



---

## coolOrange Wiki

- [06. Customization](#)
- [Anatomy of a powerJobs script](#)
- [Code snippets](#)
- [Environment Variables](#)
- [Error Handling](#)
- [IDEs for powershell](#)
- [Modules](#)
- [PDF on item lifecycle change](#)
- [PowerShell scripts and modules](#)
- [The PowerJobs Objects](#)
- [2013](#)
- [2014](#)
- [powershell general](#)
- [Create .csv from BOM](#)
- [Create DWG from an IDW](#)
- [Create PDF from IDW](#)
- [Create textfile via template](#)
- [Create visible or invisible attachments](#)
- [Data to XML](#)
- [PDF in an external folder](#)
- [Print via Inventor](#)
- [Release via jobserver](#)
- [Retrieve the user that queued the job](#)
- [Selected files to ZIP](#)
- [Send email via jobserver](#)
- [Create .csv from BOM](#)
- [Create DWG from an IDW](#)

- [Create textfile via template](#)
- [Data to XML](#)
- [PDF in an external folder](#)
- [Print via Inventor](#)
- [Release via jobserver](#)
- [Retrieve the user that queued the job](#)
- [Selected files to ZIP](#)
- [Send email via jobserver](#)
- [Copy file in a directory](#)
- [Insert into SQL server](#)
- [Print/convert Office documents](#)
- [Set default printer](#)
- [Simple document print on default printer](#)
- [07. Patchnotes](#)
- [08. Troubleshooting](#)
- [09. FAQ](#)
- [Convert PDF to DWF](#)
- [How can I add a watermark/stamp to pdf?](#)
- [How can I add the revision to the name of the pdf-file?](#)
- [How can I convert pdf to pdf/a?](#)
- [How can I create a pdf with corresponding properties from original file?](#)
- [How can I create a PDF with same category, revision, state as the original file?](#)
- [Where's documentation for the Vault API?](#)



---

## 06. Customization

---

### Overview

This section contains everything about customizing powerJobs.

---

### Related

Topics

#### [Code snippets](#)

Here you can find some code snippets and samples.

---

#### [IDEs for powershell](#)

Describes in detail some options you have when choosing an IDE for working with powershell scripts.

---

#### [PowerShell scripts and modules](#)

Instead of coding your Job in languages like C# - with the need of compiling the code and deploying the assemblies - powerJobs uses the flexibility of PowerShell scripts that can be modified at any time – without the need to compile it. powerJobs interprets the ps1 files when the Job Processor executes your job.

---

Tutorials

- [Anatomy of a powerJobs script](#) (Beginner)
- [PDF on item lifecycle change](#) (Beginner)

References

- [Environment Variables](#)
- [Error Handling](#)
- [Modules](#)
- [The PowerJobs Objects](#)





---

## Anatomy of a powerJobs script

1. [Overview](#)
2. [Details](#)
3. [Structure](#)
  - 3.1. [2013](#)
  - 3.2. [2014](#)
4. [What's Next](#)
5. [Related](#)

---

### Overview

In this section we analyze the typical building blocks of a powerJobs script.

---

### Details

The anatomy of a powerJobs script is very simple and clean, there is a preparation and a working script part. All gets also explained on the basis of the example\_script script and the included comments.

---

### Structure

---

#### 2013

In the preparation part the script reads all the parameters and saves them, if necessary, in variables. Then the powerJobs script catalyzes the important information out of the previously saved variables, afterwards it controls, if all information and parameters have regular values. Is that case the script will start with its proper function.

The first if clause has only a function if you are using it for debugging, because you won't need the manual loading of the dll or the manual login to the vault, when you are running the job with powerJobs, also you won't create the vault object by yourself. This part has only a function for debugging purposes.



```
if(!$IAmRunningInJobProcessor){Import-Module
"$env:POWERJOBS_MODULES_DIR\coolOrange.PowerJobs.VaultHelper.psm1"#
möchte[System.reflection.Assembly]::LoadFrom("$env:POWERJOBS_DLL")$vault = New-Object
-type coolOrange.PowerJobs.VaultProxy$vault.Login("Administrator","", "localhost", "Vault")# get
the file$file = $vault.GetUniqueLatestFileByFilename("Catch Assembly.idw")if(!$file) { throw ("File
cannot be found") }}else{
```

This is now the standard preparation of the script in the case you are running it with powerJobs.

```
# get the file$fileID = $vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }$file =
$vault.DocService.GetFilesByFileId($fileID)}# get the latest version of the file in case a sync prop has
been executed before the job$file = $vault.DocService.GetLatestFileByMasterId($file.MasterId)#
in the settings are some basic variables defined to get an easy customization of the script#region
Settings$showPDF = $true #change this setting to $true or $false for showing/hiding the
PDF$PDFfileName = $file.Name + ".pdf" #define here the file name for the generated
PDF#endregion$ext = [System.IO.Path]::GetExtension($file.Name)# here gets the file extension
compared with the allowed file typesif( ($ext -eq ".idw") -or ($ext -eq ".dwg") -or ($ext -eq
".iam") -or ($ext -eq ".ipt")){
```

After the preparation the script starts with its real task. Here the script can do everything you want, for example print documents, send emails to specific persons etc. Our example script will publish the chosen file as a pdf file. There are more sample jobs, listed in the sample job topic.

```
# publish (generate the pdf attachment)$localDestFile = "C:\TEMP\" + $PDFfileName# here you
can chose your generated file type with the argument$publisher=$vault.GetPublisher("PDF")#
select the destination of the newfile$publisher.OutputFile=$localDestFile# create the new pdf file
and save it$publisher.Publish($file.Id)}
```

The vault objects has some methods and members. They are listed in the \$vault object topic.

## 2014

powerJobs 2014 has a very nice way for preparing the debug environment. Just call this line in your debugger and you will have access to the \$vault, \$vaultConnection, \$vaultExplorerUtil and \$powerJobs:

### PrepareEnvironment

On top of your job script you should import the VaultHelper module to have access to well prepared debug functionalities like:

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

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

This function helps you to get the right file for your environment (Debug or JobProcessor). If you are running inside of JobProcessor it takes the file from the \$powerJobs.Job parameter. If you are running in



debug-mode, it searches the file by the passed in filename. The last parameter tells if you want to have the latest version of the file.

For easier debugging you have now the ability to use the \$powerJobs.Log property. You have direct access to the powerJobs logging (configured in the log4net configuration file).

---

## What's Next

This is what was achieved and what was omitted in this tutorial.

---

## Related

- [06. Customization](#)





---

## Code snippets

1. [Overview](#)
2. [INFO](#)
3. [Related](#)

---

## Overview

Here you can find some code snippets and samples.

---

## INFO

---

**The code examples ("The samples") were created to the best of our knowledge. The samples' main purpose is for learning and getting new ideas and are not guaranteed to conform to any programming style, standard, or be an adequate answer for any given problem.**

---

**You are free to use, extend and distribute the samples, and you are solely responsible for it's use and performance.**

---

**coolOrange is not responsible for data loss, hardware damage, or disaster related to the use of the samples.**

---

---

## Related

### [2013](#)

Here you can find some code snippets and samples.

---

### [2014](#)

Here you can find some code snippets and samples.

---



## [powershell general](#)

This section is for general powershell scripts, that aren't related to any coolOrange product. ....







---

## Environment Variables

1. [Overview](#)
2. [Details](#)

---

### Overview

Gives a short overview of the powerJobs specific environment variables.

---

### Details

PowerJobs sets, when it gets installed, 3 environment variables. They contain only paths to folder of the jobs, modules and the powerJobs.dll. powerJobs for example loads the powerJobs.dll by itself, but if you want to debug your job you need also the dll, now you can easily access the dll by the environment variable POWERJOBS\_DLL.

Thats the script command you will use to load it manually:

```
| System.reflection.Assembly]::LoadFrom($env:POWERJOBS_DLL)
```

The 3 environment variables:

- POWERJOBS\_DLL
- POWERJOBS\_JOBSDIR
- POWERJOBS\_MODULES\_DIR

You can search the environment variables also by yourself. Open first the PowerShell and then write "cd env:", then in the next row write "dir", now all environment variables of your system should be listed.





---

## Error Handling

1. [Overview](#)
2. [Details](#)

---

### Overview

Describes how error handling works in a powerJobs script.

---

### Details

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.

The following table lists the various options for the \$ErrorActionPreference:

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.

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

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

You can also use the numeric value like this:

```
$ErrorActionPreference = 2
```

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

To get more information about the log file please read the Activate Logging topic.





---

## IDEs for powershell

1. [Overview](#)
2. [Details](#)
3. [Powershell 2.0 ISE](#)
  - 3.1. [2014](#)
4. [PowerGUI](#)
  - 4.1. [2014](#)
5. [Related](#)

---

## Overview

Describes in detail some options you have when choosing an IDE for working with powershell scripts.

---

## Details

Although you can create and edit powershell scripts with a simple tool as Microsoft's notepad that doesn't mean you should. It is much more convenient to use a powershell development environment, and fortunately there are some very good ones for free.

---

## Powershell 2.0 ISE

With PowerShell 2.0 you get a free development environment, called PowerShell ISE. On Windows 7 this is already part of the operating system, on older systems you get the ISE when you install PowerShell 2.0.

### 32 bit Version

If you are working on a 64bit system, you must make sure that you use the 32 bit version of the PowerShell ISE. It can be found at C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell\_ise.exe.



If you happen to use the 64 bit version and try to run or debug a powerJobs script in the ISE you will get an error.

### Support for .net 4

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

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

Then save and copy the file besides the powershell\_ise.exe.

---

## 2014

In general it depends on the Vault clients target platform. If your Vault-Client is 64 bit, use a 64 bit PowerShell ISE. If your Vault client is 32, you have to use a 32 bit PowerShell ISE.

---

## PowerGUI

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

### 32 bit Version

If you are working on a 64bit system, you must make sure that you use the 32 bit version of the PowerGUI tools. The PowerGUI setup creates separate entries for the x86 tools on start menu group during installation.

---

## 2014

Also for powerGUI you have to use the 64bit version if you have a 64bit VaultClient installed on your machine. If your Vault client is 32bit, you have to use the 32bit powerGUI.

---

## Related

- [06. Customization](#)





---

## Modules

1. [Overview](#)
2. [Explanation](#)
3. [Details about powerJobs modules](#)
4. [Sample Modules](#)
  - 4.1. [2014](#)

---

## Overview

Here we give a description of the powershell modules concept as it is used by powerJobs.

---

## Explanation

PowerShell modules provide an efficient, manageable and production-oriented way to package code. For example, if you create in one job for powerJobs a clean and working function for sending e-mails with a pdf attachment, you don't want to copy/paste the function every time, you need it in a new job. So just save it in a module and import it every time you need the same email function in different jobs. If you are programmer, you can say a module is very similar to a class of C#, perhaps a module can be more with the cmdlets , just keep that explanation in mind.

---

## Details about powerJobs modules

In powerJobs the moduls are stored under this path: C:\ProgramData\coolOrange\powerJobs\Modules

When powerJobs is started, it loads all modules from the previously mentioned folder in its environment and provides them for every job. This way the programmers don't have to think about importing and removing powerJob modules.



---

## Sample Modules

Two modules are already provided when you buy powerJobs.

The first one is coolOrange.powerJobs.CadHelper.psm1, it functions to check if all necessary software is installed.

The second one is coolOrange.powerJobs.VaultHelper.psm1, it includes the following functions to get an easy and clean access to the vault folders/files:

- Add-VaultFile
- Add-VaultDesignVizualizationFile
- Add-VaultDesignVizualizationFile
- Get-CheckoutVaultFile
- Get-CheckinVaultFile
- Get-VaultFileAssociations
- Get-VaultFolder

---

## 2014

For a very easy starting of debugging, you can use the function: PrepareEnvironment (loads the powerJobs environment and creates all the relevant powerJobs variables automatically for you in your Debug environment). You have not to load the Autodesk assemblies manually, because powerJobs will find the Vault Client installation and use the relevant assemblies from that directory.

