmentForFile "Catch Assembly.iam"

$templatePath = "C:\temp\template.txt"
$destinationPath = "C:\temp\template.html"

$regex = [regex]"\"{+(?i)\b[A-Z\s]+\b\}"
$template = Get-Content -Path $templatePath
$propertyMatches = $regex.Matches($template)
foreach($propertyMatch in $propertyMatches){
    $propertyName = $propertyMatch.Value.Trim( @('{'','}') )
    $propertyValue = $file.$propertyName
    $template = ($template -replace $propertyMatch.Value, $propertyValue)
}
$template | Out-File $destinationPath -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"

```

$file = PrepareEnvironmentForFile "Catch Assembly.idw"

$ext = $file.'File Extension'
$validExtensions = @("idw")
if($validExtensions -notcontains $ext){
    throw("$( $ext ) is not a valid extension")
}

$localDestFile = "C:\TEMP\" + $file.Name + ".dwg"
$publisher = $powerJobs.GetPublisher("PDF")
$publisher.OutputFile=$localDestFile

$publisher.add_OnBeginPublish(
{
    param($publisher, $document)
    $document.Document.SaveAs($localDestFile,$true)
})

if(!$publisher.Open($File.Id)){
    throw ("The .dwg-translation failed!")
}

```

Creating PDFs with their parents' properties

```

$file = PrepareEnvironmentForFile "Catch Assembly.idw"

$entityClassId = $file.'Entity Type ID'
$propDefs =
$vault.PropertyService.GetPropertyDefinitionsByEntityClassId($EntityClassId)
$propertyName = $propDefs | Where { $_.IsSys -eq $false } | Select -ExpandProperty
DispName
foreach($propertyName in $propertyName){
    [hashtable]$props += @{$propertyName = $file.$propertyName}
}
#Create Pdf and check it in
#...

$pdf = Get-VaultFile -File "$($file.'Full Path').pdf"
$newpdf = Update-VaultFile -File $pdf.'Full Path' -Properties $props

```

Export Vault Data to XML

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

```

$file = PrepareEnvironmentForFile "Catch Assembly.idw"

$xmlPath = "C:\Temp\$($file.Name).xml"

$entityClassId = $file.'Entity Type ID'
$propDefs =
$vault.PropertyService.GetPropertyDefinitionsByEntityClassId($EntityClassId)
$propertyName = $propDefs | Where { $_.IsSys -eq $false } | Select -ExpandProperty
DispName
foreach($propertyName in $propertyName){
    [hashtable]$properties += @{$propertyName = $file.$propertyName}
}
$output = "<root>`n"
foreach($key in $properties.keys){
    $output += "<$key>${$properties.$key}</$key>".Replace("
", "").Replace("(", "(").Replace(")", ")") + "`n"
}
$output += "</root>"

$output | Out-File $xmlPath

```

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()

```

Get all users of a certain group


```

function Get-UsersByGroup{
param(
[string]$groupName
)
    $group = $vault.AdminService.GetGroupByName($groupName)
    $groupInfo = $vault.AdminService.GetGroupInfoById($group.Id)
    $groupInfo.Users | ForEach-Object { [array]$users += $_ }
    return $users
}
$adminusers = Get-UsersByGroup 'Administrators'

```

Print via Inventor

```

$file = PrepareEnvironmentForFile "Catch Assembly.idw"
#region Settings
$printerName = "myPrinter"
$validExtensions = @("idw")
#endregion

$ext = $file.'File Extension'
if($validExtensions -notcontains $ext){
    $powerJobs.Log.Warn("$ext is not a valid file extension")
    exit 1
}
$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, $printerName)
    $printManager.ScaleMode = [Inventor.PrintScaleModeEnum]::kPrintBestFitScale
    $printManager.PrintRange = [Inventor.PrintRangeEnum]::kPrintAllSheets
    $printManager.PaperSize = [Inventor.PaperSizeEnum]::kPaperSizeA0
    $printManager.SubmitPrint()
})
if (!$publisher.Open($file.Id)){
    throw("Failed to open $($file.name)")
}

```

Retrieve the user that queued the job

```

$jobQueue = $vault.JobService.GetJobsByDate(1000,[DateTime]::MinValue)
$currentJob = $jobQueue | Where-Object {$_.Id.Equals($powerJobs.Job.Id)}
$userinfo = $vault.AdminService.GetUserByUserId($currentJob.CreateUserId)
$email = $userinfo.Email
#place additional code here. Instead of writing the information to a textfile you
could send an email to the user
$userinfo | Out-File 'C:\Temp\user.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



This script uses the `vaultEx.psm1` and `fileSystem.psm1` modules.

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

#region Settings
$zipFilePath = "C:\myZips"
$zipFileName = "$($zipFilePath)\$($file.Name).zip"
$tempDownloadPath = "C:\Temp\zip\$($file.name)"
#endregion
$files = Copy-VaultFile -file $file -downloadPath $tempDownloadPath -includeChildren
-recurseChildren -includeAttachments
New-Folder -path $zipFilePath
set-content $zipFileName ("PK" + [char]5 + [char]6 + ("$([char]0)" * 18))
$zipFile = (new-object -com shell.application).Namespace($zipFileName)
$fileToBeZipped = Get-ChildItem $tempDownloadPath

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

Send email via jobserver

```
$file = PrepareEnvironmentForFile "Catch Assembly.iam"

$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
$smtp = "yourSMTPServer"
$password = ConvertTo-SecureString -AsPlainText "YourPassword" -Force
$cred = new-object Management.Automation.PSCredential $from, $password

$sendattachment = $true

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
}
```

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.

```
$file.SendTo
```

Send notification via email

```

$file = PrepareEnvironmentForFile "Catch Assembly.idw"

$vaultname = $vaultConnection.Vault
$vaultserver = $vaultConnection.Server
$folderpath = $file.path.replace("$","%24").replace("/","%2f")
$fullPath = $folderpath + "%2f" + $file.Name.Replace(" ","+")
$link =
"http://$( $vaultserver )/AutodeskDM/Services/EntityDataCommandRequest.aspx?Vault=$( $vaultname )&ObjectId="+$fullPath+"&ObjectType=File&Command=Select"

$lfctransId = $powerJobs.Job.Params["LifeCycleTransitionId"]
$lfctrans =
$vault.DocumentServiceExtensions.GetLifeCycleStateTransitionsByIds(@($lfctransId))
$lfcs =
$vault.DocumentServiceExtensions.GetLifeCycleStatesByIds(@($lfctrans[0].FromId,
$lfctrans[0].ToId))
$soldstate = $lfcs[0].DispName
$newstate = $lfcs[1].DispName

$from = "jobserver@yourcompany.com"
$to = "receiver@yourcompany.com"
$receivername = "receiver"
$subject = "The file $($file.Name) has changed from $oldState to $newState"
$body = "Dear $($receivername), if you like to view the related document, just follow
this link: $link"
$smtp = "yourSMTPServer"
$password = ConvertTo-SecureString -AsPlainText "YourPassword" -Force
$cred = new-object Management.Automation.PSCredential $from, $password
Send-MailMessage -From $from -To $to -Subject $subject -Body $body -SmtpServer $smtp

```

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. Open your IDE by using the shortcut in startmenu/coolorange/powerJobs/Tools and create a new file
3. Execute [Open-VaultConnection](#) to get access to your desired vault.



