powerEvents

coolOrange s.r.l

Oct 09, 2024

POWEREVENTS

CHAPTER

ONE

INSTALLATION

1.1 Requirements

As powerEvents is an extension to *Vault Applications*, the Vault system requirements defined by Autodesk leads.

Operating System: 64-bit only

- Microsoft Windows 10
- Microsoft Windows 11

Autodesk Vault Client: 2024 / 2023 / 2022 / 2021

- Vault Professional
- Vault Workgroup

Windows PowerShell: PowerShell 4.0 or higher

powerVault: is installed automatically

1.2 Setup

The powerEvents setup is delivered as an executable and accepts the standard windows installer arguments documented here.
To accept the products EULA when starting the setup in silent mode pass the ***ACCEPT_EULA=1*** argument:

```
"\\path\to\networklocation\powerEvents24.0_Vault2024.exe" -silent ACCEPT_EULA=1
```

1.3 Windows permissions

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

Path	Required Rights
C:\ProgramData\coolOrange\	**Read**, **Write** and **Delete** data

1.4 Install Locations

powerEvents is installed in the following locations on your system:

- All program libraries and executable files are placed in *C:\ProgramData\Autodesk\[Vault Version]\Extensions\powerEvents*
- The Cmdlets will be installed to *C:\Program Files\coolOrange\Modules\powerEvents*
- All client customization files, e.g. scripts and module libraries, are placed in *C:\ProgramData\coolOrange\Client Customizations*
- The script *C:\ProgramData\coolOrange\Publish-Customizations.ps1* helps distribute the client customizations to other Vault environments

Following shared libraries are installed in *GAC*:

- coolOrange.logging.dll
- coolOrange.VaultServices_[Vault Version].dll

Following shortcuts are added in the start menu:

- **powerEvents Configuration** - Opens the directoy with the files of the different client customizations
- **powerEvents ISE** - Opens the IDE and opens a sample script
- **powerEvents Information** - Opens the About dialog with product related information
- **powerEvents License Information** - Opens the License Information dialog to activate the product
- **powerEvents Logs** - Opens the log file location

1.5 Updates

To install a newer version of powerEvents just execute the setup file of the new version. This will automatically update the files in the existing installation.
Please note that also sample customizations (and standard scripts for powerJobs Client) will be updated.

If a tested version of your customizations has already been *distributed* to other workstations, it is therefore **recommended** to run the update on one environment first.
Then start a Vault application to test possible changes. In case of local modifications a warning box will remind to publish the new version of customizations to your Vault Server.
Afterwards, the new powerEvents version can also be installed on all other environments without notifying Vault users again about the already confirmed changes.

Note: When upgrading from versions *v23.0.9* or earlier with **enabled** *sample scripts* these scripts are automatically reverted to their original state after the update.

To *re-enable sample scripts* that have been enabled manually on the previous install, move them from *%ProgramData%/coolOrange/Client Customizations/Disabled* to *%ProgramData%/coolOrange/Client Customizations* .

1.6 Uninstall

To remove powerEvents from a computer one perform of the following steps :

- Execute the setup file again. This will give you the option to repair or remove powerEvents. Click on “Remove” to uninstall the program.
- Go to “Control Panel - Programs and Features”, find “coolOrange powerEvents for Vault” and run “Uninstall”.

CHAPTER

TWO

ACTIVATION AND TRIAL LIMITATIONS

2.1 Trial limitations

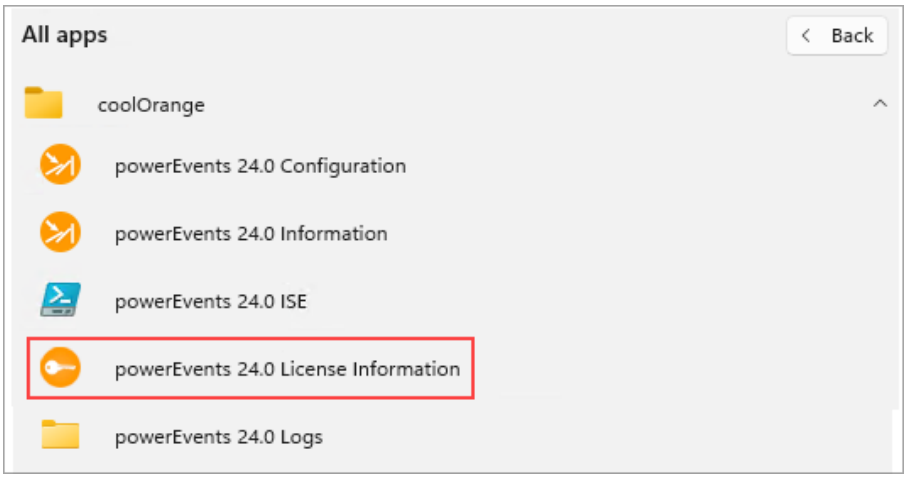
There is no difference in functionality between the trial version and the fully licensed product.
After the installation the product is available as a trial version for *30 days*.

2.2 Activation

The product can be activated during or after successfully evaluating the product.
For *manually activating* the product following Dialog can be used:

2.2.1 License Information

Open the Start Menu and click the '*powerEvents 23.0 License Information*' shortcut:



2.2.2 Command-line

Launch the License Information tool located in the install directory with the required Command-line arguments.
Example: Activating a Stand-Alone license using a serial number:

```
"C:\Program Files\coolOrange\Modules\powerEvents\License.exe" --StandAlone --
↪Serialnumber="XXXXX-XXXXX-XXXXX-XXXXX"
```

For more information about activating the product, see Licensing.

2.3 Licensing Options

2.3.1 Stand Alone Licensing

This product supports the Stand-Alone licensing model which is charged based on the time the license is valid and the number of seats the license is valid for.
For further information see the detailed description of the Stand-Alone licensing model.
In the License Information Dialog the *remaining days* until the license expires can be found.

License expired

When the license expires, powerEvents displays a Windows notification about the expired license when the application is started and no longer executes actions:

- Vault Client tabs added by the *Add-VaultTab* cmdlet stop working and no longer display the content from the *-Action* parameter.
- Vault Client and Inventor menu items added by the *Add-VaultMenuItem* cmdlet or the *Add-InventorMenuItem* cmdlet stop working and no longer execute the *-Action* parameter.
- All *registered Vault events* are restricted in order to block the configured processes of the Vault user.

Offline activation

The serial number of the license and the machine code are required to generate an activation file.
The activation file for an offline activation can be generated and downloaded on the following site: powerEvents - Activation file generator

CHAPTER

THREE

GETTING STARTED

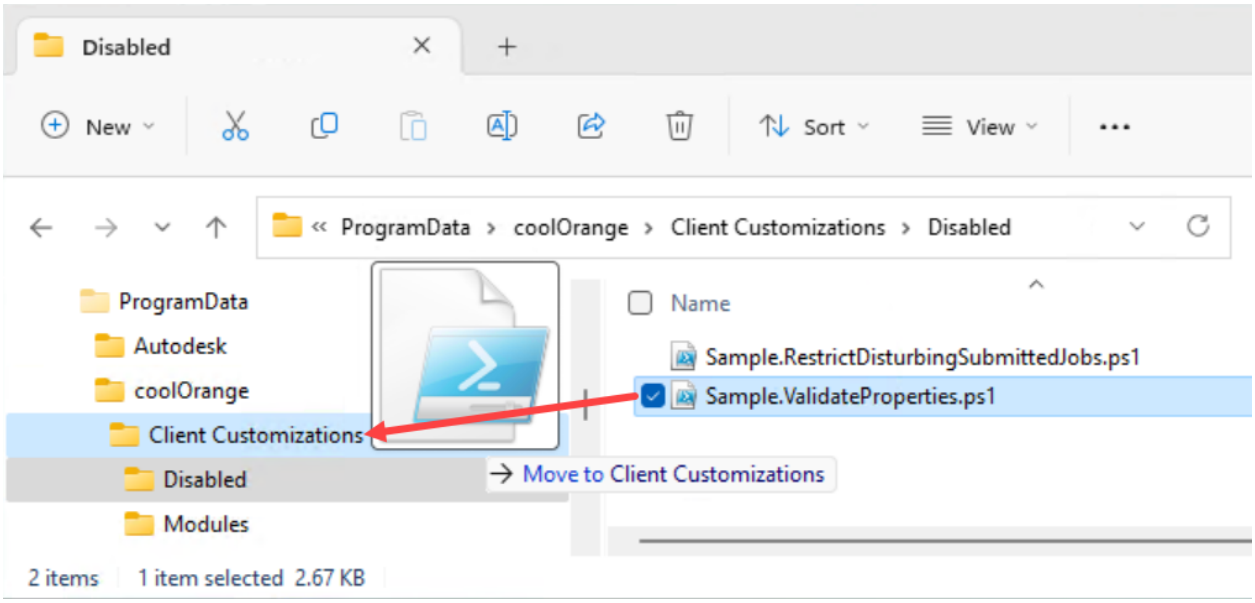
3.1 Sample scripts

Installed *Client Customizations* demonstrate possible automation processes and guardrails when working with Vault applications. They let you see the capabilities of the product in common scenarios.
The sample scripts are customizable and can be used as a template to easily create your own scripts to fit specific customer requirements.