This two functions are using the prepareEnvironment. They are creating also the \$file and the \$folder object if needet:

- PrepareEnvironmentForFile
- PrepareEnvironmentForFolder

Very often you need Generic Lists and Generic Dictionaries in combination with the Vault API. Now we give you the ability to create this generics with this functions:

- New-GenericList
- New-GenericDictionary

And because you can work now also with the new VDF, you can find some converter functions to convert the File object in a VDF.File (and Folder):

- ToVdfFile



- ToVdfFolder







---

## PDF on item lifecycle change

1. [Overview](#)
  2. [Goal](#)
    - 2.1. [First Step](#)
    - 2.2. [Second Step](#)
  3. [What's Next](#)
  4. [Related](#)
- 

### Overview

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

---

### Goal

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

---

### First Step

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



---

## Second Step

Now we know that the function for getting the files associated to an item is called `GetItemFileAssociationsByItemIds`. Unfortunately the drawings associated to an item, could be of primary link if there is no model associated, or tertiary link if there is a model associated. Additoinal you may have a DWG and IDW associated to the same item. So, the logic to pick the 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.

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

Now, in the standard PowerShell script for the PDF creation (`CreatePDFasAttachment.ps1`), quite at the beginning you will find the following lines

```
$fileID = $vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }
```

Here you can see that in case the `$fileID` is not set, then an exception will be thrown. This is the case if the job gets triggered from an object which is not a file, like in our case an item. By inserting the code above between the 2 lines, the code will search for a file, and if given it will set the `$fileID` with the ID of the according file. If there is no file, then the job will quit with success and make a entry in the log file. As not every item has a file or drawing associated, in case where no drawing can be found, we just silently quit.

---

## What's Next

You may have to test your different scenarios and potentially adapt the code to your particular needs. But overall, this code should cover the tipical scenarios.

---

## Related

- [06. Customization](#)





---

## PowerShell scripts and modules

1. [Overview](#)
2. [Technology](#)
  - 2.1. [Modules](#)
  - 2.2. [2014](#)
  - 2.3. [Vendors](#)
  - 2.4. [Background and History](#)
3. [Related](#)

---

## Overview

Instead of coding your Job in languages like C# - with the need of compiling the code and deploying the assemblies - powerJobs uses the flexibility of PowerShell scripts that can be modified at any time – without the need to compile it.

powerJobs interprets the ps1 files when the Job Processor executes your job.

---

## Technology

In order to recognize a ps1 file as a Job for the Autodesk Job Processor the file has to be located in the folder:

C:\ProgramData\coolOrange\powerJobs\Jobs

and the powerJobs.vcet.config file has to be edited as described in chapter Adding Jobs. During startup of the job processor all the ps1 files' names are evaluated. If you add or remove a job with a ps1 file you need to restart the job processor. If you edit a ps1 file while the job processor is running, you only need to save the ps1 file. The job will detect the changes when the job is executed the next time and will reload the ps1 file.



---

## Modules

Modules are sets of functions which help you writing your ps1 scripts and Jobs. The code in the modules is visible to all the ps1 script files. Modules have the file extension psm1:

In order to load a psm1 module to the context of your job the file has to be located in the folder:

%ProgramData%\Autodesk\Vault 2012\Extensions\coolOrange.PowerJobs.Handler\Modules

The module files are loaded during startup of the job processor. If you make changes to the module files (i.e. add, edit, delete a file) you need to restart the job processor. Development, Testing We recommend 2 tools for development of PowerShell scripts. The first is the PowerShell ISE which is part of the PowerShell 2.0 installation. An even better choice is the PowerGUI script editor which is part of the PowerGUI product. You can download PowerGUI for free from <http://www.powergui.org> Both ISE and PowerGUI have very useful interactive debuggers you can use to test your powerJobs scripts. When working with these tools it is important to use the 32 bit environments when you are working on an OS with 64bits. For PowerGUI you need to use the ScriptEditor\_x86.exe or the AdminConsole\_x86.exe, both located in the install directory of PowerGUI. For the PowerShell ISE you find the 32bit version in C:\Windows\SysWOW64\WindowsPowerShell\v1.0

To access the vault from a job, powerJobs creates a global variable called \$vault. If the job script is running within powerJobs in the job processor this variable exists and it represents the connection to the 'vault world' When you are running the script from within a standalone development environment no such connection exists. You need to create this manually. The Vault Proxy variable (\$vault) is not set if your script is not executed in the context of the Autodesk Job Processor. For testing purposes you can load the proxy and make a login to Vault manually:

```
if(!$vault) {[System.Reflection.Assembly]::LoadFrom("C:\Program Files (x86)\Autodesk\Vault Professional 2011\Explorer\coolOrange.PowerJobs.dll")[coolOrange.PowerJobs.VaultProxy]::Login("Administrator","", "localhost")
= New-Object -Type coolOrange.PowerJobs.VaultProxy}
```

Before executing code in PowerGUI or ISE you should first start the vault client and connect it to the vault so that all vault services are up and running and initialized. Otherwise you may get connection problems when connecting from the debugger the first time. In a job executing in the job processor you would get the object id of the vault object (file, item etc.) from the \$vault like so:

```
$fileId = $vault.Job.Params["FileId"]
```

In case of executing in the debugger you must provide the \$fileId information by retrieving it from the vault, e.g. like so:

```
$coFile=$vault.GetUniqueLatestFileByFilename("cross head sub.dwg")
```

```
$fileId=$coFile.Id
```

A typical vault script header block would then look like this:

```
if(!$vault) {[System.reflection.Assembly]::LoadFrom("C:\Program Files (x86)\Autodesk\Vault Professional 2011\Explorer\coolorange.powerJobs.dll")[coolOrange.PowerJobs.VaultProxy]::Login("Administrator","", "localhost")
= New-Object -type coolOrange.PowerJobs.VaultProxy
$coFile=$vault.GetUniqueLatestFileByFilename("cross head sub.dwg")$fileId=$coFile.Id}else{#
get the fileif(!$fileId){$fileId = $vault.Job.Params["FileId"]}}
```

---

## 2014

In order to load a psm1 module to the context of your job the file has to be located in the folder:

%ProgramData%\coolorange\powerJobs\Modules

The module files are loaded during startup of the job processor.



To access the vault from a job, powerJobs creates the global variables called \$vault,\$vaultConnection,\$vaultExplorerUtil and \$powerJobs. If the job script is running within powerJobs in the job processor this variable exists and it represents this:

- \$vault = WebserviceManager
- \$vaultConnection = Connection object from the VDF
- \$vaultExplorerUtil = IExplorerUtil
- \$powerJobs = contains the Job object, the Logging and other helpful functions for searching files,items and creating PDF,DWFX,...

When you are running the script from within a standalone development environment no such connection exists. You need to call this two lines that will create this variables for you:

```
[System.reflection.Assembly]::LoadFrom("$env:POWERJOBS_DLL")  
[coolOrange.PowerJobs.PowerShellVaultProxy]::Instance.Login(USERNAME,PASSWORD,SERVER,VAULT)
```

Please check that the logincredentials to your vault are correct!  
All the variables are created automatically in your powerShell environment.

---

## Vendors

PowerShell is a Microsoft technology that comes standard with any Windows 7 and other Windows operating systems.

---

## Background and History

To know more about Windows PowerShell read this link [http://en.wikipedia.org/wiki/Windows\\_PowerShell](http://en.wikipedia.org/wiki/Windows_PowerShell).

---

## Related

- [06. Customization](#)





---

## The PowerJobs Objects

1. [Overview](#)
2. [Details](#)
3. [Methods of \\$powerJobs](#)
  - 3.1. [GetPublisher](#)
    - 3.1.1. [Publisher attributes:](#)
  - 3.2. [add\\_OnBeginPublish](#)
  - 3.3. [DownloadDependentFiles](#)
    - 3.3.1. [2014](#)
  - 3.4. [DownloadDependentFiles](#)
  - 3.5. [GetUniqueLatestFileByFilename](#)
  - 3.6. [GetLatestItemByNumber](#)
4. [Members](#)
  - 4.1. [2014](#)
5. [Related](#)

---

## Overview

Describes in detail the available settings and controls for a feature in the product.

---

## Details

Offer the details a user needs to know about the definition, parameters, and so on. Tables are extremely useful for looking up information and organizing the details that readers want to know.

---

## Methods of \$powerJobs



---

## GetPublisher

GetPublisher(string publishformat)

### Description:

Gets the publisher for the choosen filetype

### Parameter:

PDF

DWF

DWFX

SheetMetalDXF

### Publisher attributes:

**bool** IgnoreCheckOutCheck

### Description:

If value is set to ' 1 ' there will be no check if the file is checked out or not.

### Values:

0;1

**string** Options

### Description:

In this string you can define arguments for ' .dxf-files '. The options are seperated by ' & ' and you can assign multiple values by seperating them with ' ; '.

### Values:

This is part of the inventor-api. For a full description take a look at the inventor-api help under

HTML/DataIO\_Sample.htm

**Translate - Sheet Metal to DXF API Sample**

**string** OutputFile



**Description:**

Sets Filepath and Filename + Extension for the published file

**Values:**

path+filename+extension

`int` GeneratorEnginePublishCount

**Description:**

The number of publishingprocesses till inventor will be restarted. Keep in mind that higher values are resulting in worse performance.

**Values:**

Any integer value. Standard is 10.

`bool` Publish(fileID)

**Description:**

Publishs a file

**Parameter:**

`long` fileID

`bool` Open(fileID)

**Description:**

Opens a File without publishing it

**Parameter:**

`long` fileID





---

## add\_OnBeginPublish

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

### Description:

This Method lets you make changes to the visualization document, before creating the final version of it.

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

If the publisher is working with TrueView, the \$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.

### Parameter:

`$publisher`: The instance of the publisher, the event is beeing called from.

`$document`: The document, the publisher is working with.

---

## DownloadDependentFiles

```
DownloadDependentFiles(vaultFileId)
```

### Description:

Loads the specified file into a temporary directory. You have to delete the temporary file yourself afterwards.

### Parameter:

`long` vaultFieldID

### 2014

This function is removed from the API. The VDF provides allready functionality for this:



## DownloadDependentFiles

DownloadDependentFiles(vaultFileId, targetDirectory)

### Description:

Same as DownloadDependentFiles but you can declare the temporary directory

### Parameter:

`long` vaultFieldID; `string` directorypath

---

## GetUniqueLatestFileByFilename

GetUniqueLatestFileByFilename(string fname)

### Description:

Gets the filepath of the latest file with the chosen name

### Parameter:

`string` filename

### Returns:

VaultFile

---

## GetLatestItemByNumber

GetLatestItemByNumber(string number)

### Description:

Gets the latest Item with the chosen itemnumber

### Parameter:

`string` Itemnumber

### Returns:

VaultItem

---



---

## Members

With these members you can use the respective Vault-API functions in your powershellscripts.

<b>Vault API</b>	<b>powerJobs</b>
AdminService	AdmService
BehaviorService	BehService
CategoryService	CatService
ChangeOrderService	CoService
DocumentService	DocService
DocumentServiceExtensions	DocExtService
ForumService	ForumService
InformationService	InfoService
ItemService	ItmService
JobService	JobService
KnowledgeVaultService	KvService
PackageService	PackgService
PropertyService	PropService
RevisionService	RevService



SecurityService	SecService
IExplorerUtil	ExplorerUtil

---

## 2014

In powerJobs 2014 we have introduced \$vault (WebserviceManager), \$vaultConnection (VDF Connection) and \$vaultExplorerUtil (IExplorerUtils).

Because of this keep attention when upgrading from powerJobs 2013 to 2014, because a.e. \$vault.AdmService is now called like in the real WebserviceManager: AdminService.

\$vault.ExplorerUtil is now the \$vaultExplorerUtil variable!

---

## Related

- [06. Customization](#)





---

## 2013

1. [Overview](#)
2. [Related](#)

---

## Overview

Here you can find some code snippets and samples.

---

## Related

- [Create DWG from an IDW](#) (Beginner)
- [Create PDF from IDW](#) (Beginner)
- [Create visible or invisible attachments](#) (Beginner)
- [PDF in an external folder](#) (Beginner)
- [Print via Inventor](#) (Intermediate)
- [Release via jobserver](#) (Intermediate)
- [Retrieve the user that queued the job](#) (Intermediate)
- [Send email via jobserver](#) (Intermediate)
- [Create .csv from BOM](#) (Advanced)
- [Create textfile via template](#) (Advanced)
- [Data to XML](#) (Advanced)
- [Selected files to ZIP](#) (Advanced)





---

## 2014

1. [Overview](#)
2. [Related](#)

---

## Overview

Here you can find some code snippets and samples.