You can add **Open-VaultConnection** in the prepare PrepareEnvironmentForFile method in line 24. So you will get a vault connection every time you run PrepareEnvironmentForFile from your ide and don't have to add it to every script

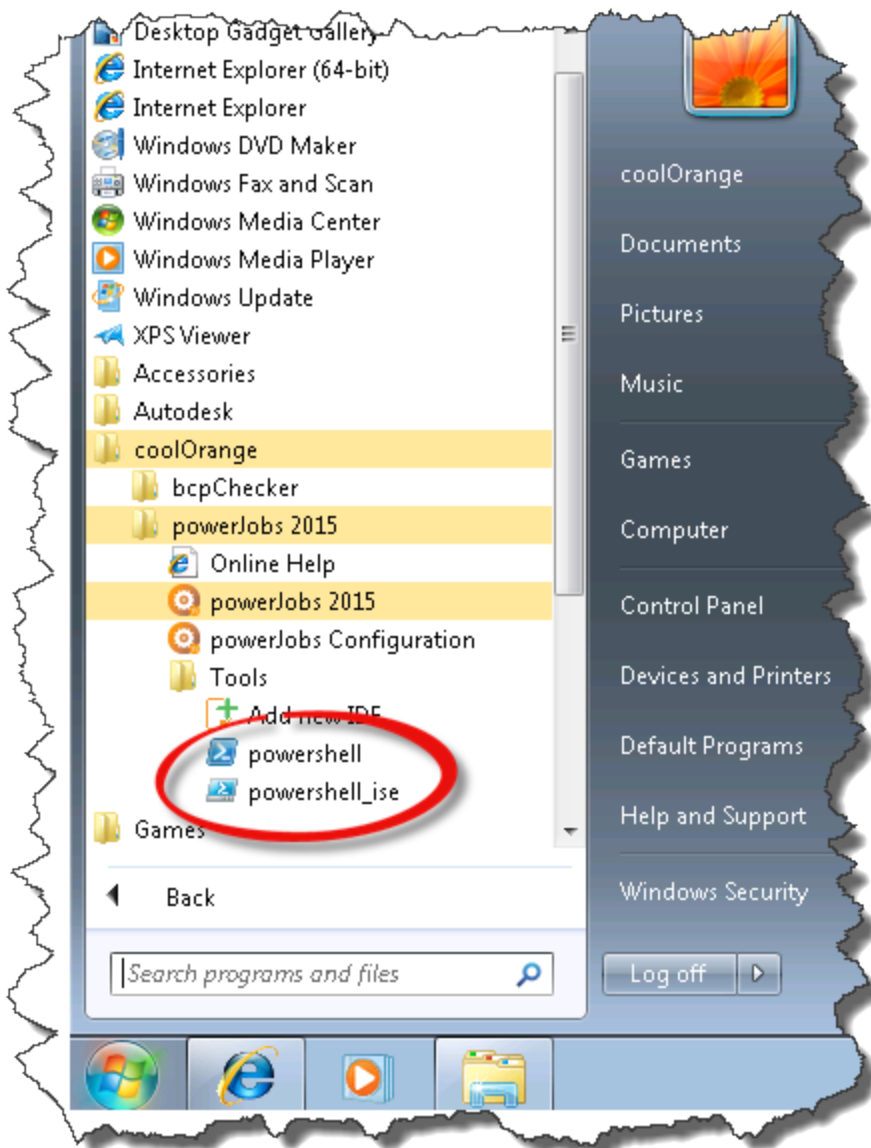
IDEs for powershell

- [PowerGUI](#)
- [Powershell 2.0 ISE](#)
- [I want to manually configure my powerShell processes to debug powerJobs scripts](#)



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.

By default powerJobs comes with two IDE's that are installed on each Windows machine: **powershell.exe** and **powershell_ise.exe**. Please use this shortcuts to start your IDE for debugging e.g. powerJobs scripts.



You will find the targeting batch-files in the powerJobs installation directory 'C:\Program Files\coolOrange\powerJobs 2015\Tools'.

Starting the powerShell process with this batch files has some advantages:

- your powerShell process is allowed to run the .Net 4 runtime environment
- all your powerJobs modules and powerVault is automatically loaded
- if you are using powershell.exe (the host without UI), the [VDF deadlock bug](#) is fixed automatically

This means your powerShell environment will not be changed, but if you run one of the shortcuts, the process is already setup to debug powerJobs scripts or play with powerVault cmdlets!

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>.


```
Set-ExecutionPolicy RemoteSigned
```

By default the value is set to Restricted, therefore script execution is disabled on the machine.

Support for .net 4

Out of the box, the powershell ISE, the powershell.exe and the powerGUI that are targeting powerShell version 2, are configured to use the .net 2.0 runtime. However, to run and debug scripts for powerJobs the processes must be configured to use the .net 4 runtime.

A common solution is to provide a config file for the executable. 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.

Import powerVault

Run the following line of code and powerVault will be loaded into your powerShell session:

```
Import-Module powerVault
```

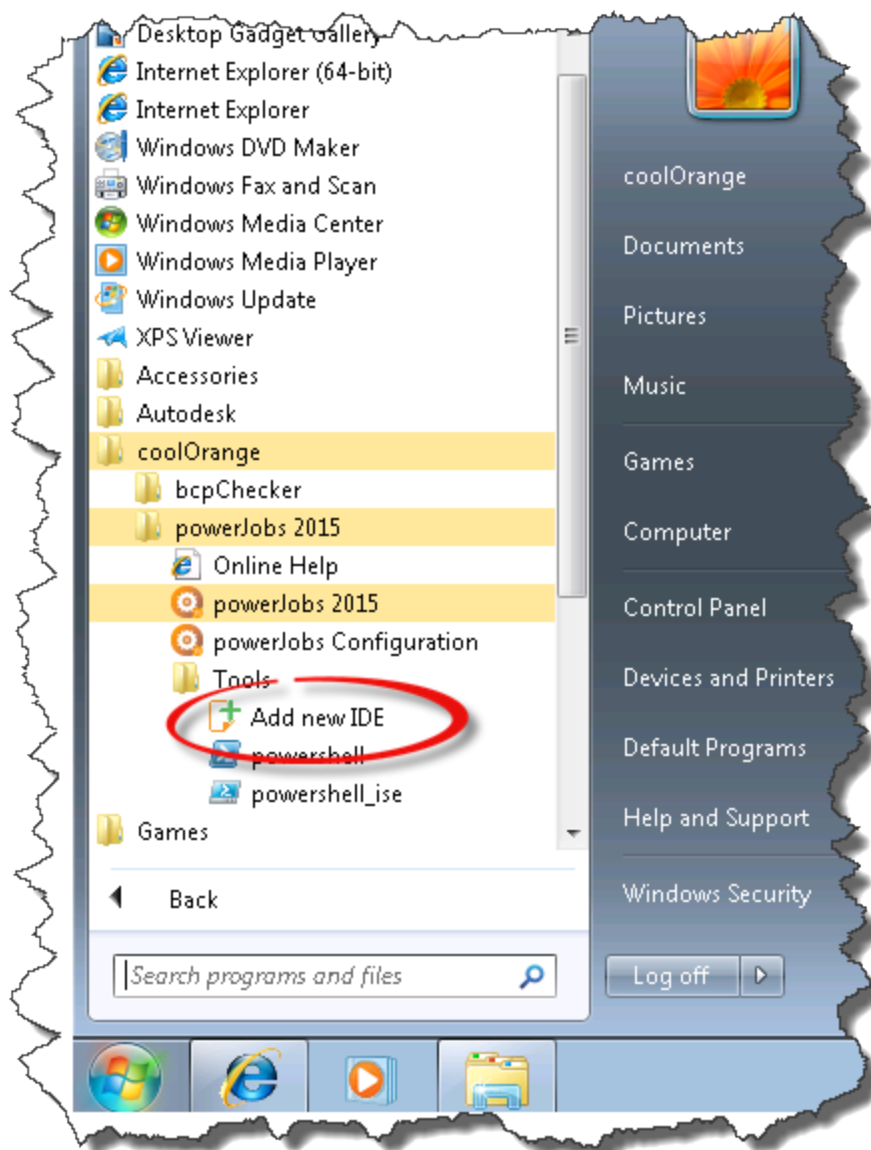
Adding a new IDE

Adding a new IDE to the powerJobs default powerShell IDE's is very simple.