One of these samples, Sample.ValidateProperties is intended to show how custom code can be executed on Vault lifecycle state transitions.

3.2 Activating a sample script

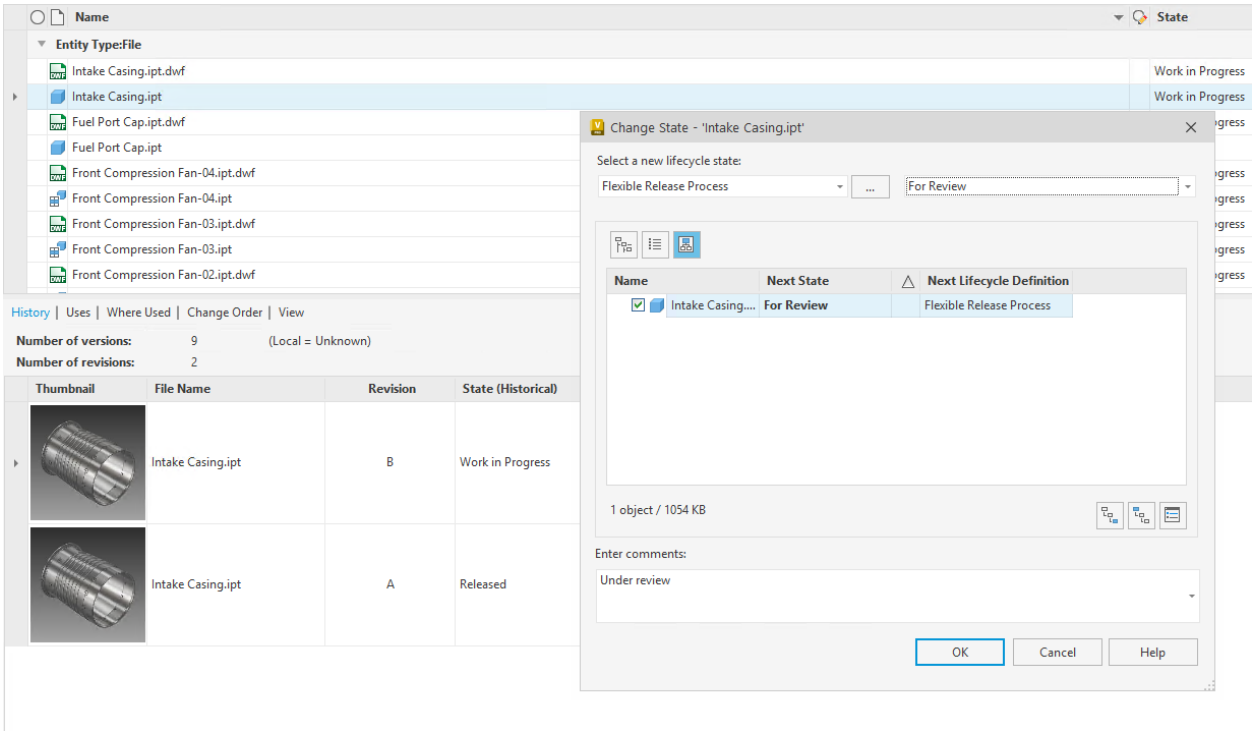
Sample scripts are disabled by default.
However, thanks to their modular structure, an individual script like *C:\ProgramData\coolOrange\Client Customizations\Disabled\Sample.ValidateProperties.ps1* can be easily enabled.
Just **move it** to the directory *C:\ProgramData\coolOrange\Client Customizations*:



3.3 Testing the script

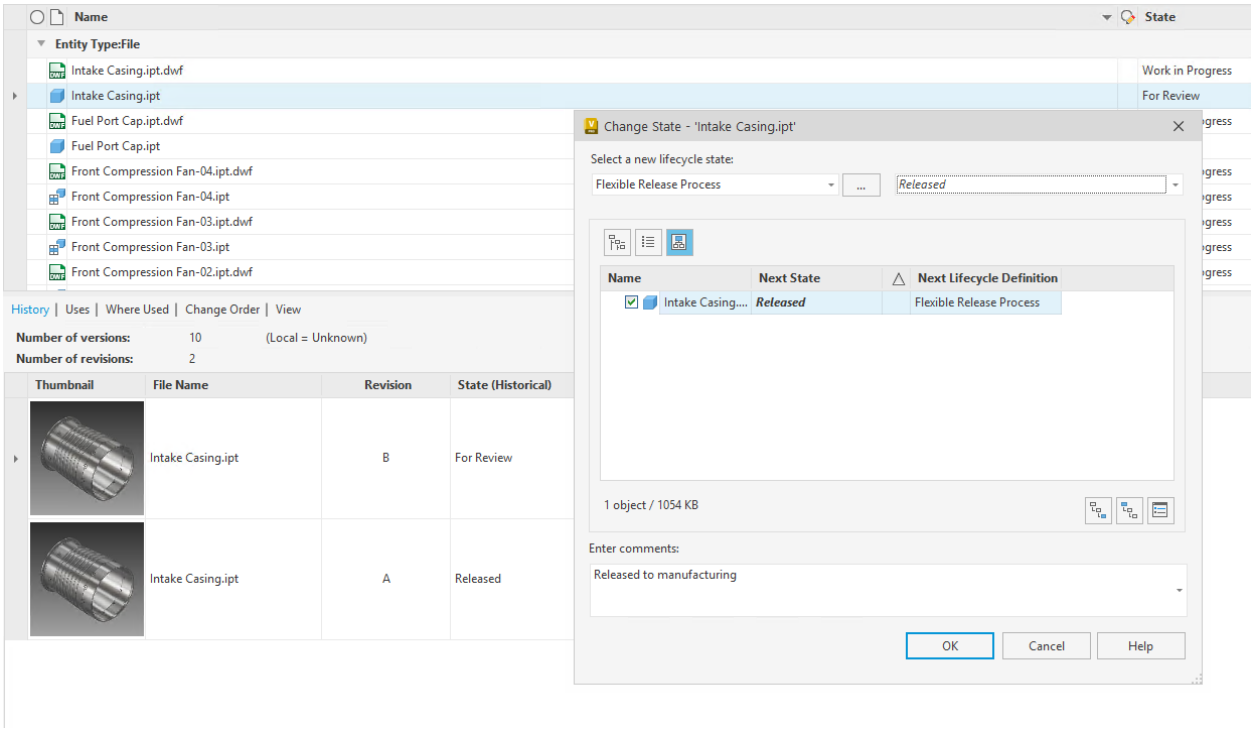
The goal of the **Sample.ValidateProperties** script is to prevent a user from releasing a file that has recently been changed by an other user.
Testing this customization requires **two different Vault accounts**.

Start the Vault Client and log in with the first account (in this example “Administrator” is used).
Navigate to a file and change the lifecycle state to a non-release state.
In this example the state of **Intake Casing.ipt** is changed to **“For Review”**:



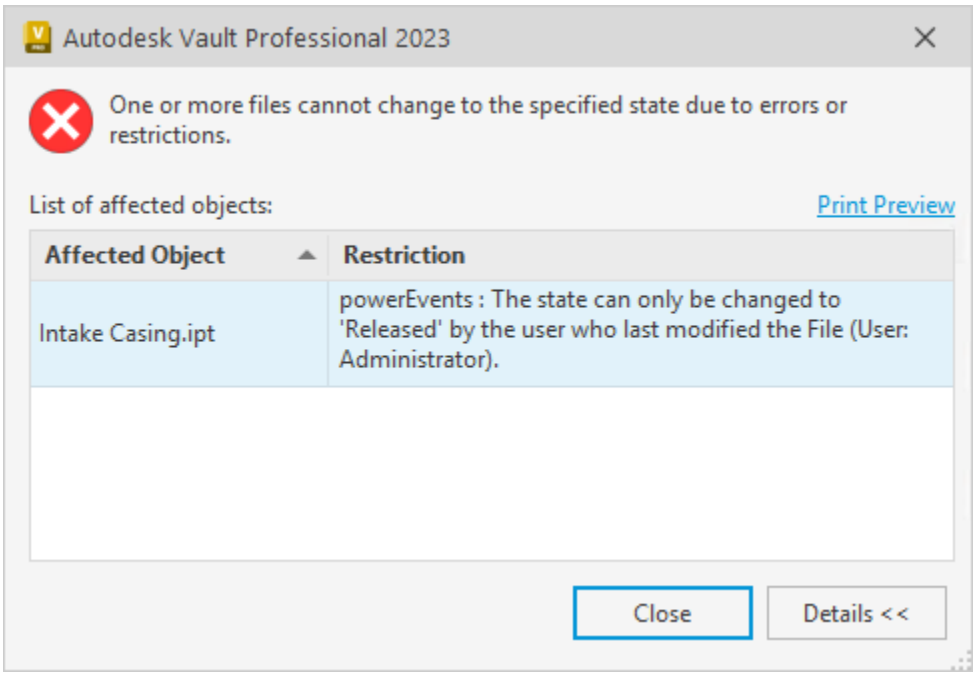
Changing Vault account
Log out by going to “File” -> “Log Out” and log in again with the **second** Vault account.