---

## Related

- [Create DWG from an IDW](#) (Beginner)
- [PDF in an external folder](#) (Beginner)
- [Print via Inventor](#) (Intermediate)
- [Release via jobserver](#) (Intermediate)
- [Retrieve the user that queued the job](#) (Intermediate)
- [Send email via jobserver](#) (Intermediate)
- [Create .csv from BOM](#) (Advanced)
- [Create textfile via template](#) (Advanced)
- [Data to XML](#) (Advanced)
- [Selected files to ZIP](#) (Advanced)





---

## powershell general

1. [Overview](#)
2. [Related](#)

---

### Overview

This section is for general powershell scripts, that aren't related to any coolOrange product.

---

### Related

- [Copy file in a directory](#) (Beginner)
- [Insert into SQL server](#) (Beginner)
- [Set default printer](#) (Beginner)
- [Simple document print on default printer](#) (Beginner)
- [Print/convert Office documents](#) (Intermediate)





---

## Create .csv from BOM

1. [Overview](#)
  2. [Code](#)
    - 2.1. [Related](#)
- 

### Overview

This tutorial gives you an example for creating a csv-file from the BOM via Powershell

---

### Code

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called BOM_to_Excel.ps1, and save it into the powerJobs folder. #Edit the
JobProcessor.exe.config to declare your new job. For queueing the job, you might use the
LifecycleEventEditor and configure your job on a given lifecycle
change.if(!$IamRunningInJobProcessor){ Import-Module
"$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
[System.Reflection.Assembly]::LoadFrom("$env:POWERJOBS_DLL") $vault = New-Object -type
coolOrange.PowerJobs.VaultProxy $vault.Login("Administrator","", "localhost", "Vault") # get
the file $file = $vault.GetUniqueLatestFileByFilename("Pad Lock.iam") if(!$file) { throw ("File
cannot be found")} }else{ # get the file $fileID = $vault.Job.Params["FileId"] if(!$fileID) {
throw ("File ID not set")} $file = $vault.DocService.GetFileById($fileID)} #gets the items with
the file-id $items = $vault.ItmService.GetItemsByFileId($file.Id) #the item-masterid which you
need to get the bom $itemMaster = $items[0] if (!$items) {throw ("File has no item")} #Library to
use specific functions [System.Reflection.Assembly]::LoadFrom("C:\Program Files
(x86)\Autodesk\Autodesk Vault 2013 SDK\bin\Autodesk.Connectivity.WebServices.dll") #Gets the
BOM-file from the vault $itemBOM =
$vault.ItmService.GetItemBOMByItemIdAndDate($itemMaster.Id, [System.DateTime]::MinValue, [Autodesk.C
variables to get specific BOM informations $boms = $itemBOM.ItemRevArray $bomsOcc =
$itemBOM.OccurArray $bomsAss = $itemBOM.ItemAssocArray #Gets LifeCycleStates of the
items $DispName = @{} $defs = $vault.ItmService.GetAllLifeCycleDefinitions() foreach ($def in
```

[http://wiki.coolorange.com/powerJobs/06.\\_Customization/Code\\_snippets/powershell\\_general](http://wiki.coolorange.com/powerJobs/06._Customization/Code_snippets/powershell_general)

Updated: Wed, 26 Feb 2014 10:18:54 GMT

Powered by mindtouch™





```

$defs){ $DispName[$def.Id]=$def.DispName}#Gets the Quantity of the items$quantity =
@()for($i = 0;$i -lt $boms.Count;$i+=1){ for($j = 0;$j -lt $bomsAss.Count;$j+=1){
if($boms[$i].MasterID -eq $bomsAss[$j].CldItemMasterID){ $quantity +=
$bomsAss[$j].CldItemUsage } } }#Write here the path where the csv-file should be
placed$FilePath = "C:\<YourFolder>\BOM$(($file.Name).csv"#Writes with the PS function
"out-file" the column names into
.csv-file"Number`tDetail_ID`tQuantity`tTitle`tRevision`tState`tUnits`n"|out-file
$FilePath#Writes with the PS function "out-file" the informations into .csv-file for($i = 0;$i -lt
$boms.Count;$i+=1){
"$($boms[$i].ItemNum)`t$($bomsOcc[$i].Val)`t$($quantity[$i])`t$($boms[$i].Title)`t$($boms[$i].RevNum
out-file $FilePath -Append }

```

---

## Related

- [2013](#)





---

## Create DWG from an IDW

1. [Overview](#)
2. [\\_](#)
  - 2.1. [Code](#)
3. [Related](#)

---

### Overview

As DWG is quite popular, you might want to create a DWG from an IDW via jobserver.

At line 24 you can change it also to other formats (.stp,...) that you want to create.

---

---

### Code

```
# get the file via jobserver$fileID = $vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }$file = $vault.DocService.GetFileById($fileID)# Limits the files which get published$ext = [System.IO.Path]::GetExtension($file.Name)# In this case, only .idw -files get publishedif($ext.ToLower().Equals(".idw")){# The .idw-file gets created in the follow path:$localDestFile = "C:\TEMP\" + $file.Name + ".pdf"# The file then gets published into a pdf$publisher=$vault.GetPublisher("PDF")$publisher.OutputFile=$localDestFile# Eventhandler in which you can create other file formats$publisher.add_OnBeginPublish({param($publisher, $document)$compDef = $document.Document.ComponentDefinition# The pdf file get saved as a .dwg file$document.Document.SaveAs("C:\TEMP\" + ($file.Name) + ".dwg", $true)})# The Eventhandler gets calledif(!$publisher.Open($File.Id)){throw "The .dwg-translation failed!"}}
```

---

### Related

- [2013](#)





---

## Create PDF from IDW

1. [Overview](#)
2. [Code](#)
  - 2.1. [Related](#)

---

### Overview

create a PDF file from an IDW file next to the original.

---

### Code

```
# get the file via jobserver$fileID = $vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }$file = $vault.DocService.GetFileById($fileID)#Limits the files which get published$ext = [System.IO.Path]::GetExtension($file.Name)#In this case, only .idw -files get publishedif($ext.ToLower().Equals(".idw")){#Path for the PDF-file$localDestFile = "C:\TEMP\" + $file.Name + ".pdf"#The file gets published into a pdf$publisher=$vault.GetPublisher("PDF")$publisher.OutputFile=$localDestFile#The Eventhandler gets calledif(!$publisher.Publish($File.Id)){throw "The PDF-translation failed!"}}
```

---

### Related

- [2013](#)





---

## Create textfile via template

1. [Overview](#)
2. [Goal](#)
  - 2.1. [First Step](#)
  - 2.2. [Example with an Assembly1.idw file:](#)
3. [Related](#)

---

### Overview

Explains how to complete a self-paced learning exercise using a feature in the product.

---

### Goal

Supposing you have to write out several information into a file, and you like to keep the file format flexible, the idea could be to use a template file to drive the format.

#### Steps

---

### First Step

To use the script, create a file called "template.txt" (use the exact path in the script). Then open the "template.txt" and write the property names from which you might write informations out into a file. Use this Syntax: the propertynames have to be written in "{...}", the properties gets seperated by ";".

**Example:** **{Name};**This is my Classification: **{Classification};**It got created by: **{Created By};**. The format is flexible. You can also define a html-page: `<html><table border="1"><tr><td>{Name};</td><td>{Classification};</td><td>{Created By};</td></tr></table></html>`

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



```

configure your job on a given lifecycle change.if($vault -eq
$null){[System.reflection.Assembly]::LoadFrom($env:POWERJOBS_DLL)[coolOrange.PowerJobs.VaultProxy]
= New-Object -type coolOrange.PowerJobs.VaultProxy}#Gets the file$fileID =
$vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }$file =
$vault.DocService.GetFileById($fileID)#Paths: #TEMPLATE-PATH$pathTemp =
"C:\<YourFolder>\template.txt"$FilePath =
"C:\<YourFolder>\file.txt"#Propertydefinitions$propDefs =
$vault.PropService.GetPropertyDefinitionsByEntityClassId("FILE")#Writes the
Propertydefinitionids in a System.Array to use a specific method$ids = @()foreach($f in
$propDefs){$ids+=$f.id}#gets the file-properties$props =
$vault.PropService.GetProperties("FILE",@($file.id),$ids)#Define RegularExpression, to get the
propertynames from the template-file$regex = [regex]"{\{+(?)\b[A-Z\s]+\b\}"$template =
Get-Content -Path $pathTemp$header = $regex.Matches($template)#Get the
valuesforeach($match in $header){$literalFieldName =
$match.Value.TrimStart("{").TrimEnd("}")$propId = 0foreach($pd in
$propDefs){if($pd.DisplayName.Equals($literalFieldName)){ $propId=$pd.Id}}if($propId -gt
0){foreach($pi in $props){if($pi.PropDefId.Equals($propId)){ $template =
$template.Replace($match.Value, $pi.Val)}}}}#Writes the values in a file$template |Out-File
$FilePath -Append

```

---

## Example with an Assembly1.idw file:

template.txt:

**{Name}**;This is my Classification: **{Classification}**;It got created by: **{Created By}**;

My output file: file.txt:

**Assembly1.idw**;This is my Classification: **Design Document**;It got created by: **Administrator**;

---

## Related

- [2013](#)





---

## Create visible or invisible attachments

1. [Overview](#)
  2. [Settings](#)
  3. [Related](#)
- 

### Overview

This is a example for configuring the CreatePdfAsAttachment.ps1" script.

---

### Settings

To create attachments you can use the "coolOrange.powerJobs.CreatePdfAsAttachment.ps1" script , which gets installed with the [coolOrangePowerJobs](#) installation. If you want to make the attachment visible or invisible, open the "coolOrange.powerJobs.CreatePdfAsAttachment.ps1" file and go to line 35 and change:

```
| $showPDF= $true
```

to

```
| $showPDF= $false
```

---

### Related

- [2013](#)





---

## Data to XML

1. [Overview](#)
  2. [Code](#)
    - 2.1. [Related](#)
- 

### Overview

You may want to export information from Vault into an XML file.

---

### Code

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called Data_to_XML.ps1, and save it into the powerJobs\Jobs folder.#Edit the
JobProcessor.exe.config to declare your new job. For queueing the job, you might use the
LifecycleEventEditor and configure your job on a given lifecycle change.if($vault -eq
$null){[System.reflection.Assembly]::LoadFrom($env:POWERJOBS_DLL)[coolOrange.PowerJobs.VaultProxy]
= New-Object -type coolOrange.PowerJobs.VaultProxy}#Gets the file$fileID =
$vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }$file =
$vault.DocService.GetFilesByFileId($fileID)#Path where the .txt file and then the .xml file should be
placed$path = "C:\<YourFolder>\$(($file.Name))"#Propertydefinitions$propDefs =
$vault.PropService.GetPropertyDefinitionsByEntityClassId("FILE")#Writes the
Propertydefinitionids in a System.Array to use a specific method$ids = @()foreach($f in
$propDefs){$ids+=$f.id}#Gets the file-properties$props =
$vault.PropService.GetProperties("FILE",@($file.Id),$ids)$values = @()$names = @()#Gets the
exact name and value of the propertiesfor($i = 0; $i -lt $propDefs.Count; $i+=1){for($j = 0; $j
-lt $props.Count; $j+=1){if($propDefs[$i].Id -eq $props[$j].PropDefId -and $props[$j].ValType
-ne "Image"){ $values += $props[$j].Val$names += $propDefs[$i].DispName}}}$#Writes the
properties first in a .txt-file#"root"-tag at the beginning and the end of the xml-file is
necessary"<root>"| Out-File "$($path).txt"for ($i = 0;$i -lt $props.Length-1; $i+=1){$xmltags =
"<$(($names[$i])>$(($values[$i])</$(($names[$i])>"#The String have to be manipulated to make
a xml-file, because characters as space or parentheses are forbidden in tags$xmltags.Replace("
","").Replace("(","").Replace(")","") + "`n" | Out-File "$($path).txt" -Append}"</root>" | Out-File
"$($path).txt" -Append#Writes the .txt-file content into the xml-fileGet-Content "$($path).txt" |
Out-File "$($path).xml"#Remove the txt-fileRemove-Item "$($path).txt"
```



---

## Related

- [2013](#)







---

## PDF in an external folder

1. [Overview](#)
2. [First Step](#)
3. [Second Step](#)
4. [Related](#)

---

### Overview

A frequent request is to create a PDF and make this accessible to other systems like an ERP. Now you may know that a small script could grab the PDF from Vault and save it to the specified folder. However, as the PDF gets created via a job, why not having the job save the PDF to that location, right during the creation process.