Just click execute the "Add new IDE" shortcut in the Tools directory or run the file 'C:\Program Files\coolOrange\powerJobs 2015\Tools\Add_new_IDE.ps1'.

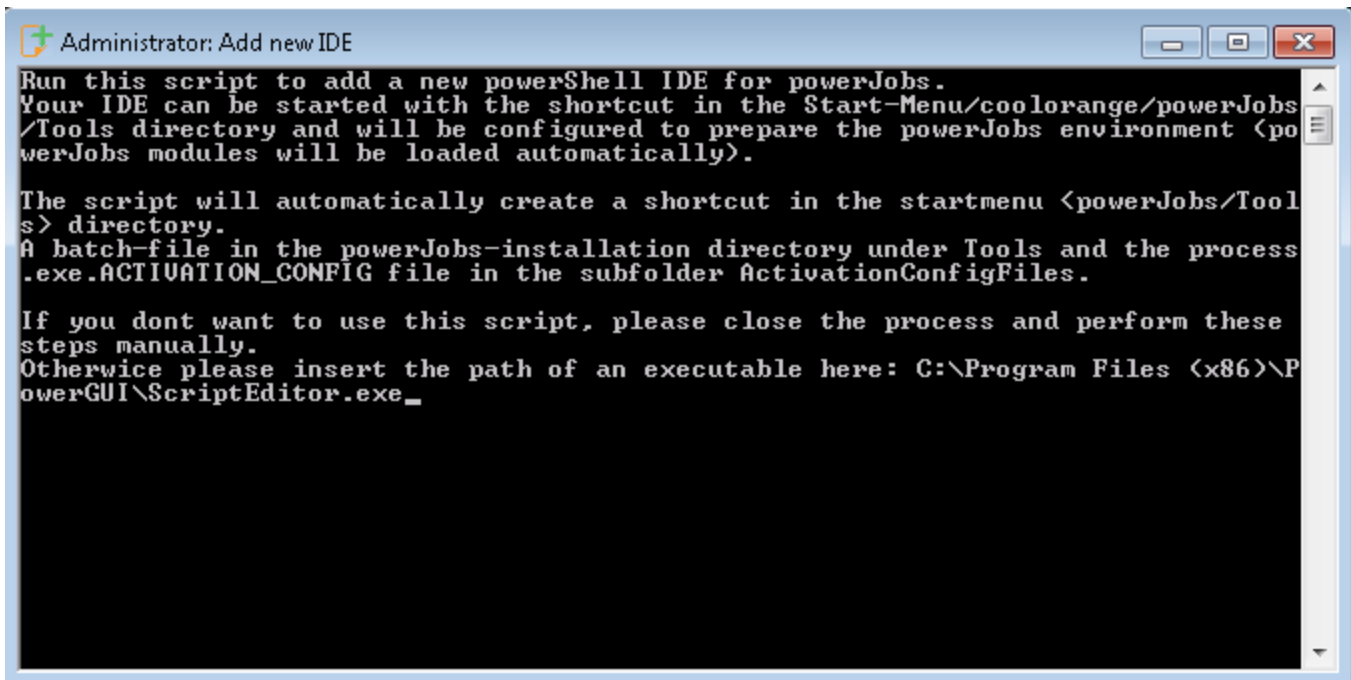


If your user has not enough rights to create files in the powerJobs installation directory under Tools, please run 'Add new IDE' as Administrator!



A PowerShell process will start and ask you for the location of your PowerShell executable.

A **powerGUI** user will enter this path: C:\Program Files (x86)\PowerGUI\ScriptEditor.exe



```
Administrator: Add new IDE

Run this script to add a new powerShell IDE for powerJobs.
Your IDE can be started with the shortcut in the Start-Menu/coolorange/powerJobs
/Tools directory and will be configured to prepare the powerJobs environment (po
werJobs modules will be loaded automatically).

The script will automatically create a shortcut in the startmenu <powerJobs/Tool
s> directory.
A batch-file in the powerJobs-installation directory under Tools and the process
.exe.ACTIVATION_CONFIG file in the subfolder ActivationConfigFiles.

If you dont want to use this script, please close the process and perform these
steps manually.
Otherwise please insert the path of an executable here: C:\Program Files (x86)\P
owerGUI\ScriptEditor.exe_
```

Press enter and wait for completion.

Now you will find a new shortcut called ScriptEditor in your start menu where the other shortcuts are located.

A new batch-file will be created in the Tools directory called ScriptEditor.bat and for the executable you can find an Activation-config-file here:

C:\Program Files\coolOrange\powerJobs 2015\Tools\ActivationConfigFiles\ScriptEditor.exe.ACTIVATION_CONFIG

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**.

Structure of a powerJobs script

- [Preperation](#)
- [Execution](#)

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 a [vaultconnection](#) should be established.

```
Open-VaultConnection
```

"**PrepareEnvironmentForFile**" returns a powerJobs file object of the latest iteration of your file. 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 the filename passed to the function.

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



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.

```
Open-VaultConnection -Vault:"Vault" -Server:"2015-SV-12-RT" -User:"Administrator"
-Password:""
$file = PrepareEnvironmentForFile "Catch Assembly.iam"
$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.Name + ".pdf" #define here the file name for the generated PDF
$inventorExtensions = @("idw","dwg","iam","ipt")
#endregion
# limit publishing to inventor files
$ext = $file.'File Extension'
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");
}
```

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

Terminating scripts with an error

If you want your script to be terminated with an error you should use powershell's **throw** function. With `ErrorActionPreference = "Stop"` it will terminate the script and shows an error in your jobqueue as well as in the logfile.

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

Terminating scripts without error

If you don't want an error entry in your jobqueue for a specific scenario you should log the error with the **\$powerJobs.log** object and then terminate your script with powershell's **exit** or **break** function. Exit will allways close the current powershell console. Break is a bit different. If it gets called in a function the function is stopped. If it gets called in a loop powershell breaks out of the loop. If you call it in your mainscript it behaves exactly like **exit**.

```
if($validFileTypes -notcontains $ext){
    $powerJobs.Log.Warn("Filetype $ext is not supported")
    exit
}
```

Additional logging

If you want to log **INFO**, **ERROR**, **WARNING** or **DEBUG** information you should use the **\$powerJobs.log** object. It makes an entry in the powerJobs logfile and doesn't terminate the script.

```
$powerJobs.Log.Info("Starting Job: ExampleJob")
```

Adding jobs

- [Adding jobs to the directory](#)
- [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.