Releasing file
Navigate back to previously changed file and try to change its lifecycle state to a released state. In this example the state of **Intake Casing.ipt** is changed to **“Released”**:



Result

The lifecycle transition for **Intake Casing.ipt** to the **"Released"** state is blocked by a powerEvents restriction because the latest modification was performed by a different Vault account ("Administrator" in this example) and the second Vault account is now trying to release the file:



CHAPTER

FOUR

CLIENT CUSTOMIZATIONS

4.1 Sample.RestrictDisturbingSubmittedJobs

Autodesk Vault environments give users the possibility to submit Jobs while they work with their files, but they are not always processed immediately.
As a result, it often happens that they either fail or don't produce the expected results if users continue to work with their files in Vault.

To prevent these problems, this client customization helps for example in the following situations:

- When new file versions are created after jobs have been submitted for that file, many Autodesk jobs simply fail.
  A common error that you can then see is e.g. "Sync properties not allowed on non-tip versions".
  Autodesk recommends "Administrators can ignore, delete, or filter out this type of error."

  Solution:
  So that it doesn't even come to the situation that all the jobs fail and fill up the Job Queue, Vault users can also be informed and restricted from creating new file versions, as long as there are still open jobs in the Job Queue.

- If powerJobs Processor jobs have been queued at certain lifecycle transitions, such as releases, and in the meantime the status of the file has been reset or changed (e.g. to "Work in Progress") then many Vault users forget about their still open jobs within the Job Queue.
  By default those jobs will be executed for the latest file version, and newly created visualization files may contain incorrect data (e.g. State) and are not attached to the released file version.

  Solution:
  To prevent this from happening, Vault users can also be prevented from changing the file status if jobs must be processed beforehand.

Therefore this sample script registers to the *UpdateFileStates* event and adds **Restrictions** when the lifecycle state of a file changes, while jobs were queued for it.
This way Lifecycle transitions are prevented for all files, for which jobs have been submitted and have not yet been processed successfully.

4.1.1 Testing

In the default delivery the script is **disabled** and located in the *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled* folder.
The customization can be tested by following these steps:

1. Enable the script by moving it to the directory *%PROGRAMDATA%\coolOrange\Client Customizations*.
2. Pause all active Job Processors during the test to ensure that the queued Synchronize Properties job will not be processed upfront.
3. Open the Vault Client.
4. Make sure the action "Synchronize properties using Job Server is enabled for a desired Lifecycle transition to a "Released" state.
5. Navigate to an Inventor- or AutoCAD file and change its Lifecycle State to the previously configured "Released" state
6. Open the Job Queue and make sure that a *"Autodesk.Vault.SyncProperties"* job has been automatically submitted for the released file version.
7. Change the Lifecycle State of the selected file again to any other state.
8. A restriction dialog appears and informs about the still active Synchronize Properties job, which would fail if a new file version gets created.

4.2 Sample.ValidateProperties

This sample script registers to the UpdateStates events for the entity types: *File*, *Item* and *ChangeOrder*.
The event is triggered when the lifecycle state of an entity is changed.

In our example we make use of the **Restrictions event**:

- In the Restrictions event we check if the state is about to be changed to Released or in case of a ChangeOrder to Closed.
- We also check if the user who is changing the state is the same who last modified the entity. If one of these rules is violated a *restriction* will be set.

4.2.1 Testing

In the default delivery the script is **disabled** and located in the *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled* folder.
The customization can be tested by doing the following steps:

1. Enable the script by moving it to the directory *%PROGRAMDATA%\coolOrange\Client Customizations*.
2. Open the Vault Client.
3. Navigate to an Item, File or Change Order which can be released and was last modified by another Vault user.
4. Change the Lifecycle State to "Released" (or "Closed" for Change Orders).
5. A restriction dialog appears informing the user that the Lifecycle State of the entity can't be released (or closed) as it was most recently edited by another user.

4.3 SubmitJobsOnLifecycleTransition

This client customization is intended to be used in combination with the powerJobs Settings Dialog to configure automatic queueing of jobs based on certain lifecycle transitions and filter conditions.

Requires powerJobs Processor

The **SubmitJobsOnLifecycleTransition** script relies on the configuration from the Job Triggers section of the powerJobs Settings Dialog to work properly.
This requires minimum version of powerJobs Processor 23.0.1

The user performing the action must have the *Vault Get Options* permission to retrieve the trigger settings configured with the powerJobs Settings Dialog.

This script makes use of the following Vault events:

- *UpdateFileStates* event which is triggered when the lifecycle state of a **File** changes
- *UpdateItemStates* event which is triggered when the lifecycle state of an **Item** changes
- *UpdateChangeOrderState* event which is triggered when the lifecycle state of a **Change Order** changes
- *UpdateCustomEntityStates* event which is triggered when the lifecycle state of a **Custom Object** changes

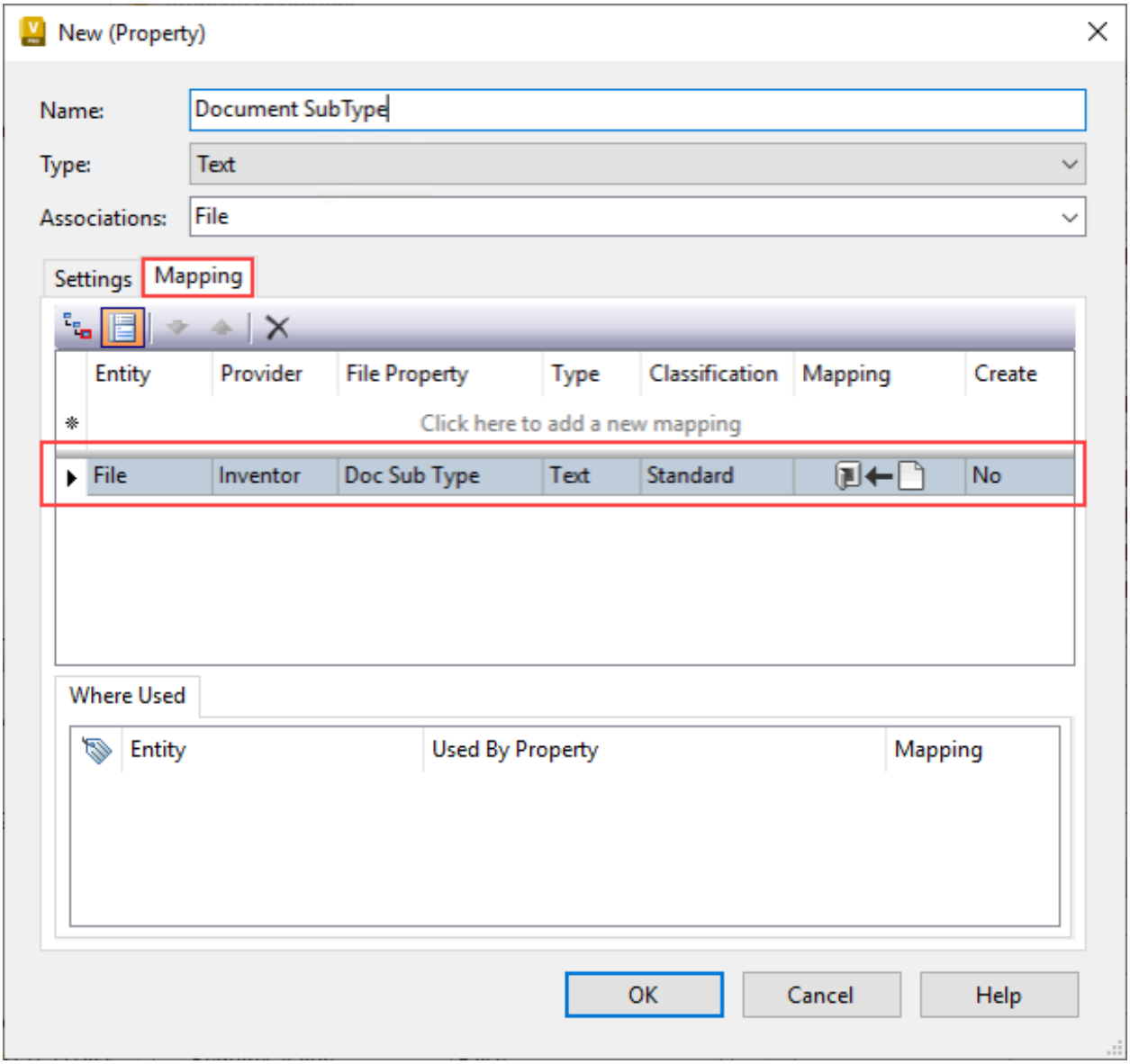
The script registers **Post Event** handlers for every above mentioned event and **queues the configured job** if an entity matches all of the following conditions:

- The 'update status' operation was successful
- One or more of the affected entities transitioned to a configured lifecycle state
- All of the configured filters match for an entity
- If the powerJobs Processor *Sample.CreateDXF&STEPfromSheetmetal* job is configured to be queued, additional checks are performed.
  The Vault environment must be *configured* correctly for the script be able to determine the document type.

4.3.1 Vault Configuration

In order to identify sheet metal parts a Vault UDP (User defined property) must be configured to differentiate between document types (i.e. Sheet Metal or regular part, Weldment or regular Assembly).
The User Defined Property must be mapped to the Inventor iProperty `Document Sub Type Name` (only works with an English or German Inventor/Vault) or to the Inventor iProperty `Document Sub Type` which contains the required information.
The name of the User Defined Property is automatically determined by script.

To configure the required mapping in Vault, navigate to *Tools -> Administration -> Vault Settings -> Behaviors tab -> Properties -> New*
Define the name for the new UDP (e.g. *Document SubType*) and select the categories used for Sheet Metal parts, so it will be automatically listed in the properties panel.
Switch to the *Mappings* tab and create a new mapping for the provider "Inventor" and select the `Doc Sub Type` or `Doc Sub Type Name` (only works with an English or German Inventor/Vault) Inventor property.



Autodesk Knowledge Tutorial: Mapping Inventor iProperty to Vault user defined property

4.3.2 Testing

This customization can be tested by performing the following steps:

1. Open the Vault Client, open the powerJobs Settings Dialog and navigate to the Job Triggers section of the dialog. Configure a job to be queued on a lifecycle transition.
2. Navigate to an entity of the selected type (e.g. file)
3. Change the Lifecycle State to the configured state
4. Open the Job Queue
5. The Job Queue contains a configured job for the changed file with the set description and priority

Out of the box this client customization is **enabled**.

When the script is not used it can by disabled by moving it to the *%PROGRAMDATA%\coolOrange\Client Customization\Disabled* folder.

4.4 SubmitJobsOnVaultMenuItemClick

This client customization is intended to be used in combination with the powerJobs Settings Dialog to configure the Vault Explorer to queue jobs directly from the Vault Context Menu for Files, Items and Change Orders.

Requires powerJobs Processor

The **SubmitJobsOnVaultMenuItemClick** script relies on the configuration from the Job Triggers section of the powerJobs Settings Dialog to work properly.
This requires minimum version of powerJobs Processor 23.0.3

The user that is logged in to Vault must have the `Vault Get Options` permission to access the context menus configured with the powerJobs Settings Dialog.

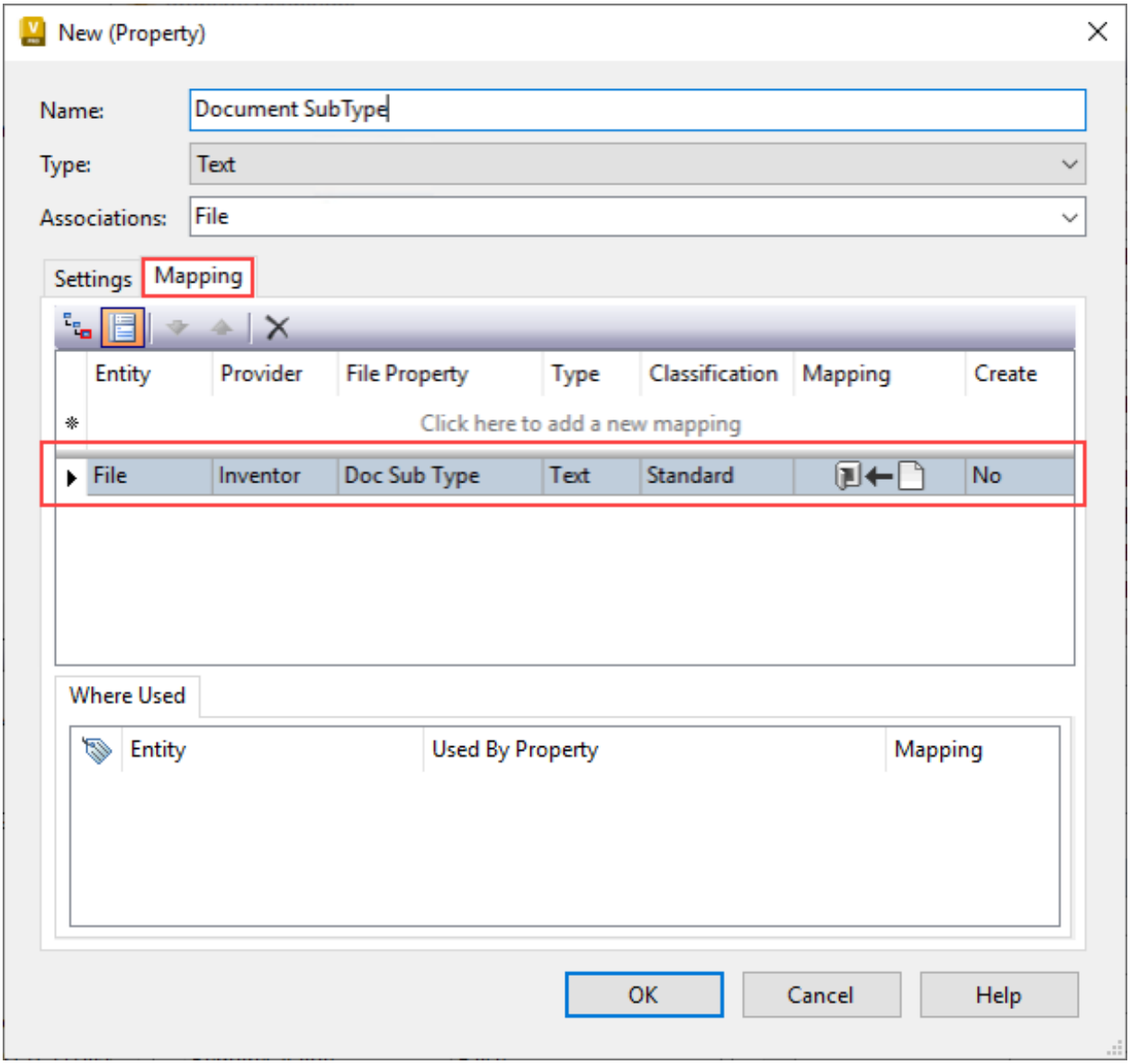
The script action gets executed on the *LoginVault* event and makes use of the *Add-VaultMenuItem* cmdlet to add context menu items for each job that has a context menu trigger enabled in the Job Triggers section in the powerJobs Settings Dialog.

- When the user clicks the menu item, jobs are only queued when the selected entity matches all of the filters configured in the powerJobs Settings Dialog
- The *SubmitJobsOnVaultMenuItemClick* script is only executed when used in the Vault Explorer.
- A dialog is displayed when none of the selected entities matches the filters and therefore no jobs were queued.
- If the powerJobs Processor *Sample.CreateDXF&STEPfromSheetmetal* job is configured to be queued, additional checks are performed.
  The Vault environment must be ***configured*** **correctly** for the script be able to determine the document type.
- The script is only executed once, when the user logs in to Vault for the first time.

4.4.1 Vault Configuration

In order to identify sheet metal parts a Vault UDP (User defined property) must be configured to differentiate between document types (i.e. Sheet Metal or regular part, Weldment or regular Assembly).
The User Defined Property must be mapped to the Inventor iProperty `Document Sub Type Name` (only works with an English or German Inventor/Vault) or to the Inventor iProperty `Document Sub Type` which contains the required information.
The name of the User Defined Property is automatically determined by script.