---

### First Step

By default the '.PDF' is created locally anyway, and uploaded to Vault later. We could grab the local file and copy it to our desired location. The variable `$localDestFile` contains the location where the '.PDF' will be created.

---

### Second Step

After the successful creation, let's copy the file:

```
| Copy-Item -LiteralPath $localDestFile -Destination "c:\myFolder\"+$PDFfileName
```

to Line 51 from `coolOrange.powerJobs.CreatePdfAsAttachment.ps1`

---

### Related

- [2013](#)





---

## Print via Inventor

1. [Overview](#)
  2. [Code](#)
    - 2.1. [Related](#)
- 

### Overview

In order to print a file via Inventor, a similar approach to the one for creating a DXF can be taken. In this case we just open the file, catch the open event and then trigger the print. For printing, the PrintManager from Inventor is used. This allows us to be quite flexible on the printing requirements. Here is the code

---

### Code

```
# get the file$fileID = $vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }$file =
$vault.DocService.GetFileById($fileID)# get the latest version of the file in case a sync prop has
been executed before the job$file = $vault.DocService.GetLatestFileByMasterId($file.MasterId)#
limit publishing to 2d inventor files$ext =
[System.IO.Path]::GetExtension($file.Name)if($ext.ToLower().Equals(".idw")){
$publisher=$vault.GetPublisher("PDF") $publisher.add_OnBeginPublish( {
param($publisher, $document) $printManager = $document.PrintManager
$printManager.GetType().InvokeMember("Printer",[Reflection.BindingFlags]::SetProperty, $null,
$printManager, "Microsoft XPS Document Writer") $printManager.ScaleMode =
[Inventor.PrintScaleModeEnum]::kPrintBestFitScale $printManager.PrintRange =
[Inventor.PrintRangeEnum]::kPrintAllSheets $printManager.PaperSize =
[Inventor.PaperSizeEnum]::kPaperSizeA0 $printManager.SubmitPrint() })
$publisher.OutputFile="c:\temp\dummy.pdf" if (!$publisher.Open($file.Id)) { throw
("Open failed") }}
```



---

## Related

- [2013](#)





---

## Release via jobserver

1. [Overview](#)
2. [Code 1](#)
3. [Code 2](#)
  - 3.1. [\\_](#)
4. [Code 3](#)
5. [Related](#)

---

### Overview

While releasing your document, some jobserver activities should be executed. To make sure that release means also that the jobserver tasks were successfully done, it's a good idea to let the jobserver finally release the document.

---

### Code 1

If you want to change a filestate (Work in Progress, Released ...) from a masterfile you can use the follow script:

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called ChangeState.ps1, and save it into the powerJobs folder.#Edit the JobProcessor.exe.config
to declare your new job. For queueing the job, you might use the LifecycleEventEditor and
configure your job on a given lifecycle change.#get the file via jobserver$fileID =
$vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }$file =
$vault.DocService.GetFileById($fileID)#All LifeCycleDefinitions get hooked, you need them to set
a filestate$def = $vault.DocExtService.GetAllLifeCycleDefinitions()#In this case we take the
"Flexible Release Process"$FlexibleReleaseProcess= $def | Where-Object
{$_.DispName.Equals("Flexible Release Process")}$#From the "FlexibleReleaseProcess"-object you
can take your favorite state to set#in this case "Released"$releaseState =
$FlexibleReleaseProcess.StateArray | Where-Object {$_.Name.Equals("Released")}$#The
masterfilestate gets changed to
"Released"$vault.DocExtService.UpdateFileLifeCycleStates(@($file.MasterId),@($releaseState.Id),"Released
from PS")
```



---

## Code 2

If you want to change filestates (Work in Progress,Released ...) from childfiles you can use this script:

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called ChangeState.ps1, and save it into the powerJobs folder.#Edit the JobProcessor.exe.config
to declare your new job. For queueing the job, you might use the LifecycleEventEditor and
configure your job on a given lifecycle change.#get the file via jobserver$fileID =
$vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }$file =
$vault.DocService.GetFileById($fileID)#Get all childfiles$assocs =
$vault.DocService.GetFileAssociationsByIds(@($file.id),[Autodesk.Connectivity.WebServices.FileAssociationT
-ne $null){$stateIds = @()$childfileIds = @()foreach($f in $assocs[0].FileAssocs){$childfileIds
+= $f.CldFile.MasterId$stateIds += $releaseState.Id}}#The childfilesstates get changed to
"Released"$vault.DocExtService.UpdateFileLifeCycleStates($childfileIds,$stateIds,"Released from
PS")
```

---

---

## Code 3

If you want to change a filecategory (Engineering,Base...) you can use this script:

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called ChangeCategory.ps1, and save it into the powerJobs folder.#Edit the
JobProcessor.exe.config to declare your new job. For queueing the job, you might use the
LifecycleEventEditor and configure your job on a given lifecycle change.#get the file via
jobserver$fileID = $vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }$file =
$vault.DocService.GetFileById($fileID)#All category-definitions get hooked$defc =
$vault.CatService.GetCategoriesByEntityClassId("FILE",$true)#In this case we want to set the
filecategory to "Engineering"$category = $defc | Where-Object
{$_ .Name.Equals("Engineering")}#Change
Category$vault.DocExtService.UpdateFileCategories(@($file.MasterId),$category.Id,"Category
Change by PS")
```

---

## Related

- [2013](#)





---

## Retrieve the user that queued the job

1. [Goal](#)
  - 1.1. [Code](#)
  2. [Related](#)
- 

### Goal

Especially if a job has been queued via a lifecycle, you may want to know the user that queued that job

---

### Code

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called retrieveUser.ps1#In the Vault jobqueue have to be some jobs, else you cant use the
script#this code is only for debug purpose within the powershell editor and will be ignored during
the regular job executionif($vault -eq
$null){[System.reflection.Assembly]::LoadFrom($env:POWERJOBS_DLL)[coolOrange.PowerJobs.VaultProxy]
= New-Object -type coolOrange.PowerJobs.VaultProxy}#retrieve all jobs in the Vault
jobqueue$jobs = $vault.JobService.GetJobsByDate(1000,[DateTime]::MinValue)#select the job I
am (me)$currentJob = $jobs | Where-Object {$_.Id.Equals($vault.Job.Id)}#Gets user
informations from the Job-creator$userinfo =
$vault.AdmService.GetUserByUserId($currentJob.CreateUserId)#get the email address of the
user that queued the job$email = $userinfo.Email#place here you additional code#For instance,
send an E-Mail to the user
```

---

### Related

- [2013](#)





---

## Selected files to ZIP

1. [Goal](#)
  2. [Code](#)
    - 2.1. [Related](#)
- 

### Goal

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.

---

### Code

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called FilesToZip.ps1, and save it into the powerJobs folder. #Edit the JobProcessor.exe.config to
declare your new job. For queueing the job, you might use the LifecycleEventEditor and configure
your job on a given lifecycle change.if(!$IamRunningInJobProcessor) { Import-Module
"$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
[System.reflection.Assembly]::LoadFrom($env:POWERJOBS_DLL) $vault = New-Object -type
coolOrange.PowerJobs.VaultProxy $vault.Login("Administrator","", "localhost", "Vault") # get
the file $file = $vault.GetUniqueLatestFileByFilename("Catch Assembly.idw") if(!$file) { throw
("File cannot be found") }}else{ # get the file $fileID = $vault.Job.Params["FileId"]
if(!$fileID) { throw ("File ID not set") } $file = $vault.DocService.GetFileById($fileID)} #Path
for the download-files $filesPath = "C:\<YourFolder>\<YourDownloadFolder>\\" #It creates a
new folder even though it exist New-Item -Path $filesPath -Force -ItemType Directory
#Download masterfile from Vault $buffer = New-Object -type
Autodesk.Connectivity.WebServices.ByteArray $test =
$vault.DocService.DownloadFile($file.Id,$true,[ref]$buffer)
[System.IO.File]::WriteAllBytes("$filesPath$($file.Name)", $buffer.Bytes) #Load the libraries, to
use specific functions [System.Reflection.Assembly]::LoadFrom("C:\Program Files
(x86)\Autodesk\Autodesk Vault 2013 SDK\bin\Autodesk.Connectivity.WebServices.dll")
[System.Reflection.Assembly]::LoadFrom("C:\Program Files (x86)\Autodesk\Autodesk Vault 2013
SDK\bin\Autodesk.Connectivity.JobProcessor.Extensibility.dll") #Download childfiles from
Vault $assocs =
$vault.DocService.GetFileAssociationsByIds(@($file.id),[Autodesk.Connectivity.WebServices.FileAssociationT
if($assocs[0].FileAssocs -ne $null){ foreach($f in $assocs[0].FileAssocs){ #Empty
```



[http://wiki.coolorange.com/powerJobs/06.\\_Customization/Code\\_snippets/2013/](http://wiki.coolorange.com/powerJobs/06._Customization/Code_snippets/2013/)

Retrieve\_the\_user\_that\_queued\_the\_job  
Updated: Wed, 26 Feb 2014 10:18:54 GMT  
Powered by mindtouch

```

buffer      $buffer = New-Object -type Autodesk.Connectivity.WebServices.ByteArray
$vault.DocService.DownloadFile($f.CldFile.Id,$true,[ref]$buffer)
[System.IO.File]::WriteAllBytes("$filePath$($f.CldFile.Name)", $buffer.Bytes)    }    }
#Create Zip-File  $zipFileName = "C:\<YourFolder>\$($file.Name).zip"    set-content
$zipFileName ("PK" + [char]5 + [char]6 + ("$([char]0)" * 18))    $ZipFile = (new-object -com
shell.application).Namespace($zipFileName)    $fileToBeZipped = Get-ChildItem $filePath
$fileToBeZipped | ForEach-Object {    #The method .MoveHere() is running asynchronous, so we
have to wait a few seconds before moving the next file into zip    #In this case: 2 seconds
Start-Sleep -Seconds 2    #Write files into the zipfile    $ZipFile.MoveHere($_.FullName,1024) }

```

---

## Related

- [2013](#)







---

## Send email via jobserver

1. [Overview](#)
2. [Code](#)
3. [Notification via Email](#)
  - 3.1. [Related](#)

---

### Overview

sending an email at the end of a job is quite simple. The powershell language provides a ready to use command-let called send-mailmessage.

---

### Code

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

```
$fileID = $vault.Job.Params["FileId"]if(!$fileID) { throw ("File ID not set") }$file =  
$vault.DocService.GetFileById($fileID)#write here the actions that should be done before the  
email should be sentSend-MailMessage -To "user@yourdomain.com" -From  
"powerJobs@yourdomain.com" -Subject "The document $($file.Name) has been processed" -Body  
"Write here your email text and use variables to add additional information" -SmtpServer  
"yourSMTPserver"#write here the actions that should be done after the email has been sent
```

if you need to pass credentials to your SMTP server, just add the 2 lines below and add the Credential option to the send-mailmessage comandlet

```
$passwd = ConvertTo-SecureString -AsPlainText "yourPassword" -Force$cred = new-object  
Management.Automation.PSCredential "yourUser", $passwdSend-MailMessage -To  
"user@yourdomain.com" -From "powerJobs@yourdomain.com" -Subject "The document
```



```
$(file.Name) has been processed" -Body "Write here your email text and use variables to add additional information" -SmtpServer "yourSMTPserver" -Credential $cred
```

if you want to add an attachment to your email, this sample downloads the file from Vault and attached it to the mail

```
$fileContent = @()$vault.DocService.DownloadFile($file.Id, $true, [ref]$fileContent)$fileName = "c:\temp\" + $(file.Name)
set-content -value $fileContent -encoding byte -path $fileName
Send-MailMessage -To "user@yourdomain.com" -From "powerJobs@yourdomain.com" -Subject "The document $(file.Name) has been processed" -Body "Write here your email text and use variables to add additional information" -SmtpServer "yourSMTPserver" -Attachments $fileName
```

---

## Notification via Email

In order to send emails, you need a SMTP server. So, we assume this is given and you have the credentials. Then, it's an easy game. PowerShell comes with a commandlet called Send-MailMessage. Just add the given arguments like from, to, subject, body and a valid SMTP server, and you have emails sent by the Jobserver at lifecycle transitions. The content of the email can be configured as needed. Here a sample that sends an email with the file name with from and to state in the subject, and a Vault link in the body. If the user clicks on the link, Vault will start and point to the according file.