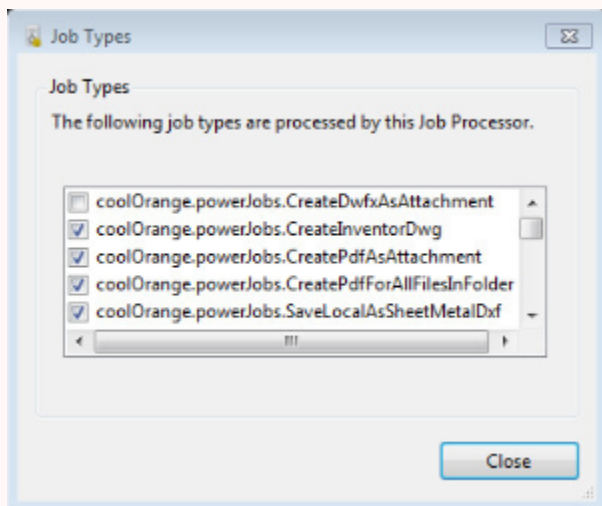
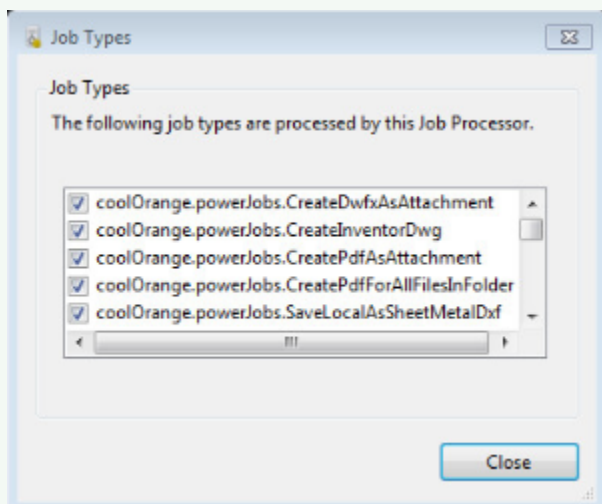
If you add a job while the PowerJobsProcessor is running you have to restart it.



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

Verify that the job gets found

Start your PowerJobsProcessor. 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 PowerJobsProcessor 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_MODULES_DIR

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

```
cd env:
dir
```

powerJobs file object

What is the powerJobs file object?

The powerJobs file object represents the latest iteration of the actual file in vault. It's made to provide the same information you can see, when you look at the file in the vault client.

Why another file object?

The vault vdf knows two file objects. The webservice file object and the file iteration object. Both have their purposes but they share the same limitations. The powerJobs file object fills this gap.

What is different?

Other than the vdf file objects the powerJobs file object has members for things like **lifecycle state**, **category** and most importantly **properties**. So if you want to know e.g. the Author of a file you just have to write `$file.author`. You don't have to care about Ids or file iterations. You always work with the latest iteration and the powerJobs file object is an exact representation of the file in vault. This saves a lot of code, because you don't have to get lifecycle objects or property objects and work through them with loops.

Creating a powerJobs file object

The powerJobs file object is created with the powerVault Cmdlets **Add-VaultFile**, **Get-Vaultfile** and **Get-VaultFiles**. `PrepareEnvironmentForFile` also uses these cmdlets and returns a powerJobs file object.

Structure of the powerJobs file object

The powerJobs file object is dynamically generated based on your files properties. The properties are named the same as in vault including whitespaces. If you want to access such a property you have to enclose it in single quotes. E.g. `$file.'full path'`.

The data types of the object and the vault file are converted automatically.

Vault	powershell
Number	long
Text	string
Thumbnail	ThumbnailType
Boolean	bool

When not to use the powerJobs file object?

If you have very special use cases, where our commandlets are not flexible enough, you have to use functions from the VDF or directly the webservices.

In that case, you can just convert the powerJobs file into an Autodesk file by using the Id or MasterId property.

Customizing the file conversion

- Altering the existing functionality
 - Valid destination file types
 - Valid source file types
 - Using other programs for file conversion
 - Post processing files right after creation
- Example: Adding additional output formats
 - Adding file types to the Save-FileAs function
 - Adding the helper function
 - Using the new function

Altering the existing functionality

Open the coolOrange.powerJobs.Publish.psm1 file in a powerShell [IDE](#).

Valid destination file types

The first cmdlet you find is **Save-FileAs**.

In this function is defined what target formats are valid for file conversion. For each format a subfunction is called that does the real work. For example if you take a look at the format PDF yousee that the helper function **SaveAs_PDF** is called.

```

function Save-FileAs{
param(
    [Parameter(Mandatory=$True,Position=1)]
    [PSCustomObject]$file,
    [Parameter(Mandatory=$True,Position=2)]
    [string] $localDestFile
)
$fromExtension = $file.($SYSPROPS.Extension).ToUpper()
$global:publishResult = $false;
Create-Directory $localDestFile
Remove-File $localDestFile
Try {
    For-TheFormat '.PDF' -ofFile $localDestFile {
        $global:publishResult = SaveAs_PDF $file $localDestFile
    }
    For-EachFormat @('.DWF','.DWFX') -ofFile $localDestFile {
        $global:publishResult = SaveAs_DWF $file $localDestFile
    }
    For-TheFormat '.DXF' -ofFile $localDestFile {
        $global:publishResult = SaveAs_DXF $file $localDestFile
    }
    For-TheFormat '.DWG' -ofFile $localDestFile {
        $global:publishResult = SaveAs_DWG $file $localDestFile
    }
    For-TheFormat '.GIF' -ofFile $localDestFile {
        $global:publishResult = SaveAs_GIF $file $localDestFile
    }
    For-EachFormat @('.IGS','.IGES') -ofFile $localDestFile {
        $global:publishResult = SaveAs_IGES $file $localDestFile
    }
    For-EachFormat @('.STP','.STEP') -ofFile $localDestFile {
        $global:publishResult = SaveAs_STEP $file $localDestFile
    }
    For-EachFormat @('.JPG','.BMP','.TIFF','.TIF','.PNG') -ofFile $localDestFile {
        $global:publishResult = SaveAs_OtherFormat $file $localDestFile
    }
} Catch
{
    $powerJobs.Log.Error("Failed to publish the file", $_.Exception )
}
return $global:publishResult
}

```

Valid source file types

To figure out for which file types this function can be called you have to scroll down to that function. The array passed to the **Is-SupportedForFormat** function contains the valid file extensions. If you want to allow more formats you have to add them to the array.



It has to be technically possible to do the conversion.

Using other programs for file conversion

If you want to use other software for the file conversion you have to replace the Publish-VaultFile function with your own. For example use a PostScript plotter to create a PS file then use GhostScript to translate the PS file into a PDF.


```
function SaveAs_PDF($file, $localDestFile)
{
    Is-SupportedForFormats $file @('IAM','IPT','IDW','DWG')
    return Publish-VaultFile -File $file -ToFile $localDestFile -Format 'PDF'
    -UseTrueView
}
```

Post processing files right after creation

If you want to use the existing technique to create PDFs, but you want to do some post processing like placing your companies logo as a [watermark](#) then you should do it in the actual job after **Save-FileAs** finished successful.

Example: Adding additional output formats

This example shows based on the formats JPG, BMP, TIFF, TIF, PNG how to add additional output formats to the Save-FileAs function.

Adding file types to the Save-FileAs function

In the Save-FileAs function you have to create a new section for your new file type/s. Just copy one of the others and modify it.

```
...
Try {
    ...
    For-EachFormat @('.JPG','.BMP','.TIFF','.TIF','.PNG') -ofFile $localDestFile {
        $global:publishResult = SaveAs_OtherFormat $file $localDestFile
    }
}
Catch{
    $powerJobs.Log.Error("Failed to publish the file", $_.Exception );
}
return $global:publishResult
}
```

Adding the helper function

It is important to name the function exactly the same as in the Save-FileAs function or vice versa.

```
function SaveAs_DGN($file, $localDestFile){

}
```

Define for which formats this function should work and implement your the actual conversion functionality.

```
SaveAs_DGN($file, $localDestFile){
    Is-SupportedForFormats $file @('DWG')
    INSERT CONVERSION FUNCTIONALITY HERE
}
```



Inventor and AutoCAD are capable of a lot more than we implemented. You can add a lot of other filetypes by copying our functions and slightly altering them.

Using the new function

The new function can be used like every other one. You just have to use the new file type in the output path.