To configure the required mapping in Vault, navigate to *Tools -> Administration -> Vault Settings -> Behaviors tab -> Properties -> New*
Define the name for the new UDP (e.g. *Document SubType*) and select the categories used for Sheet Metal parts, so it will be automatically listed in the properties panel.
Switch to the *Mappings* tab and create a new mapping for the provider "Inventor" and select the `Doc Sub Type` or `Doc Sub Type Name` (only works with an English or German Inventor/Vault) Inventor property.



Autodesk Knowledge Tutorial: Mapping Inventor iProperty to Vault user defined property

4.4.2 Testing

The customization can be tested by performing the following steps:

1. Open the Vault Client, open the powerJobs Settings Dialog and navigate to the Job Triggers tab of the dialog. Then navigate to the *Context Menu Trigger* section and enable it.
2. Restart the Vault Client for the changes to take effect
3. Navigate to an entity of the selected type (e.g. File)
4. Right-click the entity, expand the sub-menu “powerJobs Client” and click on the menu item to queue the job
5. To verify the job Job Queue can be opened and it should contain a job for the selected entity with the configured job type, description and priority

Out of the box this client customization is **enabled**.
When the script is not used it can by disabled by moving it to the *%PROGRAMDATA%\coolOrange\Client Customization\Disabled* folder.

As an add-in for Vault Client and CAD applications, powerEvents enables easy and powerful customization of Vault functionality with additional workflows, guardrails, automations, tabs and more.

This is done via various PowerShell *scripts* and *modules* which are *executed in every Vault application*.
The directory containing these customizations can be opened using the *powerEvents Configuration* shortcut on the desktop or in the Start Menu.

Since powerEvents is an *IWebServiceExtension* all the customization scripts will be executed in any application that uses the *Autodesk.Connectivity.WebServices.dll*:

- Vault Client
- Inventor with Vault Add-in
- AutoCAD with Vault Add-in
- AutoLoader
- powerVault
- *at the moment also in the JobProcessor and powerJobs Processor

4.5 Scripts

All PowerShell scripts, which should be executed automatically when Vault applications are launched, must be placed in the *%PROGRAMDATA%\coolOrange\Client Customizations* directory.
This directory contains scripts that ensure all settings in the powerJobs Settings Dialog for submitting jobs, also take effect on the workstations:

- *SubmitJobsOnLifecycleTransition.ps1*
- *SubmitJobsOnVaultMenuItemClick.ps1*

In addition the directory consists of two sample scripts, both starting with the name '*Sample.*'. They are disabled by default and are therefore located in a subdirectory named *'Disabled'*:

- *Sample.RestrictDisturbingSubmittedJobs.ps1*
- *Sample.ValidateProperties.ps1*

Their purpose is to demonstrate possible automation processes when working with the Vault Client and to help you get started creating your own scripts.

The goal of all these customization scripts is to cover some common workflows and be easy to configure.

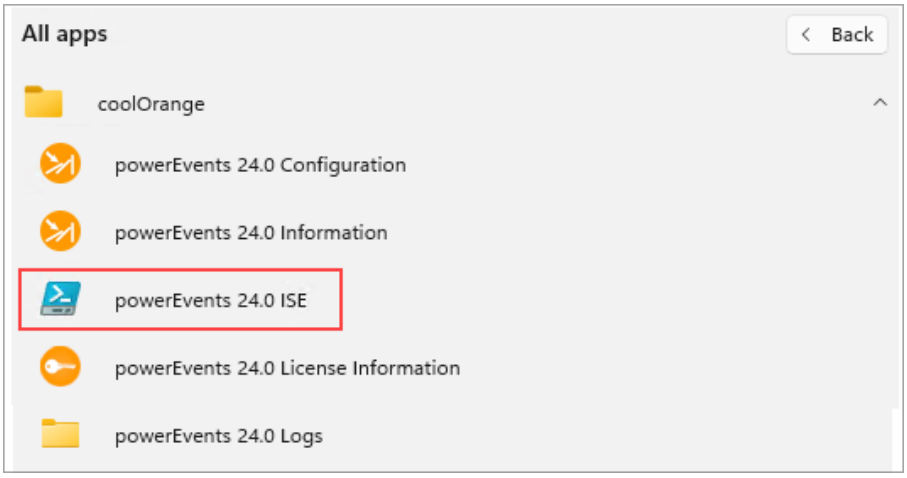
4.5.1 Enable or Disable scripts

The modular structure of scripts allows to easily *disable or active* individual customizations.
To do this, the corresponding script can simply be moved to the *%PROGRAMDATA%\coolOrange\Client Customizations\Disabled* directory to turn it off, or back to the *Client Customizations* folder to re-enable it.
Scripts placed in the *Disabled* folder remain disabled after product *updates*.

4.5.2 Create or edit scripts

When creating your own client customization, it is recommended to copy a sample script and modify it to your needs. It is not necessary to restart the application, because all customizations (scripts and *modules*) are *automatically reloaded* by default when a change is detected.

In order to get started with creating your own script either open any PowerShell IDE or the *powerEvents ISE* shortcut in the start menu, which already opens one of the *sample scripts*.
The powerEvents module can be imported with `Import-Module powerEvents`.



Keep in mind, that code in the scripts is executed while the Vault application is launched, and thus no Vault connection is available.
A Vault connection (required for powerVault Cmdlets) is only available during the execution of a *registered* Vault event, in Vault *tab* actions, or when you click buttons in the Vault Client *menu* or Inventor *ribbon*.
Therefore, the use of most powerVault Cmdlets is possible only in such *-Action* script blocks or functions.

4.5.3 Debug scripts

As mentioned above, individual scripts can be opened for debugging in the Windows PowerShell ISE and then simply executed by pressing F5.

It is also possible to attach the debugger to another process that hosts PowerShell. This allows debugging customization scripts that run in a Vault Client or CAD application.
Here, the Windows PowerShell ISE is used again, as this tool is already integrated into Windows environments. However, the informations are also helpful when using other PowerShell IDEs such as Visual Studio Code.
In the Console Pane, run the following commands to attach to a running Vault Client process:

```
Enter-PSHostProcess -Name 'Connectivity.VaultPro' # or first run Get-PSHostProcessInfo␣
↪to get the list of PowerShell hosts running on the local computer
```

```
Debug-Runspace -Name 'coolOrange'
```

For subsequent script executions, the debugger stops automatically in the according customisation script. Note: After the execution, however, the debugger is detached and must be readded to pause in additional executions (STRG+C and repeat Debug-Runspace).

Alternatively, by adding the following line in a problematic script area, the debugger pauses exactly there as soon as this change is *reloaded*:

```
if([System.Management.Automation.Runspaces.Runspace]::DefaultRunSpace.Debugger.IsActive)
↪{ Wait-Debugger }
```

Debugging can then be continued in single-step mode (F11) or new breakpoints (F9) can be reached after continuing script execution (F5), allowing to inspect the values of current variables.

4.6 Modules

PowerEvents is delivered with module scripts which are installed in the powerEvents module directory *%ProgramData%\coolOrange\Client Customizations\Modules*.

The *Common.psm1* module provides the **$processName** variable which is available in all customization scripts.
This variable returns the name of the process in which the script is currently executed.
The global flag **$powerEvents_ReloadPsScripts** can be used to disable the *automatic script reloading*:

```
$global:powerEvents_ReloadPsScripts = $false
```

The *JobTriggerSettings.psm1* modules provides functions required in the ***SubmitJobsOnLifecycleTransition.ps1*** and ***SubmitJobsOnVaultMenuItemClick.ps1*** scripts.
The *IsSheetMetalPart()* function can be used to check whether a file is sheet metal part. In order for the function to check this, at least one of the following two Inventor iProperties must to be mapped to a Vault property:

- 'Document SubType Name'
- 'Document SubType'

The respective *Vault Configuration* sections describe in detail how the Inventor iProperties should be mapped in Vault.