```
$lfcTransId = $vault.Job.Params["LifeCycleTransitionId"]$fileID = $vault.Job.Params["FileId"]$file = $vault.DocService.GetFileById($fileID)$folders = $vault.DocService.GetFoldersByFileMasterId($file.MasterId)$folder = $folders[0]$folderPath = $folder.FullName$folderPath = $folderPath.Replace("$", "%24").Replace("/", "%2f")$fullPath = $folderPath + "%2f" + $file.Name.Replace(" ", "+")$link = "http://localhost/AutodeskDM/Services/EntityDataCommandRequest.aspx?Vault=Vault&ObjectId="+$fullPath+"&ObjectType=File&Command=Select"
=$vault.DocExtService.GetLifeCycleStateTransitionsByIds(@($lfcTransId))$lfcs = $vault.DocExtService.GetLifeCycleStatesByIds(@($lfcTrans[0].FromId,$lfcTrans[0].ToId))$oldState = $lfcs[0].DispName$newState = $lfcs[1].DispName
Send-MailMessage -From "marco.mirandola@coolorange.com" -To "marco.mirandola@coolorange.com" -Subject "The file $(file.Name) has changed from $oldState to $newState" -Body "Dear xxx, if you like to view the related document, just follow this link: $link" -SmtpServer 10.0.0.18
```

---

## Related

- [2013](#)





---

## Create .csv from BOM

1. [Overview](#)
2. [Code](#)
  - 2.1. [Related](#)

---

### Overview

**!!! Not updated for 2014 yet !!!**

This tutorial gives you an example for creating a csv-file from the BOM via Powershell

---

### Code

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called BOM_to_Excel.ps1, and save it into the powerJobs folder. #Edit the
JobProcessor.exe.config to declare your new job. For queueing the job, you might use the
LifecycleEventEditor and configure your job on a given lifecycle
change.if(!$IAMRunningInJobProcessor){ Import-Module
"$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"
[System.Reflection.Assembly]::LoadFrom("$env:POWERJOBS_DLL") $vault = New-Object -type
coolOrange.PowerJobs.VaultProxy $vault.Login("Administrator","", "localhost", "Vault") # get
the file $file = $vault.GetUniqueLatestFileByFilename("Pad Lock.iam") if(!$file) { throw ("File
cannot be found") }}else{ # get the file $fileID = $vault.Job.Params["FileId"] if(!$fileID) {
throw ("File ID not set") } $file = $vault.DocService.GetFileById($fileID)}#gets the items with
the file-id $items = $vault.ItmService.GetItemsById($file.Id)#the item-masterid which you
need to get the bom$itemMaster = $items[0]if (!$items) {throw ("File has no item") }#Library to
use specific functions[System.Reflection.Assembly]::LoadFrom("C:\Program Files
(x86)\Autodesk\Autodesk Vault 2013 SDK\bin\Autodesk.Connectivity.WebServices.dll")#Gets the
BOM-file from the vault$itemBOM =
$vault.ItmService.GetItemBOMByItemIdAndDate($itemMaster.Id,[System.DateTime]::MinValue,[Autodesk.C
```



```

variables to get specific BOM informations
$boms = $itemBOM.ItemRevArray
$bomsOcc = $itemBOM.OccurArray
$bomsAss = $itemBOM.ItemAssocArray
#Gets LifeCycleStates of the items
$DispName = @{}
$defs = $vault.ItmService.GetAllLifeCycleDefinitions()
foreach($def in $defs){
    $DispName[$def.Id]=$def.DispName
}
#Gets the Quantity of the items
$quantity = @{}
for($i = 0;$i -lt $boms.Count;$i+=1){
    for($j = 0;$j -lt $bomsAss.Count;$j+=1){
        if($boms[$i].MasterID -eq $bomsAss[$j].CldItemMasterID){
            $quantity += $bomsAss[$j].CldItemUsage
        }
    }
}
#Write here the path where the csv-file should be placed
$FilePath = "C:\<YourFolder>\BOM$( $file.Name).csv"
#Writes with the PS function "out-file" the column names into .csv-file
"Number`tDetail_ID`tQuantity`tTitle`tRevision`tState`tUnits`n"|out-file $FilePath
#Writes with the PS function "out-file" the informations into .csv-file
for($i = 0;$i -lt $boms.Count;$i+=1){
    "$($boms[$i].ItemNum)`t$($bomsOcc[$i].Val)`t$($quantity[$i])`t$($boms[$i].Title)`t$($boms[$i].RevNum)"|out-file $FilePath -Append
}

```

---

## Related

- [2014](#)





---

## Create DWG from an IDW

1. [Overview](#)
  - 1.1. [Code](#)
2. [Related](#)

---

### Overview

As DWG is quite popular, you might want to create a DWG from an IDW via jobserver.

At line 20 you can change it also to other formats (.stp,...) that you want to create.

---

### Code

```
Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"$file
= PrepareEnvironmentForFile "Assembly1.idw" $true$powerJobs.Log.Info("Starting job 'Create
DWG' ...")# limit publishing to idw files$ext =
[System.IO.Path]::GetExtension($file.EntityName).ToLower()$inventorExtensions =
@(".idw")if($inventorExtensions -contains $ext){ # publish (generate the dwfx attachment)
$localDestFile = "C:\TEMP\" + $file.EntityName + ".dwg"
$publisher=$powerJobs.GetPublisher("PDF") $publisher.OutputFile=$localDestFile
#Eventhandler in which you can create other file formats $publisher.add_OnBeginPublish(
{ param($publisher, $document) $document.Document.SaveAs($localDestFile,$true)
}) #The Eventhandler gets called if(!$publisher.Open($File.EntityIterationId)) {throw
"The .dwg-translation failed!"}}
```

---

### Related

- [2014](#)





---

## Create textfile via template

1. [Overview](#)
  2. [Goal](#)
    - 2.1. [First Step](#)
    - 2.2. [Example with an Assembly1.idw file:](#)
  3. [Related](#)
- 

### Overview

Explains how to complete a self-paced learning exercise using a feature in the product.

---

### Goal

Supposing you have to write out several information into a file, and you like to keep the file format flexible, the idea could be to use a template file to drive the format.

#### Steps

---

### First Step

To use the script, create a file called "template.txt" (use the exact path in the script). Then open the "template.txt" and write the property names from which you might write informations out into a file. Use this Syntax: the property names have to be written in "{...}", the properties gets seperated by ";".

**Example:** {Name};This is my Classification: {Classification};It got created by: {Created By};. The format is flexible. You can also define a html-page: <html><table border="1"><tr><td>{Name}</td><td>{Classification}</td><td>{Created By}</td></tr></table></html>

```
Import-Module "$env:POWERJOBS_MODULES\coolOrange.PowerJobs.VaultHelper.psm1"$file
= PrepareEnvironmentForFile "Assembly1.idw" $true$powerJobs.Log.Info("Starting job
'Create_txt' ...")#Paths:#TEMPLATE-PATH$pathTemp = "C:\template.txt"$FilePath =
```



```
"C:\file.txt"#Propertydefinitions$propDefs =
$vault.PropertyService.GetPropertyDefinitionsByEntityClassId("FILE")#Writes the
Propertydefinitionids in a System.Array to use a specific method$sids = @()foreach($f in
$propDefs){ $sids+=$f.Id}#gets the file-properties$props =
$vault.PropertyService.GetProperties($file.EntityClass.Id,@($file.EntityIterationId),$sids)#Define
RegularExpression, to get the propertynames from the template-file$regex =
[regex]"{\{+(?i)\b[A-Z\s]+\b\}"$template = Get-Content -Path $pathTemp$header =
$regex.Matches($template)#Get the valuesforeach($match in $header){ $literalFieldName =
$match.Value.TrimStart("{").TrimEnd("}") $propId = 0 foreach($pd in $propDefs) {
if($pd.DispName.Equals($literalFieldName)) { $propId=$pd.Id } }
if($propId -gt 0) { foreach($pi in $props) {
if($pi.PropDefId.Equals($propId)) { $template = $template -replace
$match.Value, $pi.Val } } }#Writes the values in a file$template |Out-File
$FilePath -Append
```

---

## Example with an Assembly1.idw file:

template.txt:

**{Name}**;This is my Classification: **{Classification}**;It got created by: **{Created By}**;

My output file: file.txt:

**Assembly1.idw**;This is my Classification: **Design Document**;It got created by: **Administrator**;

---

## Related

- [2014](#)





---

## Data to XML

1. [Overview](#)
  2. [Code](#)
    - 2.1. [Related](#)
- 

### Overview

You may want to export information from Vault into an XML file.

---

### Code

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called Data_to_XML.ps1, and save it into the powerJobs\Jobs folder.#Edit the
JobProcessor.exe.config to declare your new job. For queueing the job, you might use the
LifecycleEventEditor and configure your job on a given lifecycle change.Import-Module
"$env:POWERJOBS_MODULES_DIR\coolOrange.PowerJobs.VaultHelper.psm1"$file =
PrepareEnvironmentForFile "Assembly1.idw" $true$powerJobs.Log.Info("Starting job 'Create_xml'
...")#Path where the .txt file and then the .xml file should be placed$path =
"C:\$(($file.EntityName))"#Propertydefinitions$propDefs =
$vault.PropertyService.GetPropertyDefinitionsByEntityClassId($file.EntityClass.Id)#Writes the
Propertydefinitionids in a System.Array to use a specific method$ids = @()foreach($f in
$propDefs){ $ids+= $f.id}#Gets the file-properties$props =
$vault.PropertyService.GetProperties("FILE",@($file.EntityIterationId),$ids)$values = @()$names
= @()#Gets the exact name and value of the propertiesfor($i = 0; $i -lt $propDefs.Count;
$i+=1){ for($j = 0; $j -lt $props.Count; $j+=1){ if($propDefs[$i].Id -eq
$props[$j].PropDefId -and $props[$j].ValTyp -ne "Image"){ $values +=
$props[$j].Val $names += $propDefs[$i].DispName } }#Writes the properties
first in a .txt-file#"root"-tag at the beginning and the end of the xml-file is necessary"<root>"|
Out-File "$($path).txt"for ($i = 0;$i -lt $props.Length-1; $i+=1){ $xmltags =
"<$(($names[$i])>$(($values[$i])</$(($names[$i])>" #The String have to be manipulated to
make a xml-file, because characters as space or parentheses are forbidden in tags
$xmltags.Replace(" ", "").Replace("(", "").Replace(")", "") + "`n" | Out-File "$($path).txt"
-Append}"</root>" | Out-File "$($path).txt" -Append#Writes the .txt-file content into the
xml-fileGet-Content "$($path).txt" | Out-File "$($path).xml"#Remove the txt-fileRemove-Item
"$($path).txt"
```





---

## Related

- [2014](#)





---

## PDF in an external folder

1. [Overview](#)
2. [First Step](#)
3. [Second Step](#)
4. [Related](#)

---

### Overview

A frequent request is to create a PDF and make this accessible to other systems like an ERP. Now you may know that a small script could grab the PDF from Vault and save it to the specified folder. However, as the PDF gets created via a job, why not having the job save the PDF to that location, right during the creation process.

---

### First Step

By default the '.PDF' is created locally anyway, and uploaded to Vault later. We could grab the local file and copy it to our desired location. The variable `$localDestFile` contains the location where the '.PDF' will be created.

---

### Second Step

After the successful creation, let's copy the file:

```
| Copy-Item -LiteralPath $localDestFile -Destination "C:\myFolder\$(($PDFfileName))"
```

to Line 38 from `coolOrange.powerJobs.CreatePdfAsAttachment.ps1`

---

### Related

- [2014](#)





---

## Print via Inventor

1. [Overview](#)
  2. [Code](#)
    - 2.1. [Related](#)
- 

### Overview

**!!! Not updated for 2014 yet !!!**

In order to print a file via Inventor, a similar approach to the one for creating a DXF can be taken. In this case we just open the file, catch the open event and then trigger the print. For printing, the PrintManager from Inventor is used. This allows us to be quite flexible on the printing requirements. Here is the code

---

### Code