You have to restart an already running powershell session in order for the changes to take effect.

```
$file = Get-VaultFile -File "$/AutoCAD/HelloWorld.dwg"
Save-FileAs $file "C:\Temp\HelloWorld.dgn"
```

Tutorials

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

Adding the file revision to the PDF name

The following code returns the revisionlabel of the file. The revisionlabel is what you see in Vault. E.g. "A" or "C"

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

$file.revision
```

You can add the revisionlabel to your filename string like this

```
$PDFfileName = $file.EntityName + "$($file.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 powerfull 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](#)
- [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 [pdfModule](#). Also you will need the [fileSystemModule](#) and the [vaultExModule](#).

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 -dNOSAfer -dPDFa -dUseCIEColor
-sProcessColor=DeviceCMYK $convertedparam $original
cd $oldloc.Path
}
```

Calling the function

You can call the function like this:

```
$sourcefile = "C:\TEMP\original.pdf"
$destfile = "C:\TEMP\conv.pdf"
Convert-PDF $sourcefile $destfile
```



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

Automated Conversion for existing PDFs

```

$file = PrepareEnvironmentForFile "Catch Assembly.iam.pdf"
$powerJobs.Log.Info("Starting job 'Pdf Conversion' ...")

#region PDF Settings
$errorActionPreference = "Stop"
$sourcePath = "C:\TEMP\orig\"
$destPath = "C:\TEMP\PDF_A\"
$pdfFileName = $file.Name
$sourceFilePath = $sourcePath + $pdfFileName
$destFilePath = $destPath + $pdfFileName
$validExtensions = @("pdf")
$overrideNewerPDF = $true
#endregion

New-Folder -path $sourcePath
New-Folder -path $destPath

Download-VaultFile -file $file -downloadPath $sourcePath
try{
    if(!$overrideNewerPDF){
        if([System.IO.File]::Exists($destFilePath)){
            if(Compare-FileDate -sourcefile $sourceFilePath -destfile $destFilePath
            -filename $pdfFileName){
                throw("Destfile:$(($destFilePath)) is newer than
Source:$(($sourceFilePath))")
            }
        }
    }
    if( $validExtensions -notcontains $file.'File Extension' ){
        throw("$(($ext)) is not a valid filetype")
    }
    Convert-PDF -sourcefile $sourceFilePath -destfile $destFilePath
    #Do something with the new PDF. Copy, Move, Check-In etc
}
finally{
    Remove-Item -Force $sourceFilePath
    #Uncomment the next line to delete $destFilePath. It`s commended out for testing
    purposes
    #Remove-Item -Force $destFilePath
}

```

> **Expand source**

```

fu
    param (
        [System.IDisposable] $inputObject = $(throw "The parameter -inputObject is
required."),
        [ScriptBlock] $scriptBlock = $(throw "The parameter -scriptBlock is
required.")
    )

    Try {
        &$scriptBlock
    }
    Finally {

```

```

        if ($inputObject.psbase -eq $null) {
            $inputObject.Dispose()
        } else {
            $inputObject.psbase.Dispose()
        }
    }
}

function Add-WatermarkToPdf($localDestFile, $path_text){
    Do-ForeachPage $localDestFile {
        PARAM(
            [int]$pageNumber,
            [iTextSharp.text.pdf.PdfReader]$reader,
            [iTextSharp.text.pdf.PdfStamper] $stamper
        )

        $pdfSize = $reader.GetPageSizeWithRotation($pageNumber)
        $underContent = $stamper.GetUnderContent($pageNumber)
        if( Test-Path $path_text)
        {
            Pdf-InsertImage $overContent $path_text $pdfSize
        }
        else
        {
            Pdf-InsertText $overContent $path_text $pdfSize
        }
    }
}

function Add-StampToPdf($localDestFile , $path_text){
    Do-ForeachPage $localDestFile {
        PARAM(
            [int]$pageNumber,
            [iTextSharp.text.pdf.PdfReader]$reader,
            [iTextSharp.text.pdf.PdfStamper] $stamper
        )

        $pdfSize = $reader.GetPageSizeWithRotation($pageNumber)
        $overContent = $stamper.GetOverContent($pageNumber)

        Pdf-CreateOverLayer $overContent $pdfSize

        if( Test-Path $path_text)
        {
            Pdf-InsertImage $overContent $path_text $pdfSize
        }
        else
        {
            $overContent.SetRGBColorFill(0, 0, 0)
            Pdf-InsertText $overContent $path_text $pdfSize
        }
    }
}

function Pdf-CreateOverLayer($overContent, $pdfSize){
    $padding = 2.0
    $overContent.SetRGBColorFill(0, 0, 0)
    $overContent.MoveTo(0 + $padding, 0 + $padding)
    $overContent.LineTo(0 + $padding, $pdfSize.Top - $padding)
    $overContent.LineTo($pdfSize.Right - $padding, $pdfSize.Top - $padding)
    $overContent.LineTo($pdfSize.Right - $padding, 0 + $padding)
    $overContent.ClosePath();
}

```

```

function Pdf-InsertImage($overContent, $imagePath, $pdfSize){
    $image = [iTextSharp.text.Image]::GetInstance($imagePath)
    $image.SetAbsolutePosition($pdfSize.Width / 2 - $image.Width/2, $pdfSize.Height /
2-$image.Height/2)
    $overContent.AddImage($image)
}
function Pdf-InsertText($overContent, $text, $pdfSize ){
    $baseFont =
[iTextSharp.text.pdf.BaseFont]::CreateFont([iTextSharp.text.pdf.BaseFont]::HELVETICA,[
System.Text.Encoding]::ASCII.EncodingName,"false")
    $fontSize = 120.0
    $fcolor = New-Object iTextSharp.text.BaseColor (252,175,33)
    #calculating the angel of the stamp with trigometrics the angel is around 45° in
an A4 page
    $textAngle = ([System.Math]::Atan2($pdfSize.Height, $pdfSize.Width) +
(180/[System.Math]::PI))*1.0

    $underContent.BeginText()
    $underContent.SetFontAndSize($baseFont,$fontSize)
    $underContent.setColorFill($fcolor);
    $underContent.ShowTextAligned([iTextSharp.text.pdf.PdfContentByte]::ALIGN_CENTER,
$text, $pdfSize.Width / 2, $pdfSize.Height / 2, $textAngle)
    $underContent.EndText()
}
function Do-ForeachPage($localDestFile, [ScriptBlock] $operation){
    [System.Reflection.Assembly]::LoadFrom("C:\Temp\itextsharp.dll")
    $file = New-Object System.IO.FileInfo $localDestFile
    PSUsing ( $mStream = New-Object System.IO.MemoryStream ) {
        PSUsing ( $reader = New-Object iTextSharp.text.pdf.PdfReader $file.FullName )
    {
        PSUsing ( $stamper = New-Object iTextSharp.text.pdf.PdfStamper ($reader,
$mStream) ) {
            for ($pageNumber = 1; $pageNumber -le $reader.NumberOfPages;
$pageNumber ++ )
            {
                $operation.Invoke($pageNumber, $reader, $stamper)
            }
        }
        PSUsing ( $fileStream = [System.IO.File]::OpenWrite($file.FullName) ) {
            $fileStream.Write($mStream.ToArray(), 0, $mStream.ToArray().Length)
        }
    }
}
function Format-PdfToPdfalb($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 -dNOSAFTER -dPDFA
-dUseCIEColor -sProcessColor=DeviceCMYK $convertedparam $original
    cd $oldloc.Path
}
function Convert-PDF{
    Param(
        [Parameter(Mandatory=$True,Position=1)]
        [String]$sourcefile,
        [Parameter(Mandatory=$True,Position=2)]

```