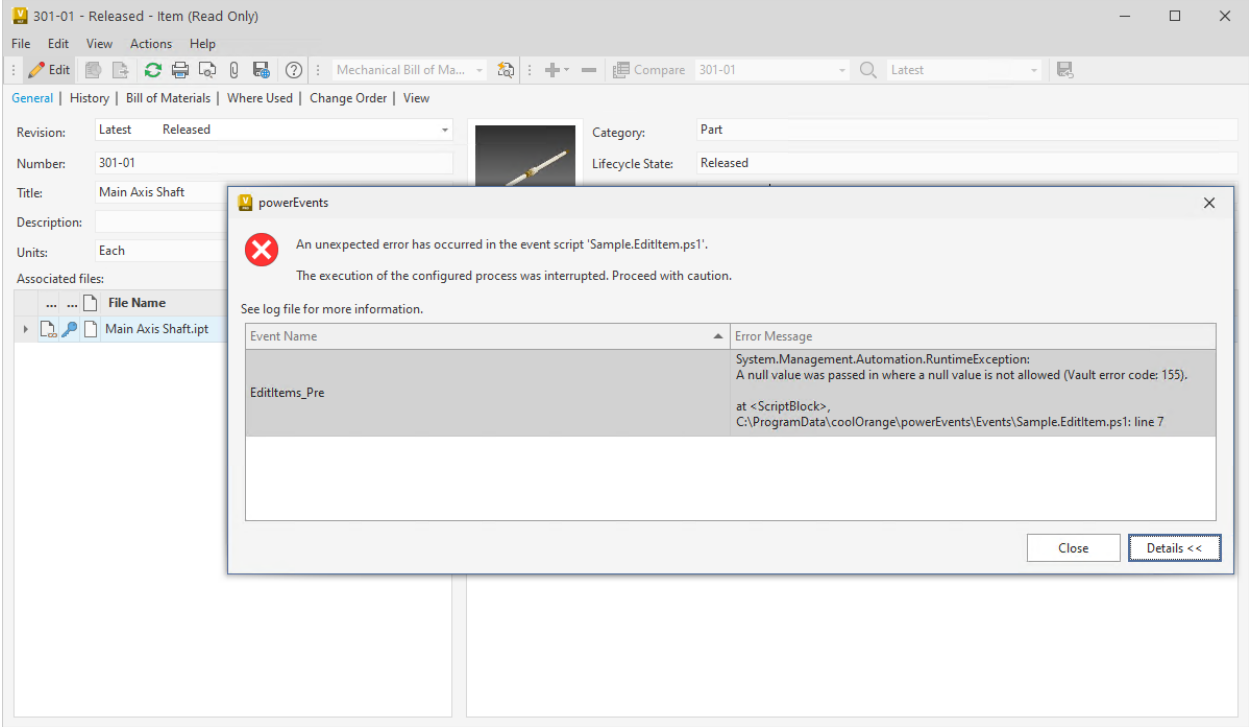
4.7 Errors

powerEvents *notifies* the Vault user about Terminating Errors that occur during application startup, when executing customization *Scripts* and *Modules*.
In this case the configured processes might not be executed successfully, because of missing Vault event registrations.
Even if the execution of one script fails, this has no effect on other scripts as they are executed independently.

In order to raise exceptions manually you can use Powershell's throw keyword and you can handle them by using *try/catch blocks*:

```
Register-VaultEvent -EventName EditItems_Pre -Action {
        try {
                $vault.ItemService.EditItems($null)
        }
        catch {
                throw "A null value was passed in where a null value is not allowed␣
↪(Vault error code: 155)."
        }
}
```

Also if a *registered Vault event* is raised, and only then an exception occurs within the action (as in the previous example), an Error Message Box will be shown.
Details of the failed event execution are displayed directly after the PowerShell execution terminates:



This happens every time an event is triggered by a Vault API method and has no effect on any other registered Vault events as they will still be executed.

The same details and all the Warnings and Errors that where logged during the script execution can be found in the *logfile*.

Note that PowerShell executions continue for Non-Terminating Errors by default.
For changing this error handling behavior *globally*, the variable *$ErrorActionPreference* can be changed to 'Stop' in order to terminate the script execution even for such errors:

```
$ErrorActionPreference = "Stop"
```

In case a Vault operation aborts because a *Vault Server Error Code* was returned the according *Post event* can make use of the `$successful` variable in order to check if the underlying Vault Web Service call was successful or not.

```
Register-VaultEvent -EventName EditItems_Post -Action {
        param( $items, $successful)

        if(-not $successful) {
                return
        }
        #...
}
```

4.8 Distribution

Once the local client customisations have been successfully tested, they can be easily shared to other Vault workstations by **installing** the coolOrange Customization Server on the Autodesk Data Management Server and *powerEvents* on all workstations.
Then simply run the script **Publish-Customizations.ps1** in the directory *C:\ProgramData\coolOrange*, which publishes all these scripts and modules for this Vault Server.
From this point on, your client customizations are **automatically downloaded** to all workstations as soon as the Vault Client or any other application that connects to the Vault Server is launched.

The distribution mechanism is based on the popular version control system Git, but does not require a git installation. Simplified functions for publishing and synchronizing customizations are even designed to help Git-experienced scripting guys.

Publish Customizations:
By running the *Publish-Customizations* script, the local *C:\ProgramData\coolOrange* directory is automatically linked to the Git repository http://{Your_Vault_Server}/customizations/coolOrange.git.
By default, the Vault Server against which the current customizations were tested is used, meaning the currently (or lastly) connected Vault Server.
To directly publish from any PowerShell IDE, the following command can be used:

```
Import-Module powerEvents

Publish-Customizations

# OR to use a specific Server instead of the lastly connected ADMS or the currently␣
↪connected 'Open-VaultConnect -Server' (e.g. for automations where the Vault Client has␣
↪perhaps never been logged on to a server before):
Publish-Customizations -Server {Your_Vault_Server}
```

Note that only files located in the directory *%PROGRAMDATA%\coolOrange\Client Customizations* are published! Other product directories (such as Jobs) are ignored.
After distribution, each Vault application launch informs about changes in this directory that may still need to be published.
This way, local fixes or improvements to client customizations can be easily distributed from any Vault environment by running the publishing script.
Also if newer script or module versions have already been published by teammates, it helps to ensure that these are retrieved without changes being lost or accidentally overwritten. If conflicts occur, they have to be resolved and tested before the changes can be re-published.
Git tools and Visual Studio Code integrations offer excellent support here in particular.
After executing the *Publish-Customizations* script once, all the necessary settings are made (a *.git* folder is created) to use all these tools seamlessly.

Synchronizing Customizations:
During the start of Vault applications, all local scripts and modules are updated to the latest version available on the connected Vault Server.
The synchronised changes will then take effect immediately and are therefore used even if the coolOrange Customization Server would become unavailable for some reason.
In the case such distribution problems occur, Vault users immediately recognise an Error Message Box with additional details, that help in resolving the issue effectively.

The synchronization also allows published customizations to be automatically distributed to new workstations simply by *installing powerEvents*.
Right at the first Vault start, all default-installed sample scripts and modules (as well as the standard scripts for powerJobs Client), will be replaced with your published customizations.

Please also see our *update notes* for further details.

Warning: Please do not **uninstall** the coolOrange Customization Server on the Autodesk Data Management Server, for example, after a project Go-Live.
All Vault environments that have retrieved client customization from that Vault Server in the past, would warn the Vault users about the failing distribution!

Stopping a synchronization is only possible by removing the *C:\ProgramData\coolOrange\.git* directory on all workstations.

CHAPTER

FIVE

CODE REFERENCE

5.1 Cmdlets

5.1.1 Add-InventorMenuItem

Add a menu item to an Inventor menu with an Action that is invoked when the menu item is clicked.

Syntax

```
Add-InventorMenuItem -Name <String> -Action <Scriptblock | String> [<CommonParameters>]
```

Parameters

Type	Name	Description	Default Value	Optional
String	Name	Display name of the menu item displayed in the Inventor tab		no
Scriptblock / String	Action	Script block or function name that is executed when the menu item is clicked.		no

Warning: ***Name*** **must be unique**

When multiple menu items are registered, the *Name* parameter must be **unique** for each menu item.

Return type

empty

Remarks

The cmdlet can be used to extend the Inventor user interface with a new menu item which is displayed in a **coolOrange Tab**.
The **coolOrange Tab** and its menu items will be displayed in the following Inventor Ribbons:

- Part
- Assembly
- Drawing
- Presentations

The script block or function name passed to the `Action` parameter is invoked when the menu item is clicked.
The PowerShell variables provided by Open-VaultConnection and a *`$inventor` variable* are available during the execution of the registered action to allow access to the Vault and Inventor APIs.

Menu items created by the cmdlet are **not permanent**, when Inventor is restarted the menus are reset to their original state.

Calling the cmdlet with the same *Name* parameter after a menu item has already been added, will **update the Action** for the menu item.

Note:

- The cmdlet can **only be used** when running directly in an **Inventor process**.
- The cmdlet is **executed only** after a successful **login to Vault**, which can be done via the Inventor Vault Add-in.