```
# get the file $fileID = $vault.Job.Params["FileId"] if (!$fileID) { throw ("File ID not set") } $file =
$vault.DocService.GetFileById($fileID) # get the latest version of the file in case a sync prop has
been executed before the job $file = $vault.DocService.GetLatestFileByMasterId($file.MasterId) #
limit publishing to 2d inventor files $ext =
[System.IO.Path]::GetExtension($file.Name) if ($ext.ToLower().Equals(".idw")) {
    $publisher = $vault.GetPublisher("PDF") $publisher.add_OnBeginPublish( {
    param($publisher, $document) $printManager = $document.PrintManager
    $printManager.GetType().InvokeMember("Printer", [Reflection.BindingFlags]::SetProperty, $null,
    $printManager, "Microsoft XPS Document Writer") $printManager.ScaleMode =
    [Inventor.PrintScaleModeEnum]::kPrintBestFitScale $printManager.PrintRange =
    [Inventor.PrintRangeEnum]::kPrintAllSheets $printManager.PaperSize =
    [Inventor.PaperSizeEnum]::kPaperSizeA0 $printManager.SubmitPrint() })
    $publisher.OutputFile = "c:\temp\dummy.pdf" if (!$publisher.Open($file.Id)) { throw
    ("Open failed") }}
```



---

## Related

- [2014](#)





---

## Release via jobserver

1. [Overview](#)
  2. [INFO](#)
  3. [Code 1](#)
  4. [Code 2](#)
  5. [Code 3](#)
  6. [Related](#)
- 

### Overview

While releasing your document, some jobserver activities should be executed. To make sure that release means also that the jobserver tasks were successfully done, it's a good idea to let the jobserver finally release the document.

---

### INFO

**In order to change states via the api, you have to be able to change them via the gui as well. Otherwise it won't work.**

---

### Code 1

If you want to change a filestate (Work in Progress, Released ...) from a masterfile you can use the follow script:

```
Import-Module "$env:POWERJOBS_MODULES\coolOrange.PowerJobs.VaultHelper.psm1"$file
= PrepareEnvironmentForFile "Part2.ipt" $true$powerJobs.Log.Info("Starting job 'Release_File'
...")#Path where the .txt file and then the .xml file should be placed$path =
"C:\$(($file.EntityName))"#All LifeCycleDefinitions get hooked, you need them to set a filestate$def
= $vault.DocumentServiceExtensions.GetAllLifeCycleDefinitions()#In this case we take the
"Flexible Release Process"$FlexibleReleaseProcess = $def | Where-Object
{$_.DispName.Equals("Flexible Release Process")}$#From the "FlexibleReleaseProcess"-object you
```

---



```
can take your favorite state to set#in this case "Released"$releaseState =
$FlexibleReleaseProcess.StateArray | Where-Object {$_.Name.Equals("Released")}}#The
masterfilestate gets changed to
"Released"$vault.DocumentServiceExtensions.UpdateFileLifeCycleStates(@($file.EntityMasterId),@($releaseS
from PS")
```

---

## Code 2

If you want to change filestates (Work in Progress,Released ...) from childfiles you can use this script:

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called ChangeState.ps1, and save it into the powerJobs folder.#Edit the JobProcessor.exe.config
to declare your new job. For queueing the job, you might use the LifecycleEventEditor and
configure your job on a given lifecycle change.Import-Module
"$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"$file =
PrepareEnvironmentForFile "Assembly1.iam" $true$powerJobs.Log.Info("Starting job
'change_state_child' ...")#All LifeCycleDefinitions get hooked, you need them to set a
filestate$def = $vault.DocumentServiceExtensions.GetAllLifeCycleDefinitions()#In this case we
take the "Flexible Release Process"$FlexibleReleaseProcess = $def | Where-Object
{$_.DispName.Equals("Flexible Release Process")}}#From the "FlexibleReleaseProcess"-object you
can take your favorite state to set$releaseState = $FlexibleReleaseProcess.StateArray |
Where-Object {$_.Name.Equals("Released")}}#Get all childfiles$assocs =
$vault.DocumentService.GetFileAssociationsByIds(@($file.EntityIterationId),[Autodesk.Connectivity.WebServ
-ne $null){ $stateIds = @() $childfileIds = @() foreach($f in $assocs[0].FileAssocs){
$childfileIds += $f.CldFile.MasterId $stateIds += $releaseState.Id }}#The childfilesstates
get changed to
"Released"$vault.DocumentServiceExtensions.UpdateFileLifeCycleStates($childfileIds,$stateIds,"Released
from PS")
```

---

## Code 3

If you want to change a filecategory (Engineering,Base...) you can use this script:

```
Import-Module "$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"$file
= PrepareEnvironmentForFile "Part2.ipt.pdf" $true$powerJobs.Log.Info("Starting job
'change_category' ...")#All category-definitions get hooked$defc =
$vault.CategoryService.GetCategoriesByEntityClassId("FILE",$true)#In this case we want to set
the filecategory to "Engineering"$category = $defc | Where-Object
{$_.Name.Equals("Engineering")}}#Change
Category$vault.DocumentServiceExtensions.UpdateFileCategories(@($file.EntityMasterId),$category.Id,"Cate
Change by PS")
```

---

## Related

- [2014](#)





---

## Retrieve the user that queued the job

1. [Goal](#)
2. [Info](#)
3. [Code](#)
4. [Related](#)

---

### Goal

Especially if a job has been queued via a lifecycle, you may want to know the user that queued that job

---

### Info

If you have trouble debugging the job, then execute it via the jobserver. To test the script you can add the code on line 25 of our 'Create PDF' - job and make a pdf after saving it. As a result you should get a textfile C:\usertest.txt with the information of the user, which executed the job.

---

### Code

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

---

### Related

- [2014](#)





---

## Selected files to ZIP

1. [Goal](#)
  2. [Code](#)
    - 2.1. [Related](#)
- 

### Goal

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.

---

### Code

```
#To test this script, just copy&paste the content above into a new powershell file, for instance
called FilesToZip.ps1, and save it into the powerJobs folder. #Edit the JobProcessor.exe.config to
declare your new job. For queueing the job, you might use the LifecycleEventEditor and configure
your job on a given lifecycle change.Import-Module
"$env:POWERJOBS_MODULESDIR\coolOrange.PowerJobs.VaultHelper.psm1"$file =
PrepareEnvironmentForFile "Assembly1.idw" $true$powerJobs.Log.Info("Starting job
'Zip_Structure' ...")#Path for the final zipfile$zipout = "C:\myZips\"#region
DownloadSettings#Generate a Settingsobject. It is necessary for the new AcquireFiles
method$settings = New-Object
Autodesk.DataManagement.Client.Framework.Vault.Settings.AcquireFilesSettings($vaultconnection,
>false)$settings.AddEntityToAcquire($file)#Temporary path for the downloaded
files$settings.set_LocalPath("C:\TEMP\zip")#"Checkout", "Download", "NoAction"are possible
values$settings.set_DefaultAcquisitionOption("Download")#What should be included in the
zip?$settings.OptionsRelationshipGathering.FileRelationshipSettings.set_IncludeChildren("TRUE")$settings.O
the files$files = $vaultConnection.FileManager.AcquireFiles($settings)#It creates a new folder
even though it existNew-Item -Path $settings.get_LocalPath() -Force -ItemType
DirectoryNew-Item -Path $zipout -Force -ItemType Directory#Create Zip-File$zipFileName =
"($zipout)$($file.EntityName).zip"set-content $zipFileName ("PK" + [char]5 + [char]6 +
"$([char]0)" * 18))$ZipFile = (new-object -com
shell.application).Namespace($zipFileName)$fileToBeZipped = Get-ChildItem
$settings.get_LocalPath()$fileToBeZipped | ForEach-Object { #The method .MoveHere() is
running asynchronous, so we have to wait a few seconds before moving the next file into zip
```





```
#In this case: 2 seconds    Start-Sleep -Seconds 2    #Write files into the zipfile  
$ZipFile.MoveHere($_.FullName,1024)}
```

---

## Related

- [2014](#)





---

## Send email via jobserver

1. [Overview](#)
2. [Code](#)
3. [Notification via Email](#)
4. [To address from Vault property](#)
  - 4.1. [Related](#)

---

### Overview

sending an email at the end of a job is quite simple. The powershell language provides a ready to use command-let called send-mailmessage.

---

### Code

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

```
Import-Module "$env:POWERJOBS_MODULES_DIR\coolOrange.PowerJobs.VaultHelper.psm1"$file
= PrepareEnvironmentForFile "Part2.ipt" $true$powerJobs.Log.Info("Starting job 'send_email'
...")#region Config Mail  $from = "powerjobs@yourdomain.com" #Required  $to =
"user@targetdomain.com" #Required  $subject = "The document $($file.EntityName) has been
processed" #Required  $body = "Write here your email text and use variables to add additional
information" #Optional#endregion#region Config SMTP  $smtp = "yourSMTPServer"  $passwd
= ConvertTo-SecureString -AsPlainText "YourPassword" -Force  $cred = new-object
Management.Automation.PSCredential $from, $passwd#endregion#region Config Attachment
$sendattachment = $true#endregion#region Send Message  if($sendattachment -eq
$true){      Send-MailMessage -To $to -From $from -Subject $subject -Body $body -SmtpServer
$smtp -Credential $cred -Attachments "C:\TEMP\$(($file.EntityName))"  }  else{
Send-MailMessage -To $to -From $from -Subject $subject -Body $body -SmtpServer $smtp
-Credential $cred  }#endregion
```



---

## Notification via Email

**!!! Not updated for 2014 yet !!!**

In order to send emails, you need a SMTP server. So, we assume this is given and you have the credentials. Then, it's an easy game. PowerShell comes with a commandlet called Send-MailMessage. Just add the given arguments like from, to, subject, body and a valid SMTP server, and you have emails sent by the Jobserver at lifecycle transitions. The content of the email can be configured as needed. Here a sample that sends an email with the file name with from and to state in the subject, and a Vault link in the body. If the user clicks on the link, Vault will start and point to the according file.

```
$lfcTransId = $vault.Job.Params["LifeCycleTransitionId"]$fileID = $vault.Job.Params["FileId"]$file = $vault.DocService.GetFileById($fileID)$folders = $vault.DocService.GetFoldersByFileMasterId($file.MasterId)$folder = $folders[0]$folderPath = $folder.FullName$folderPath = $folderPath.Replace("$", "%24").Replace("/", "%2f")$fullPath = $folderPath + "%2f" + $file.Name.Replace(" ", "+")$link = "http://localhost/AutodeskDM/Services/EntityDataCommandRequest.aspx?Vault=Vault&ObjectId="+$fullPath+"&ObjectType=File&Command=Select"$newState = $vault.DocExtService.GetLifeCycleStateTransitionsByIds(@($lfcTransId))$lfc = $vault.DocExtService.GetLifeCycleStatesByIds(@($lfcTrans[0].FromId,$lfcTrans[0].ToId))$oldState = $lfc[0].DispName$newState = $lfc[1].DispNameSend-MailMessage -From "marco.mirandola@coolorange.com" -To "marco.mirandola@coolorange.com" -Subject "The file $($file.Name) has changed from $oldState to $newState" -Body "Dear xxx, if you like to view the related document, just follow this link: $link" -SmtpServer 10.0.0.18
```

---

## To address from 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.

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

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.

---

## Related

- [2014](#)





---

## Copy file in a directory

1. [Overview](#)
2. [Goal](#)
  - 2.1. [Code](#)
3. [Related](#)

---

### Overview

Gives you an idea to solve this problems with powershell

---

### Goal

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

Steps

---

### Code

```
#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 existingif(!(Test-Path $targetDirectory)){mkdir $targetDirectory}#file get copyed to directoryCopy-Item -Path $sourceFile -Destination $targetDirectory#The Copy-Item cmdlet contains a lot more interesting options, for instance -Force#to force overwriting, or -Recurse to copy a complete folderstructure
```

---

### Related

- [powershell general](#)





---

## Insert into SQL server

1. [Goal](#)
  - 1.1. [First Step](#)
  - 1.2. [Second Step](#)
2. [What's Next](#)
3. [Related](#)

---

### Goal

Supposing you like insert some rows into an SQL server, powershell can help you. This could be useful to inform a foreign System about some changes in Vault, or fill tables with data for further use, or make data available to other applications, etc.

#### Steps

---

### First Step

**Prerequisites** for the example below

- 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

---

### Second Step

```
#create connection object$conn = New-Object System.Data.SqlClient.SqlConnection("Data Source=MySQLServer; Initial Catalog=TestDB; Integrated Security=SSPI")#open database
$conn.Open()#get a command object$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()
```

---

## What's Next

This is what was achieved and what was omitted in this tutorial.