```
[String]$destfile
)

try{
    #Converting PDF
    $gserror = Format-PdfToPdfalb -original $sourcefile -converted $destfile
}
catch{
```

```

        $powerJobs.Log.Error("PDF conversion failed: Ghostscript error: $($gserror)")
    }
}

```

> **Expand source**

```

function Compare-FileDate{
    Param(
        [Parameter(Mandatory=$True,Position=0)]
        [String]$path
    )

    if(!(Test-Path -Pathtype Container -Path $path)){
        New-Item -Path $path -ItemType Directory
    }
}

function Compare-FileDate{
    Param(
        [Parameter(Mandatory=$True,Position=0)]
        [PSCustomObject]$sourceLoc,
        [Parameter(Mandatory=$True,Position=1)]
        [String]$destLoc
    )
    $dest = Get-ChildItem -Path $destLoc
    return !($dest.LastWriteTime -gt $source.CheckInDate)
}

```

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/>

How to handle different 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\cool\Orange\powerJobs\Modules\Publish\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\coolOrange\powerJobs\Modules\Publish\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 the file object I call the script to change the template. `$file.'Paper Size'` returns the value of the Vault Property 'Paper Size'. More about the [file object](#).

```
$file = PrepareEnvironmentForFile "DEMO01.dwg" $true
Change-TemplateTo $file.'Paper Size'
```

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
)
$path = [System.IO.Path]::Combine($env:POWERJOBS_MODULESDIR, "Publish")
$setupDwg = [System.IO.Path]::Combine($path, "TrueViewSetup.dwg")
$template = [System.IO.Path]::Combine($path, $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")
}
}
```

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 is to identify a convenient 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. Additionally you may have a DWG and IDW associated to the same item. So, the logic to pick the right 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 before **PrepareEnvironmentForFile**, but after **Import-Module "\$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"**

```
$itemID = $powerJobs.Job.Params["ItemId"]
if(!$itemID){
    $powerJobs.Log.Warn("itemId is not set")
    exit 1
}
$fileItemAssociations = $vault.ItemService.GetItemFileAssociationsByItemIds(
@($itemID),
[Autodesk.Connectivity.WebServices.ItemFileLnkTypOpt]::Primary -bor
[Autodesk.Connectivity.WebServices.ItemFileLnkTypOpt]::Secondary -bor
[Autodesk.Connectivity.WebServices.ItemFileLnkTypOpt]::Tertiary
)
$drawings = $fileItemAssociations | Where-Object { $_.FileName -like '*.idw' -or
$_.FileName -like '*.dwg' }
if($drawings -is [System.Array]){
    $primaryDrawing = $drawings | Where-Object { $_.Typ -eq 'Primary' }
    if($primaryDrawing -ne $null){
        $fileId = $drawings[0].CldFileId
    }
}
else{
    $fileId = $drawings.CldFileId
}
if(!$fileId){
    $item = $vault.ItemService.GetItemsByIds(@($itemID));
    $powerJobs.Log.Warn("No drawing found for item "+$item.SyncRoot[0].ItemNum)
    return 1;
}
$powerJobs.Job.Params.Add("FileId", "$($fileId)")
```

FAQ

How to get a filename without extension?

```
$fileName =  
[System.IO.Path]::GetFileNameWithoutExtension("C:\EXAMPLE1\Example2\Example.dwg")
```

Where is the documentation for the Vault API?

Question

Where is the documentation for the Vault API?

Answer

First you have to install the Autodesk Vault SDK. You can find the installer in "**C:\Program Files\Autodesk\Vault Professional 2015\SDK**".

Afterwards you will find the documentation where you installed the sdk.

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

Where is the powerJobs2013 documentation?



powerJobs2013 documentation

From now on powerJobs2013 documentation is available as Pdf only. You can download it here: [Part1](#) [Part2](#)

Troubleshooting

- [Logging Levels](#)
- [DLL not found in Powergui](#)
- [Jobprocessor freezes](#)
- [The powerJobs button doesn't show up in the menu](#)
- [powerVault deadlocks in powershell](#)
- [AutoCAD COM exceptions](#)
- [powerJobs doesn't start with Vault Workgroup](#)
- [Startup-Operation: 'RestrictionChecks' failed!](#)
- [PDF is created for the wrong file version](#)
- [DWG/DXF Export of 2D Inventor files uses the wrong configuration 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 2015\Extensions\coolOrange.PowerJobs.Handler\

In the lines

```
<filter type="log4net.Filter.LevelRangeFilter">  
<levelMin value="WARN" />  
<levelMax value="FATAL" />  
</filter>
```

you can configure the logginglevel. You could set the minimal filter level to "DEBUG", than all the levels between the range Debug and Fatal will be logged.

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

The powerJobs button doesn't show up in the menu

▼ Symptoms

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

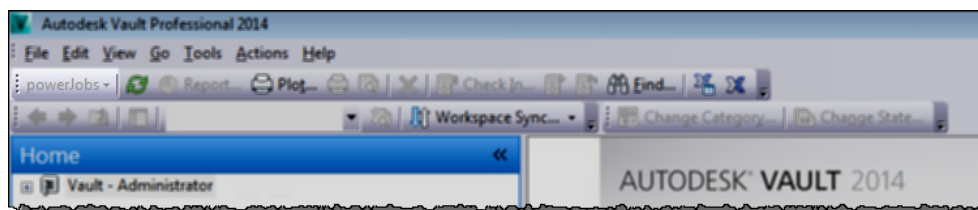
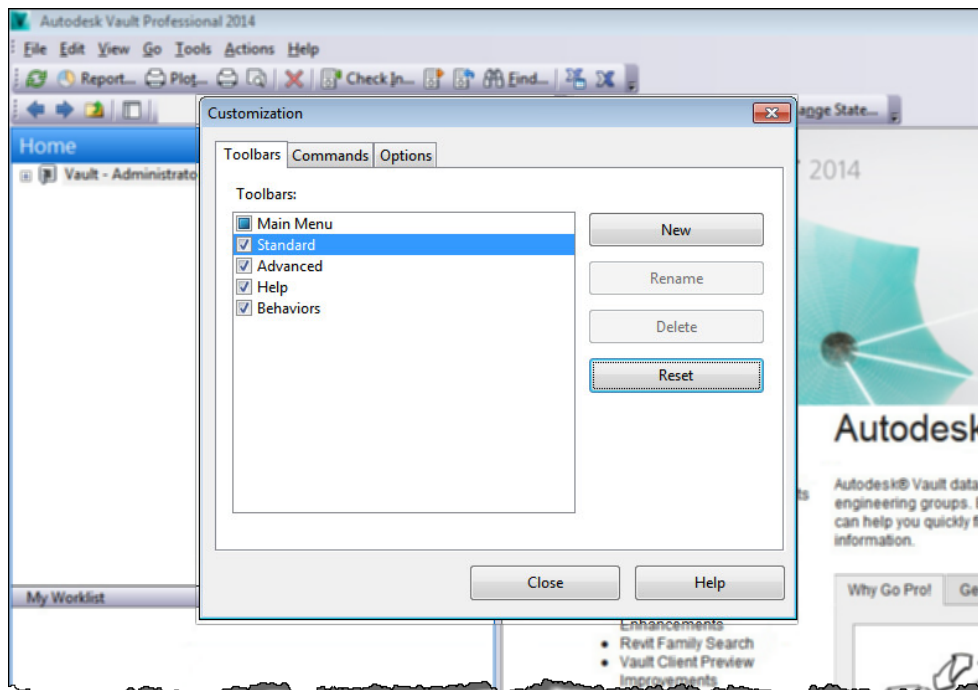
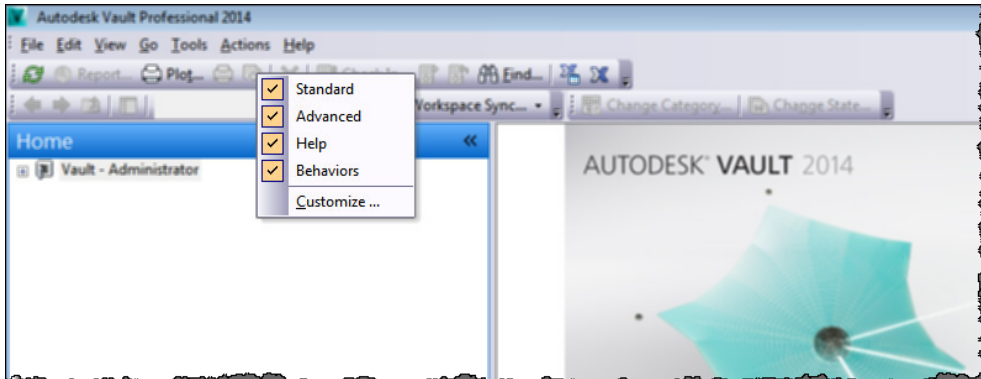
▼ Cause