Examples

Adds a menu item to add a virtual compontent with an overridden Quantity to the structured BOM:

```
Add-InventorMenuItem -Name "Add oil to BOM" -Action {
        $compName = 'Oil'
        $document = $inventor.ActiveDocument
        $occur = $document.ComponentDefinition.Occurrences
        try{
            $virtualComponent = $occur.AddVirtual($compName, $inventor.TransientGeometry.
↪CreateMatrix())
        }catch {
            Write-Host "Could not add virtual component!"
            return
        }
        $BOM = $document.ComponentDefinition.BOM

        if (-not $BOM.StructuredViewEnabled) {
            $BOM.StructuredViewEnabled = $True
        }
        $structBomView = $BOM.BOMViews | Where-Object {$_.ViewType -eq [Inventor.
↪BOMViewTypeEnum]::kStructuredBOMViewType } | Select-Object -First 1
        $bomCom = $structBomView.BOMRows | Where-Object {($_.ComponentDefinitions |␣
↪Select-Object -First 1).DisplayName -eq $compName}
        $bomCom.TotalQuantity = '10'
}
```

Adds a menu item to create new Vault Change Order with current document attached:

```
Add-InventorMenuItem -Name "Create changeorder" -Action {
    $docName = $inventor.ActiveDocument.DisplayName
    $file = Get-VaultFile -Properties @{'Name' = $docName}
    if($null -eq $file){
        Write-Host "Could not find vaulted file for current document: '$docName'"
        return
    }
    $coNumScheme = $vault.NumberingService.GetNumberingSchemes('CO', [Autodesk.
↪Connectivity.WebServices.NumSchmType]::ApplicationDefault) | Select-Object -First 1
    $coNumber = $vault.ChangeOrderService.GetChangeOrderNumberBySchemeId($coNumScheme.
↪SchmID)
    $co = Add-VaultChangeOrder -Number $coNumber
    Write-Host "Created ChangeOrder with number '$coNumber'"
    if($null -eq (Update-VaultChangeOrder -Number $co._Number -AddAttachments @($file._
↪FullPath))){
        Write-Host "Attaching file to changeorder failed"
    }
}
```

5.1.2 Add-VaultMenuItem

Add a Menu Item to a Vault Explorer menu with an Action that is invoked when the menu item is clicked.

Syntax

```
Add-VaultMenuItem -Location <VaultExplorerMenu> -Name <String> -Action <Scriptblock |␣
↪String> [<CommonParameters>]
```

Parameters

Type	Name	Description	Default Value	Optional
VaultExplorerMenu	Location	Vault Explorer Menu to be extended with the Menu Item		no
String	Name	Display name of the menu item displayed in the Vault Explorer		no
Scriptblock / String	Action	Script block or function with one parameter (`$entities`) that is executed when the menu item is clicked.		no

VaultExplorerMenu

Enum values:

Name	Description
FileContextMenu	Context menu (right-click menu) for Files
ItemContextMenu	Context menu (right-click menu) for Items
ChangeOrderContextMenu	Context menu (right-click menu) for Change Orders
FolderContextMenu	Context menu (right-click menu) for Folders
ToolsMenu	Tools menu in Vault's Main Menu toolbar

Warning: *Name* must be unique

When multiple menu items for the same *Location* are registered, the *Name* parameter must be **unique** for each menu item.

Return type

empty

Remarks

The cmdlet can be used to extend the Vault Explorer's menus with custom Powershell code.
The script block or function passed as the `Action` parameter is invoked with an array of powerVault entities as parameter, which contains all selected entities selected in the Vault Client.

Contex menu items
Added menu items are grouped into a sub-menu named "*powerJobs Client*".
Depending on the Location of the menu item the objects passed to the `-Action` can be:

- `FileContextMenu`: Only File objects
- `ItemContextMenu`: Only Item objects
- `ChangeOrderContextMenu`: Only Change Order objects
- `FolderContextMenu`: Only Folder objects

Tools menu items
Added menu items are appended directly to Vault Explorer's **Tools** menu.
Depending on the current selection in the Vault Explorer the object passed to the `-Action` can be powerVault files, items or change orders.
Only entities selected in the main Vault Explorer window are passed to the `-Action`. The selection in other windows e.g. in the Search dialog or Edit Item window is ignored.

Menu items created by the cmdlet are **not permanent**, when the Vault Client is restarted the menus are reset to their original state.
The Menu items cannot be removed nor can the name be updated once they have been added.

Calling the cmdlet with the same *Location* and *Name* parameter after a menu item has already been added, will update the **Action** for the menu item.

Note: The cmdlet can **only be used** when running directly in a **Vault Explorer Process**

Examples

The following samples are meant to be saved as *.ps1* files in the *Client Customizations* folder.

Adds a menu item to the File context menu to queue a DWF job if the file type is supported:

```
function SubmitDWFJob($entities){
    foreach($file in $entities){
        if($file._Extension -in @('iam', 'ipt', 'ipn', 'idw', 'dwg', 'dxf', 'rvt', 'rfa',
↪ 'rte', 'nwd', 'nwf', 'nwc', 'ifc')){
            Add-VaultJob -Name "Autodesk.Vault.DWF.Create.$($file._Extension)" -
↪Parameters @{'FileVersionId' = $file.Id} -Priority 10
        }else{
            [System.Windows.Forms.MessageBox]::Show("File extension '$($file._Extension)
↪' is not supported!")
        }
    }
}
Add-VaultMenuItem -Location FileContextMenu -Name 'Queue DWF job' -Action 'SubmitDWFJob'
```

Adds a menu item to the Item context menu to queue the powerPLM (powerFLC) 'Sample.TransferItemBOMs' job:

```
Add-VaultMenuItem -Location ItemContextMenu -Name "Transfer Item BOM to FLC" -Action {
    param($entities)
    foreach($item in $entities){
        Add-VaultJob -Name "Sample.TransferItemBOMs" -Parameters @{"EntityId"=$item.Id;
↪"EntityClassId"="Item"} -Description "Transfer Item BOMs for: $($item._Number)"
    }
}
```

Adds a menu item to the Folder context menu to queue the powerJobs Processor 'Sample.CreatePDF' job for all files in the folder:

```
Add-VaultMenuItem -Location FolderContextMenu -Name "Create PDF's for all files in folder
↪" -Action {
    param($entities)
    foreach($folder in $entities){
        $files = Get-VaultFiles -Folder $folder._FullPath
        foreach($file in $files) {
            Add-VaultJob -Name "Sample.CreatePDF" -Parameters @{"EntityId"=$file.Id;
↪"EntityClassId"="File"} -Description "Create PDF for: $($file._Name)" -Priority 10
        }
    }
}
```

Adds tools menu item to open powerEvents log directory with Windows Explorer:

```
Add-VaultMenuItem -Location ToolsMenu -Name "Open Log directory" -Action {
        explorer.exe "$env:LOCALAPPDATA\coolOrange\powerEvents\Logs"
}
```

5.1.3 Add-VaultRestriction

Adds a restriction to block the current operation.

Syntax

```
Add-VaultRestriction -EntityName <String> -Message<String> [<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
String	EntityName	The title or affected object of the restriction.	no
String	Message	The message of the restriction.	no

Return type

empty

Remarks

- The Cmdlet makes it possible to block the executing Vault webservice call and upcoming *Pre-* or *Post* events.
- It is possible to add multiple restrictions with the same **EntityName**.
- If a Restriction event has multiple subscribers then the event actions of all subscribers are executed even though the first subscriber already sets a restriction.
- For Vault events which are raised only for a single entity (e.g *CheckoutFile*...) , the EntityName is not displayed in the restriction Dialog by the Vault Client.

Note: The Cmdlet is **only usable** in Vault **Restriction** events (e.g *AddFile_Restrictions*, *UpdateFileStates_Restrictions*...).

Examples

Adds a restriction for files which state is trying to be changed to released

```
Register-VaultEvent -EventName UpdateFileStates_Restrictions -Action {
        param($files)
        foreach($file in $files)
        {
                if($file._NewState -eq "Released") {
                        Add-VaultRestriction -EntityName $file.Name -Message "File state␣
↪can't be set to released!"
                }
        }
}
```

5.1.4 Add-VaultTab

Adding a new Tab for a specific entity type to the Vault Explorer with an Action that is called when the Tab is clicked or the selection changes in Vault.
The cmdlet provides the possibility to define a user interface for the registered Tab.

Syntax