---

## Related

- [powershell general](#)





---

## Print/convert Office documents

1. [Overview](#)
  2. [Prerequisites](#)
    - 2.1. [Word](#)
    - 2.2. [Excel](#)
  3. [Powerpoint](#)
  4. [Related](#)
- 

### Overview

You may want to print or convert Office documents, to a PDF file.

---

### Prerequisites

You need to have Microsoft Office 2010 or higher installed

---

### Word

You may want to print or convert Office documents, like **Word** , to a PDF file.

```
$word = New-Object -ComObject Word.ApplicationSleep -Seconds 10$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

You may want to print or convert **Excel** documents, to a PDF file.

```
#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

You may want to print or convert **Powerpoint** documents, to a PDF file.

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

This is what was achieved and what was omitted in this tutorial.

---

## Related

- [powershell general](#)







---

## Set default printer

### 1. [Goal](#)

#### 1.1. [Code](#)

#### 1.2. [Related](#)

---

## Goal

Setting the default printer might help you to print documents on the device you want, also with applications that have a poor API to control the printer.

---

## Code

```
#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()
```

---

## Related

- [powershell general](#)





---

## Simple document print on default printer

1. [Overview](#)
2. [Code](#)
3. [Related](#)

---

### Overview

How to simply print a document on the default printer.

---

### Code

```
#Write here the file-path$document = "C:\<YourFolder>\<YourFile>"#Prints the document content and waits until the process endsStart-Process -FilePath $document -Verb Print -Wait
```

---

### Related

- [powershell general](#)





---

## 07. Patchnotes

1. [Overview](#)
  2. [powerJobs 2014](#)
  3. [powerJobs 2013](#)
  4. [Related](#)
- 

### Overview

The following lists contain all related versions of powerJobs including a description of the relevant changes for each version.

---

### powerJobs 2014

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

---

### powerJobs 2013

Date	Version	Description
24.09.2013	13.0.129	fixed bug multisheet of pdf creation with inventor 2013 in designdrawgins



Date	Version	Description
	13.0.128	

---

## Related

- [powerJobs](#)





---

## 08. Troubleshooting

1. [Overview](#)
2. [Log4Net](#)
  - 2.1. [Logginglevel](#)
  - 2.2. [Errorlog Paths](#)
  - 2.3. [When to use this feature?](#)
3. [You get an error that some ddls cannot be found, while debugging in Powergui](#)
4. [Your jobserver hangs if you try to make PDFs or stops with an error](#)
5. [Related](#)

---

### Overview

This page contains information about how to handle some known issues.

---

### Log4Net

In the file 'coolOrange.powerJobs.dll.log4net' you can configurate powerJobs errorlogging. In the standardinstallation you can find the file

coolOrange.powerJobs.dll.log4net

under

C:\ProgramData\Autodesk\Vault 2013\Extensions\coolOrange.PowerJobs.Handler

In the line

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

you can config the logginglevel.



In the line

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

you can config the outputpath and name of the logfile.

For further information about Log4Net you could 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.

---

## When to use this feature?

Use this feature if you like to better understand what happens during the execution of your job, or in case we request more information to support you.



---

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

Make sure you are using the right version of powergui. For older versions of powerJobs you have to use 32bit version of powergui, but for powerJobs2014 you have to use the 64bit version.

---

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

Make sure that you started DWGTrueView and Inventor at least once and restarted your pc afterwards, before using powerJobs.

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

---

## Related

- [powerJobs](#)





---

## 09. FAQ

1. [Overview](#)
2. [Related](#)

---

### Overview

Frequently asked questions.

---

### Related

Topics

[Where's documentation for the Vault API?](#)

Description of the Vault API

---

Tutorials

- [Convert PDF to DWF](#) (Beginner)
- [How can I add a watermark/stamp to pdf?](#) (Beginner)
- [How can I add the revision to the name of the pdf-file?](#) (Beginner)
- [How can I convert pdf to pdf/a?](#) (Beginner)
- [How can I create a pdf with corresponding properties from original file?](#) (Beginner)
- [How can I create a PDF with same category, revision, state as the original file?](#) (Beginner)







---

## Convert PDF to DWF

1. [Overview](#)
2. [Goal](#)
  - 2.1. [First Step](#)
  - 2.2. [Related](#)

---

### Overview

The standard jobs of powerJobs can not create a DWF from a PDF-file, but when you are looking for a solution of doing this, we will give you some help.

---

### Goal

After reading this tutorial and the blog you will be able to create your DWFs from a PDF.

Steps

---

### First Step

All the information you'll need for creating your own job for powerJobs you can find on our blog, please use the link to our blog:

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

---

### Related

- [09. FAQ](#)





---

## How can I add a watermark/stamp to pdf?

1. [Overview](#)
  2. [Goal](#)
    - 2.1. [First thing to do](#)
    - 2.2. [Sample](#)
    - 2.3. [How to use the these module in the powerJobs.CreatePdfAsAttachment job](#)
      - 2.3.1.1. [with text](#)
      - 2.3.1.2. [with images](#)
    - 2.4. [How does the function work ?](#)
      - 2.4.1. [Insert a stamp](#)
      - 2.4.2. [Insert a watermark](#)
      - 2.4.3. [Insert a image](#)
  3. [iText API](#)
  4. [Related](#)
- 

## Overview

Explains how to add watermarks or stamps to pdf attachments via the iTextSharp.dll on PowerShell base. The watermark/stamp can be a text or an image. Also you get showed how to add the script as a module to the coolOrange.powerJobs.CreatePdfAsAttachment.ps1 job.

---

## Goal

After completing this tutorial you will have the knowledge to customize your pdf attachments with images, watermarks and stamps.



---

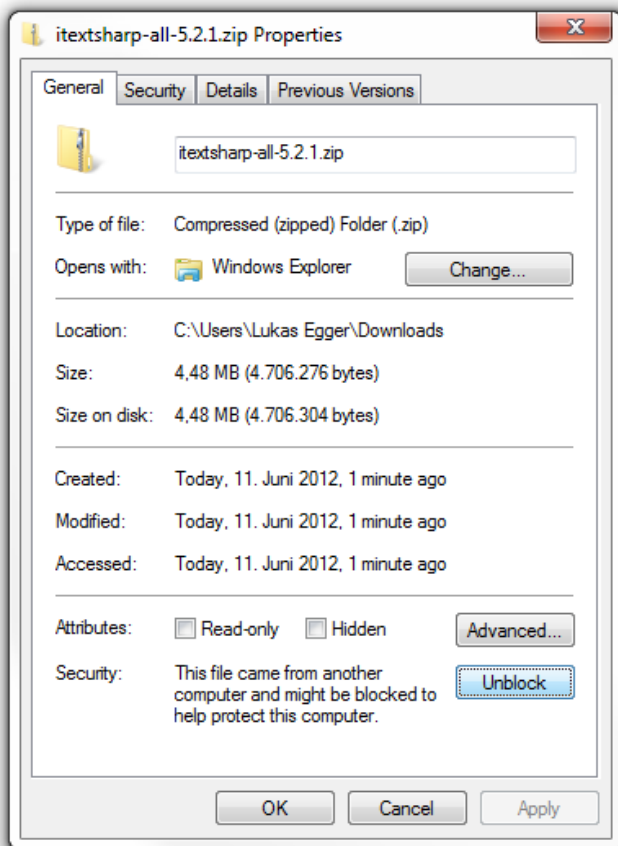
## First thing to do

The first thing we have to do, is the download of the iTextSharp.dll. You can find the newest version of iTextSharp you can find under this link: <http://sourceforge.net/projects/itextsharp/files/latest/download>

After downloading you have to "Unblock" your archive, else Windows wont allow the access to the iTextSharp.dll via PowerShell.

You find the "Unblock" button with rightclick on the iTextSharp archive, then properties and afterwards in the general-tab at the bottom.

It should look like this:



Click on the button to unblock the file, then you can extract the file.

Once the content is extracted, take the itextsharp.dll from the itextsharp-dll-core-5.2.1 folder and copy it to C:\ProgramData\coolOrange\powerJobs\Modules. The PowerShell script attached to this post points to this folder.



---

## Sample

You can simply use our module coolOrange.powerJobs.PdfHelper.psm1, provided as attachment on this page below. Just download it and move it in the C:\ProgramData\coolOrange\powerJobs\Modules folder. Now you can use or customize the functions of the module in every Job, you want.

---

## How to use the these module in the powerJobs.CreatePdfAsAttachment job

### with text

Just add the marked row at the same point as noted in this picture if you want plain text as stamp in pdf files:

```
51 | if ($publisher.Publish($file.Id) -eq $true)
52 | {
53 |     Add-StampToPdf $localDestFile "sampletext"
54 |     $folders = @($vault.DocService.GetFoldersByFileMasterId($file.MasterId))
```

Or add the following row if you wan the lifecyclestate of your file as watermark

```
50 | if ($publisher.Publish($file.Id) -eq $true)
51 | {
52 |     Add-WatermarkToPdf $localDestFile $file.FileLfCyc.LfCycStateName
53 |     $folders = @($vault.DocService.GetFoldersByFileMasterId($file.MasterId))
```

Of course you can use the same row also for Add-StampToPdf.

### with images

Just use instead of a normal text the path of an image you want as stamp or watermark. The watermark and stamp function will search the file and if the path is correct, the functions will insert the image in the pdf, else the functions use the text as in the example above.

---

## How does the function work ?

### Insert a stamp

Now we are making our Stamp script. It gets explained at the end how to add an image or a watermark, because there is only a little difference.

To write the script make sure you use atleast Powershell ISE or to have a better workflow powerGUI as IDE. For more information about IDEs look into the "[IDEs for powershell](#)" topic.

The first step to our script is the loading of the iTextSharp.dll in our script enviroment, we get this done with this row.

```
[System.Reflection.Assembly]::LoadFrom("C:\Sample_Path\itextsharp-all-5.2.1\itextsharp-dll-core-5.2.1\ite
```



These are some font settings of the text, that we want to insert as a stamp in the pdf:

```
#region fontsettings$baseFont =  
[iTextSharp.text.pdf.BaseFont]::CreateFont([iTextSharp.text.pdf.BaseFont]::HELVETICA,[System.Text.Encoding]::UTF8,120.0,$color = New-Object iTextSharp.text.BaseColor (200,200,100)#calculating the angel of  
the stamp with trigometrics the angel is around 45° in an A4 page$textAngle =  
([System.Math]::Atan2($pageSize.Height, $pageSize.Width) + (180/  
[System.Math]::PI))*1.0#endregion
```

\$basefont just describes our font, what fondstyle it will use, what character encoding is preferred and if the text is embedded. \$fcolor is the font color in RGB. The \$textangel decides in what angel the text is written.

Now we begin with the proper PowerShell script. Here we go:

```
$file = New-Object System.IO.FileInfo "c:\\sample.pdf"$reader = New-Object  
iTextSharp.text.pdf.PdfReader $file.FullNametry{ $mStream = New-Object  
System.IO.MemoryStream $stamper = New-Object iTextSharp.text.pdf.PdfStamper ($reader,  
$mStream) $padding = 2.0
```

\$file is our pdf that we want to manipulate with a stamp, with this variable we get the pdf destination. Then we create the PdfReader, it is a basic object of iTextsharp, with what we get the reading access to the file. The \$mStream MemoryStream has to save all our changes to the original pdf content, because the PdfStamper stamper will only provide a copy of the pdf content, we have later to overwrite the old pdf with the new manipulated pdf content.

```
$index = 1 do{ $pdfSize = $reader.GetPageSizeWithRotation($index) $overContent =  
$stamper.GetOverContent($index) # begin creating the overLayer $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(); #  
finished creating
```

The \$index must be 1 because in a pdf the page 0 doesnt exist. In the following loop we are manipulating one by one the pagecontent of our pdf. \$pdfSize contains the measures of the current pdf page, it makes a difference if the page is rotated or not. overContent is the variable filled by the \$stamper.getOverContent(\$index) and provides the highest layer of the pdf. Here we will place our stamp text or image, there is just one problem, if the highest layer is empty, there will be no area to write on. So we create our own layer(rectangle) and draw it on the page. For that we are using the MoveTo() and LineTo(), how you can see the code will just draw from one corner to the next corner till we have a rectangle with one missing site. ClosePath() will add the missing site to the rectangle and place the rectangle in the overLayer.

```
# inserting text $overContent.BeginText()  
$overContent.SetFontAndSize($baseFont,$fontSize) $overContent.setColorFill($color);  
$overContent.ShowTextAligned([iTextSharp.text.pdf.PdfContentByte]::ALIGN_CENTER,  
$stampText, $pdfSize.Width / 2, $pdfSize.Height / 2, $textAngle) $overContent.EndText() #  
finished inserting text $index ++ }while($index -lt $reader.NumberOfPages)
```



Afterwards we can begin to write the text in the now writeable layer. Just write `$overContent.BeginText()` to activate the write-mode in our `$overContent`, then we set the font, the font size, font color and we insert the text with `$overContent.ShowTextAligned()` the first parameter decides, if the text is centered or aligned to the left etc., the second parameter is the text as a string, the two following parameters are the x and y coordinate of the text and the last one is the angle of the text.

```
$stamper.Close(); $fileStream = [System.IO.File]::OpenWrite($file.FullName)
$fileStream.Write($mStream.ToArray(), 0, $mStream.ToArray().Length)}catch{ throw
$_}finally{ $fileStream.Close() $mStream.Close()}
```

Now we have nearly finished. We close the stamper, because we wont change more in the content and the changes, that are already done, are saved in the `$mStream`. We open a `fileStream` to our original pdf and write the hole new content from `$mStream` to our pdf with `$fileStream.Write()`. In the last finally statement we make sure to close our 2 streams, so afterwards we wont have problems with them.