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](#).

powerVault deadlocks in powershell

Are you using cmdlets from powerVault like e.g. Add-VaultFile and this function seems to run into an endless loop?

If you are using e.g. PowerShell.exe or another application without Userinterface than it could happen that you find this [Autodesk Bug](#).

A quick and simple solution, is to start PowerShell.exe out from the powerJobs Tools directory: read [IDEs for powershell](#).

If you want to use powerVault in an IDE that is not started via the shortcuts in the Tools folder, you can choose one of the next solutions:

The solution that Autodesk provides for this is the following:

As early as possible in your Job Handler's implementation of IJobHandler.Execute, add the following line:
`Autodesk.DataManagement.Client.Framework.Library.Initialize(false);`

This means you have to call this fix in your PowerShell like this:

```
[Autodesk.DataManagement.Client.Framework.Library]::Initialize($false)
```

Another simple possibility, if you are using the PowerShell IDE to perform debugging on your scripts would be to use the PowerShell ISE that has a UserInterface (or other IDE's like PowerGUI Script Editor)

Other appearances of this bug

Usually after the deadlock all the functions that are using internally functionality of the VDF are crashing with similar error messages:

```

System.Threading.Tasks.TaskCanceledException: A task was canceled.
  at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)
  at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Ta
sk task)
  at System.Windows.Threading.DispatcherOperation.Wait(TimeSpan timeout)
  at System.Windows.Threading.Dispatcher.InvokeImpl(DispatcherOperation operation,
CancellationToken cancellationToken, TimeSpan timeout)
  at System.Windows.Threading.Dispatcher.Invoke[TResult](Func`1 callback,
DispatcherPriority priority, CancellationToken cancellationToken, TimeSpan timeout)
  at System.Windows.Threading.Dispatcher.Invoke[TResult](Func`1 callback)
  at
Autodesk.DataManagement.Client.Framework.Internal.ImageUtils.GetFileSystemImage(String
fileName, ShellIconSize size, String key)
  at
Autodesk.DataManagement.Client.Framework.Vault.Internal.PropertyExtensions.FilePropert
yExtensions.PostGetPropertyValues(Connection vltConn, IEnumerable`1 entities,
PropertyDefinitionDictionary propDefs, PropertyValues resultValues,
PropertyValueSettings settings)
  at
Autodesk.DataManagement.Client.Framework.Vault.Services.Connection.Implementation.Prop
ertyManager.getPropertyValuesByEntityClass(String eclassId, IEnumerable`1 entities,
IEnumerable`1 propDefs, PropertyValueSettings settings)
  at
Autodesk.DataManagement.Client.Framework.Vault.Services.Connection.Implementation.Prop
ertyManager.getPropertyValues(String eclassName, IEnumerable`1 entities, IEnumerable`1
propDefsToRetrieve, PropertyValueSettings settings, BackgroundWorker worker,
LoadPropertiesThreadRequestArgs workerArgs)
  at
Autodesk.DataManagement.Client.Framework.Vault.Services.Connection.Implementation.Prop
ertyManager.GetPropertyValues(IEnumerable`1 entities, IEnumerable`1
propDefsToRetrieve, PropertyValueSettings settings)

```

In the powerShell.exe you will see the error like this:

AutoCAD COM exceptions

The message filter indicated that the application is busy

This can appear at several places when using [AutoCAD engine](#) in the JobProcessor. Here is an example of this exception:

```

2014-06-10 07:14:14,785 [Pipeline Execution Thread] ERROR . - An error occurred during
exiting AutoCAD Application
System.Runtime.InteropServices.COMException (0x8001010A): The message filter indicated
that the application is busy. (Exception from HRESULT: 0x8001010A
(RPC_E_SERVERCALL_RETRYLATER))
  at Autodesk.AutoCAD.Interop.IAcadApplication.Quit()
  at ..Stop()

```

Details

To prevent this problem we introduced an IMessageFilter for a better communication between the .Net environment and the COM process. IMessageFilters are working only in STA appartements, therefore publishing with AutoCAD works perfectly when using powerGUI or powerShell

ISE.

powerShell is running by default as MTA, and the JobProcessor.DelegateHost (where powerJobs is running inside) always runs as MTA thread. The powerShell can be launched as STA like this:

```
start "powerJobs STA" "powershell.exe" "-STA"
```

When powerShell is launched as STA, the IMessageFilter can take affect and helps to coordinate the communication between AutoCAD and powerJobs.

Solutions

At the moment AutoCAD engine works stable only in STA appartements, but not in MTA threads. The JobProcessor is running as MTA, and therefore we will work on fixing this problem as soon as possible.

Retrieving the COM class factory for component with CLSID {0B628DE4-07AD-4284-81CA-5B439F67C5E6} failed

This can happen when your machine is very busy, or when AutoCAD is not registered. Here is an example of this exception:

```
[ ERROR] - Failed to start AutoCADSystem.Runtime.InteropServices.COMException
(0x80080005): Retrieving the COM class factory for component with CLSID
{0B628DE4-07AD-4284-81CA-5B439F67C5E6} failed due to the following error: 80080005
Server execution failed (Exception from HRESULT: 0x80080005
(CO_E_SERVER_EXEC_FAILURE)).
    at System.RuntimeTypeHandle.CreateInstance(RuntimeType type, Boolean publicOnly,
Boolean noCheck, Boolean& canBeCached, RuntimeMethodHandleInternal& ctor, Boolean&
bNeedSecurityCheck)
    at System.RuntimeType.CreateInstanceSlow(Boolean publicOnly, Boolean skipCheckThis,
Boolean fillCache, StackCrawlMark& stackMark)
    at System.RuntimeType.CreateInstanceDefaultCtor(Boolean publicOnly, Boolean
skipCheckThis, Boolean fillCache, StackCrawlMark& stackMark)
    at System.Activator.CreateInstance(Type type, Boolean nonPublic)
    at ..CreateApplication()
```

Details

Usually we faced this issue when running AutoCAD on virtual machines that are very slow. But it could appear also if the JobProcessor machine is very busy with other things.

Solutions

If AutoCad was not started before, please start AutoCad or any flavour of AutoCad the first time on your machine. Then AutoCad will be registered corretly and the problem should be fixed

powerJobs doesn't start with Vault Workgroup


Issue

powerJobs won't start on Vault Workgroup machines.

Cause

Currently there is an issue if only Vault Workgroup is installed. The Registrykey of Vault Workgroup is not recognized only the one of Vault Professional.