## Insert a watermark

If you want a watermark. Just remove the part where we created the rectangle in the `overLayer` and change the method `getOverLayer($index)` to `getUnderLayer($index)`. That should do the magic.

## Insert a image

If you want your stamp as an image and not as a text. You have to remove the "inserting text" part, make sure you haven't removed the "creating rectangle" part! And in place of the "inserting text" part, write the following code:

```
$image = [iTextSharp.text.Image]::GetInstance("C:\\Users\\Lukas
Egger\\Documents\\praktikum\\myJobs\\test.png")$image.SetAbsolutePosition($pdfSize.Width /
2 - $image.Width/2, $pdfSize.Height / 2-$image.Height/2)$overContent.AddImage($image)
```

Do the same thing as in "Insert a watermark" and then remove also here the " inserting text" part and add also this time the 3 rows above. Regardless if you want to insert a stamp- or a watermark-image the font settings part has here no use, just remove it.

---

## iText API

On the web is unfortunately only a free Java API, so we have to work with that API. You can find it under this link: <http://api.itextpdf.com/itext/>

Note that on the world wide web are many Java and C# tutorials for iText. You can use these as an example and rewrite/modify them to PowerShell scriptcode without much of an effort in most cases.

---

## Related

- [09. FAQ](#)





---

## How can I add the revision to the name of the pdf-file?

1. [Overview](#)
  2. [Goal](#)
    - 2.1. [First Step](#)
    - 2.2. [Second Step](#)
  3. [What's Next](#)
  4. [Related](#)
- 

### Overview

This tutorial gives you an overview to change the name of the creating pdf-file, for example by using the revision of the file.

---

### Goal

The goal is to create pdf-files with a specific name on powerjobs by editing the job.

#### Steps

---

### First Step

By giving a look at the job-file "coolOrange.powerJobs.CreatePdfAsAttachment" you will find the entry on line #36 where the name of the created pdf-file is defined. Here we have to make some changes in the way that the revision will be appear in the name of the pdf-file:

```
#define here the file name for the generated PDF
```

```
$PDFfileName=$file.Name + ".pdf"
```



---

## Second Step

Before we can add the revision we have to get this information from the Vault using the Vault-API.

```
#get the revision of a file$props =$vault.PropService.GetPropertiesByEntityIds("FILE",
@($file.Id))$propDefs =$vault.PropService.GetPropertyDefinitionsByEntityClassId("FILE")$revDef
=$propDefs | Where-Object { $_.DispName -eq "Revision" }$revision =$props | Where-Object {
$_PropDefId -eq $revDef.Id }
```

With this powershell-lines we can get the revision from the selected file \$revision. After this we can define the name of the created pdf:

```
$PDFfileName= [System.IO.Path]::GetFileNameWithoutExtension($file.Name)
+"_"+"$revision.Val "+"pdf"
```

This is an example for the new pdf-filename.

---

## What's Next

This is what was achieved and what was omitted in this tutorial.

---

## Related

- [09. FAQ](#)







---

## How can I convert pdf to pdf/a?

The following script renders the table of contents for this page.

1. [Overview](#)
2. [Automated Conversion](#)
3. [First Step](#)
4. [Second Step](#)
5. [Note](#)
6. [Related](#)

---

## Overview

Is it possible to convert pdf files from the coolOrange.powerJobs.CreatePdfAttachment script to pdf/a ?  
Yes it is! The following instructions will show you how you can access that function with Ghostscript.

---

## Automated Conversion

Do you want that the CreatePdfAttachment job creates immediately a PDF/a 1-b ?

Then create under the LocalDestFile variable a new variable called localOrigFile with the path "C:\TEMP\original" + \$PDFfileName. Afterwards just copy the function in one of the modules that you use for powerJobs, preferably just use the coolOrange.powerJobs.PdfHelper.psm1, where Format-PdfToPDfa1b is already included. Now call under the row 52 Format-PdfToPDfa1b and specify, that the original is the one in the \$localOrigFile and the converted file is the \$LocalDestfile. At the end where the script Removes the \$localDestFile you have to add Remove-Item \$localOrigFile.

The begin of the file should look like this:



```

45 # publish (generate the pdf attachment)
46 $localOrigFile = "C:\TEMP\original" + $PDFFileName
47 $localDestFile = "C:\TEMP\" + $PDFFileName
48 $publisher=$vault.GetPublisher("PDF")
49 $publisher.OutputFile=$localDestFile
50 $checkInAsHidden = !$showPDF
51 if ($publisher.Publish($file.Id) -eq $true)
52 {
53     Format-PdfToPdfa1b $localOrigFile $localDestFile
54 }
55 $folders = @($vault.DocService.GetFoldersByFileMasterId($file.MasterId))

```

The part with the removing has to look like this:

```

94 }
95
96 # delete the physical file
97 Remove-Item $localDestFile
98 Remove-Item $localOrigFile
99 }

```

## First Step

The first we do, is downloading Ghostscript, you can use this link: <http://sourceforge.net/projects/ghostscript/files/GPL%20Ghostscript/9.05/gswin64.exe/download>

Note: You have to use 64bit version, because there can be issues with the 32bit, because the converting can need alot ressources.

After downloading just execute the .exe and install the tool.

## Second Step

The function itself has only some rows:

```

function Format-PdfToPdfa1b($original,$converted){$convertedparam = "-sOutputFile=" +
$converted$oldloc = Get-Locationcd "C:\Program Files\gs\gs9.05\bin\".\gswin64c.exe
-sDEVICE=pdfwrite -q -dNOPAUSE -dBATCH -dNOSAFER -dPDFSA -dUseCIEColor
-sProcessColor=DeviceCMYK $convertedparam $originalcd $oldloc.Path}

```

Make sure where your Ghostscript is located. If it has a different location, you have to update the path at row 5. Then you must copy the function Format-PdfToPdfa1b in a module that you want to include in your job or in a other job, where you want to use this function and etc. Then you can call the function with the following line :

```
Format-PdfToPdfa1b "sample.pdf" "convertedsample.pdf"
```

If you need more information about the parameters for the gswi64c.exe, you can use the documentation of Ghostscript: <http://www.ghostscript.com/doc/9.05/Readme.htm>



---

## Note

The PDF/a 1-b file can be really large. That is because all fonts have to be embedded and layers increase also the file.

There is no warranty that the Conversion will work always, we tested the function it worked with every pdf file we used, but still it may not work with special pdfs.

Use one of the following Pdf-Validator, 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>

You can find the coolOrange.powerJobs.PdfHelper.psm1 in the attachments.

---

## Related

- [09. FAQ](#)





---

## How can I create a pdf with corresponding properties from original file?

1. [Overview](#)
  2. [Goal](#)
    - 2.1. [First Step](#)
    - 2.2. [Second Step](#)
  3. [What's Next](#)
- 

### Overview

We will Vault function `UpdateFileProperties`, but before we need know how to use that: the file must before checket out, then we can update the properties and after that we can check the file in. We will give you also an exemple how to get the properties of a file from the Vault.

---

### Goal

After completing this tutorial you will have some understanding of work with the Vault. The goal is not just to give you a complete working code, but to give you some ideas of how the Vault is working.

#### Steps

---

### First Step

We need the Vaultfunction `UpdateFileProperties` to set the properties of a file that is checked out. After this we can check in the file. This function needs some parameters like property-ids and property-values, so we have to get this informations before.

---

### Second Step

We can give a look to some lines of code:

---

[http://wiki.coolorange.com/powerJobs/09.\\_FAQ/How\\_can\\_I\\_convert\\_pdf\\_to\\_pdf%2F%2Fa%3F](http://wiki.coolorange.com/powerJobs/09._FAQ/How_can_I_convert_pdf_to_pdf%2F%2Fa%3F)

Updated: Wed, 26 Feb 2014 10:18:56 GMT

Powered by mindtouch™



```
#region collect property definitions information$propDefs
=$vault.PropService.GetPropertyDefinitionsByEntityClassId("FILE") #get all property definitions
from Vault$sudpDefs=$propDefs | Where-Object { $_.IsSys -eq$false } #filter user defined prop.
only$sudpIds= @()$sudpDefs| ForEach-Object { $sudpIds+=$_ .Id } #collect just the
ids#endregion#region collect values from existing
file$props=$vault.PropService.GetProperties("FILE",@($file.Id),$sudpIds) #get selected (user
defined)properties from orig file$pIds= @()$vals= @()$props| ForEach-Object {
$pIds+=$_ .PropDefId; $vals+=$_ .Val } #create 2 lists: ids and vals#endregion
```

---

## What's Next

Now we can check out the file and update the properties and then check it in:

```
#region apply properties to PDF#check out the PDF$newPdfFile
=Get-CheckoutVaultFile$vault$folder$pdfFile#update the
properties$vault.DocService.UpdateFileProperties(@($newPdfFile.MasterId),$pIds,$vals)#checkin
the file$newFile
=Get-CheckinVaultFile$vault$existingFile$localDestFile$null$checkInAsHidden#endregion
```

- [09. FAQ](#)





---

## How can I create a PDF with same category, revision, state as the original file?

1. [Overview](#)
2. [Goal](#)
  - 2.1. [First Step](#)
  - 2.2. [Second Step](#)
3. [Related](#)

---

### Overview

Supposing you would like to create a PDF with the same Category, same State and the same Revision Number as the file from which you would like to create a PDF, this tutorial will help you.

---

### Goal

The goal is to have the PDF-generation with the same Category, State and Revision as the original file.

---

### First Step

open the "coolOrange.powerJobs.CreatePdfAsAttachment.ps1" script and copy&paste these following lines in the line 85 (before this function "Add-VaultDesignVizualizationFile \$vault \$file \$newFile " gets called) :

```
$vault.DocExtService.UpdateFileCategories(@($newFile.MasterId),@($file.Cat.CatId),"<Your  
Comment>")$vault.DocExtService.UpdateFileRevisionNumbers(@($newFile.id),@($file.FileRev.Label),"<Y  
Comment>")$vault.DocExtService.UpdateFileLifeCycleStates(@($newFile.MasterId),@($file.FileLfCyc.LfCy  
Comment>")
```



---

## Second Step

Instead of using this function

`$vault.DocExtService.UpdateFileCategories(@($newFile.MasterId),@($file.Cat.CatId),"<Your Comment>")` we advise you to use a Vault Option. Just open the vault, click on Tools/Administration/Vault Settings a window should open, then click on the Behaviors tab and click on the Rules button to declare your new rule. In this case, click on New set the Rule Name maybe to "PDF" and choose your Category Assignment, click on the ok button and make a tick in the "Apply rules on file creation"-checkbox and apply your changes.

---

## Related

- [09. FAQ](#)





---

## Where's documentation for the Vault API?

1. [Overview](#)
2. [Details](#)

---

### Overview

Description of the Vault API

---

### Details

If you have installed the Autodesk Vault 2013, you can find the documentation local under this path C:\Program Files (x86)\Autodesk\Autodesk Vault 2013 SDK\docs. For all previous versions you have perhaps only to change the version of Vault in the path.

You can also visit the Autodesk Developer Network(<http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=13433205>). There you will find a developer community and some tutorials for the Vault API.