Solution

 Fixed in 15.1.67

Add the following key to your registry. Make sure you have a Backup of your registry before you make any changes to it.

In "Executable" you have to enter the location of your Vault Workgroup executable, instead of the Vault Professional. In order to add the key to the registry save the code in a textfile and rename it to *.reg and execute it.

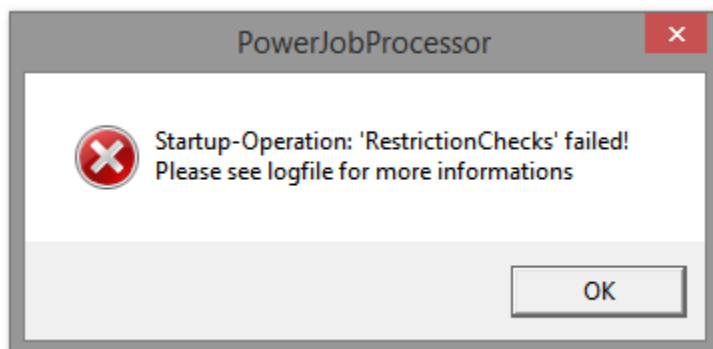
```
Windows Registry Editor Version 5.00
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Autodesk\PLM\Autodesk Vault Professional 19.0]  
"Executable"="C:\\Program Files\\Autodesk\\Vault Professional  
2015\\Explorer\\Connectivity.VaultPro.exe"  
@=" "
```

Startup-Oeration: 'RestrictionChecks' failed!

▼ Symptoms

On startup of powerJobs the following error appears.
Startup-Oeration: 'RestrictionChecks' failed! Please see logfile for more informations



In the logfile something like this is written

Path: 'C:\ProgramData\Autodesk\Vault 2014\Extensions\coolOrange.PowerJobs.Handler\PowerJobs.vcet.config' Requires rights: 'Write', 'WriteAttributes', 'WriteExtendedAttributes' and 'Append'.

▼ Cause

Unknown

▼ Resolution

Copy the file 'C:\ProgramData\Autodesk\Vault 2015\Extensions\coolOrange.PowerJobs.Handler\PowerJobs.vcet.config' to 'C:\ProgramData\Autodesk\Vault 2014\Extensions\coolOrange.PowerJobs.Handler\PowerJobs.vcet.config'

▼ More information

PDF is created for the wrong file version

▼ Symptoms

You are using the synchronize properties job and the revision table in your PDFs is outdated.

▼ Cause

The synchronize property job creates a new file iteration, but the PDF job is still pointing to the previous file iteration.

▼ Resolution

Get the latest file iteration in your PDF job. (coolOrange.powerJobs.CreatePdfAsAttachment.ps1)

Insert the following line of code before the settings part of the script.

```
$file = Get-VaultFile -FileId $file.MasterId
```

▼ More information

DWG/DXF Export of 2D Inventor files uses the wrong configuration file

▼ Symptoms

The configuration file that is configured in your job isn't used for the file conversion.

▼ Cause

Inventor saves the last used configuration file and will use it again. This has priority over powerJobs.

▼ Resolution

- Start Inventor
- Open a Drawing
- SaveCopyAs
- Select DXF as file type
- Select Options
- Configure the export how you fancy
- On the second screen choose to save the configuration
- Ok
- Create the DXF somewhere

From now on this configuration file should be used for DXF conversion. You can do the same for DWG and other formats.

Change logs

Current Version of powerJobs 2015 is 15.2.66

Latest Change log

Release

16.03.2015

General

powerJobs has now seperated powerVault

All Change logs

- [15.0.32](#)
- [15.1.49](#)
- [15.1.67](#)
- [15.1.73](#)
- [15.1.87](#)
- [15.2.6](#)
- [15.2.41](#)
- [15.2.66](#)

15.0.32

Release

25.04.2014

General

- This is the initial release of powerJobs 2015
- All psm1 files in the powerShell modules folder will be loaded automatically from now on.

Fixes

- PDFcreation fails by changing the state of a file from "work in progress" to "released"
- createPDF from Folder: throws exception if no idw or dwg is found
- Apply PowerShell profile failes
- handle CLR 4 exception of powershell

15.1.49

Release

13.06.2014

Features

- support other export formats
- Checking windows rights
- Using AutoCAD to generate PDFs
- trigger jobs to certain time
- Cmdlet for file download
- dock trace and inspect window to jobserver
- powerGate as part of powerJobs
- having our own startup exe
- setup for powerJobs 15.1
- sample job: ReleasedDocuments with trigger
- job already exists in the queue, exception 237
- Execution Policy not set
- powershell.exe cannot execute CLR4 assemblies
- PowerShell IDE from powerJobs tools directory

General

- Better logging in logwindow
- computer not added to domain but always errors in log

Fixes

- sample scripts for german vaults
- Console notification: Inc -> s.r.l.
- Add-Log same logs to powerJobProcessor not working
- when powerJobsProcessor closes, exit jobProcessor delegate host
- Logging im coolOrange Job Processor nicht ausreichend
- logging is not all visible
- RestrictionCheck failes with exceptions
- Nullpointer when not using AutoCAD publisher
- Error logging after creating files by using Autocad
- Support Copy to Clipboard for Trace Window
- upgrade: powerShell profile contains two times import-Module cmd
- title name of powerJobs executable shall be
- powerJobs log file name
- Install Locations in Wiki contains wrong path
- wiki.coolorange.com/powerjobs says page not found
- During Setup it's possible to change installation path for Vault extension
- inventor restarted after every Job
- when stored login credentials from Jobprocessor becomes invalid -> strange workaround to get PowerJobProcessor running again
- Default logfile log-level set to WARN instead ERROR
- Windows 8.1 - Restriction Checks do not work correctly

15.1.67

Release

07.08.2014

Fixes

- powerJobs should now work with Vault Workgroup
- Fixed some stability issues

15.1.73

Release

17.09.2014

Features

- standard jobs support multiple languages

General

- multilingual vault support

Fixes

- Pdf generation uses the latest revision of the file
- TrueView as standard for AutoCAD files
- removed TrueViewSetup-files
- removed default PDF generation on lifecycle transition

15.1.87

Release

24.2.2015

Fixes

The Environment-Variables are set correctly after first installation

The cmdlet "Save-FileAs" works now also on a not english vault for Dxf generation

PDF's with more versions are attached correctly to the corresponding file versions

15.2.6

Release

10.10.2014

Features

supporting Vault 2015 R2

15.2.41

Release

24.02.2015

Fixes

The Environment-Variables are set correctly after first installation

The cmdlet "Save-FileAs" works now also on a not english vault for Dxf generation

PDF's with more versions are attached correctly to the corresponding file versions

15.2.66

Release

16.03.2015

General

powerJobs has now seperated powerVault

What's new in powerJobs2015

- powerJobs file **object**.
- powerVault **Cmdlets**.
- Two new developer tools. The **Trace Window** and the **Inspector**.
- Less and more consistent code. With the powerVault Cmdlets we made the first step to cleaner and better maintainable code. I.e. You have to spend less time migrating your jobs to new versions and looking for bugs.
- Load powerJobs **automatically** when you open your ide.
- Alternatively Now you can use **AutoCAD** instead of DwgTrueView in your jobs
- Export your CAD files into many **new file formats**
- powerJobs has a fancy new **user interface**
- It isn't necessary anymore to edit configuration files in order to **add new jobs** to the job processor
- Now it's possible to **trigger jobs automatically** at a certain time and date
- by starting **powerShell IDE's** from our Tools you get a perfectly configured session is started, ready for working with powerJobs
- The powerJobs - **powerGate integration** can be installed together with **coolOrange powerGate**