```
Add-VaultTab -Name <String> -EntityType <VaultTabEntityType> -Action <Scriptblock |␣
↪String> [<CommonParameters>]
```

Parameters

Type	Nam	Description	De-fault Value	Op-tional
String	Nam	Label of the Tab displayed in the Vault Explorer.		no
Vault-TabEn-tity-Type	En-ti-ty-Type	The entity type for which the Tab should be displayed.		no
Script-block / String	Ac-tion	Script block or function that gets executed for the currently selected entity in the Vault Client (`$selectedEntity`) if the Tab is activated. This can be used to define or update the Tab interface by simply returning a WPF Control from the script block or function.		no

VaultTabEntityType

Enum values:

Name	Description
File	Tab gets displayed for Files.
Item	Tab gets displayed for Items.

Return type

empty

Remarks

The Cmdlet extends the Vault Client's user interface with a new tab with the specified **-Name** as its label.
When multiple tabs with the same *-Name* for the same *-EntityType* are registered, only the latest registered tab is displayed.

The script block or the function name passed as **-Action** parameter is invoked with an powerVault entity that represents the element that is selected in the Vault Client.
If multiple entities are selected, the first selected entity will be used. Depending on the EntityType of the Tab, the passed object can be either a file or an item.
The *Action* is executed every time the Vault user changes a selection while the Tab is visible, or when the Tab is activated by the user.

The tab content is **cleared** and the log provides information when:

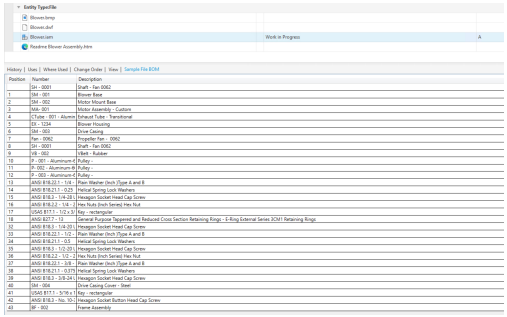
- unhandled terminating exceptions occur on the hosting applications (Vault Client) UI thread
- the *Action* returns an invalid WPF Control

Note:

- The cmdlet can **only be used** when running directly in a **Vault Explorer** process.
- Created tabs remain available as long as this process exists and disappear again when the Vault Client is restarted.
- Only the use of the cmdlet in client customization *scripts* ensures that the tab is displayed in every Vault Client session.
- The Vault Client remembers activated Tabs (based on their position) even after a restart. In this case, the passed *-Action* gets invoked immediately after *LoginVault* event registrations.

Examples

Adds a Tab for files displaying a DataGrid with their FileBom rows:



```
Add-VaultTab -EntityType File -Name 'Sample File BOM' -Action {
    param($selectedFile)

    $fileBom = Get-VaultFileBOM -File $selectedFile._FullPath

    if(-not $fileBom) {
        return BuildFromXaml @'
            <StackPanel xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation
↪">
                <Label Content="There are no rows to show in this view."␣
↪HorizontalAlignment="Center" />
            </StackPanel>
'@
    }

    #Print BOM data as Table like in Inventor
    $bomrows_table = BuildFromXaml @'
        <DataGrid xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"␣
↪AutoGenerateColumns="False" IsReadOnly="True">
            <DataGrid.Columns>
                <DataGridTextColumn Header="Position" Binding="{Binding Bom_
↪PositionNumber}" Width="60" />
                <DataGridTextColumn Header="Number" Binding="{Binding Bom_Part␣
↪Number}" Width="120"  />
                <DataGridTextColumn Header="Description" Binding="{Binding␣
↪Description}" Width="*" />
                <DataGridTextColumn Header="Quantity" Binding="{Binding Bom_Quantity}
↪" Width="60" />
                <DataGridTextColumn Header="Unit of Measure" Binding="{Binding Bom_
↪Unit}" Width="60"/>
            </DataGrid.Columns>
        </DataGrid>
'@
    $bomrows_table.ItemsSource = @($fileBom | sort-object {[int]$_.Bom_PositionNumber})
    return $bomrows_table
}

function BuildFromXaml([xml]$xaml){
    $xamlReader = New-Object System.Xml.XmlNodeReader $xaml
```

(continues on next page)

(continued from previous page)

```
    return [Windows.Markup.XamlReader]::Load($xamlReader)
}
```

Adds a Tab for items that displays their Thumbnail image:

```
Add-VaultTab -EntityType Item -Name 'Thumbnail' -Action 'DisplayItemThumbnail'

function DisplayItemThumbnail($selectedItem){
    $imageControl = New-Object System.Windows.Controls.Image
    $imageControl.Source = $selectedItem._Thumbnail.Image
    return $imageControl
}
```

5.1.5 Register-VaultEvent

Registers an action that gets invoked when the corresponding Vault API event is raised.

Syntax

```
Register-VaultEvent -EventName <String> -Action <Scriptblock | String> [-
↪SourceIdentifier <string>] [<CommonParameters>]
```

Parameters

Type	Name	Description	Default Value	Optional
VaultEvent	EventName	The name of the Vault event to hook up		no
Scriptblock / String	Action	The script block or function that becomes executed when the according Vault event is raised		no
String	SourceIdentifier	Unique string which represents the registered event, required for *unregister*	new GUID	yes

Return type

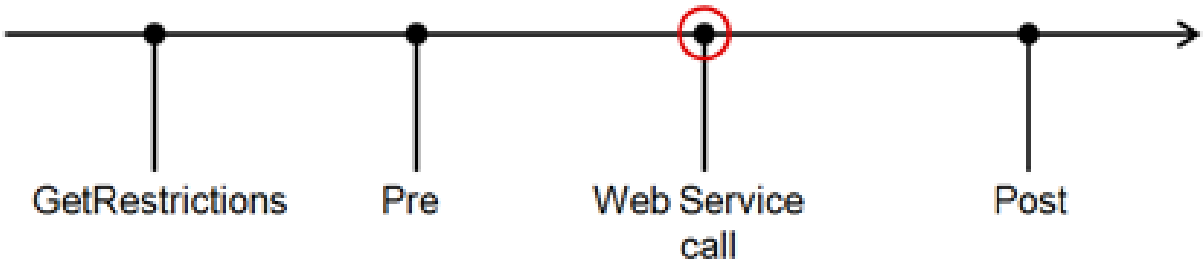
Event ← on success
empty ← on failure

Remarks

The Cmdlet registers the passed action to a specific *Vault Event*.
When a specific method of the Vault API gets called then the according Vault event is fired and powerEvents will execute the registered PowerShell action for this event.

The Vault user gets *notified about the Error* when an *exception* is thrown within a registred script block or function. This has no effect on all the other registered Vault events that will still become executed.

The information about which event gets raised for the different Vault Web Services calls can be found in the Vault SDK documentation under `Web Service Command Event Mappings`.
You can hook to **3 different stages** for each event, sorted in the sequence they are executed:



Restrictions

Gets raised before the actual *Vault Web Service call* started and before the *Pre* event is raised.
Only in here you have the possibility to *add restrictions* in order to block the upcoming events.

Pre

Gets fired after the *Restrictions* event and before the actual *Vault Web Service call*.
If *restrictions where added for this event* (could be even another Vault Extension), then the *Pre* event *will not get executed*.

Post

Gets raised after the actual *Vault Web Service call* was executed, *no matter if it was successful or not*.
Remember, when *restrictions* where blocking the execution of the *Vault Web Service call*, neither the *Post* event gets invoked.
Whether the *Vault Web Service call* to the Vault Server was successful or not can be tested by using the `$successful` parameter in your PowerShell function.

> **Warning:** **Item and Change Order events** are very special and they may behave different than you think, therefore *be sure to know how they behave: Explained here*

Examples

Registers a function that will be executed before the State of a file becomes updated:

```
function Add-Log($files) {
        Write-Host -Object "$($files[0]._Name)"
}

$event = Register-VaultEvent -EventName UpdateFileStates_Pre -Action "Add-Log"

Write-Host -Object "Registered event with name '$($event.Name)'"
```

The registered Scripblock will be executed when new files become added:

```
$event = Register-VaultEvent -EventName AddFile_Post -Action {
        param($file, $parentFolder, $successful)

        if(-not $successful) { return }
        Write-Host -Object "Added file $($file._Name) to the folder '$parentFolder._
↪FullPath'"
}
```

5.1.6 Unregister-VaultEvent

Unregisters *Vault API events* or Vault Client *Tab actions* so that the registered actions are no longer invoked when the events are raised.

Syntax

```
Unregister-VaultEvent [-EventName <String>] [-SourceIdentifier <string>] [
↪<CommonParameters>]
```

Parameters

Type	Name	Description	Optional
VaultEvent	EventName	The name of the Vault event from which to unregister	yes
String	SourceIdentifier	Unregister a specific registration by specifying it's *SourceIdentifier*	yes

Return type

Event[] ← on success
empty ← on failure

Remarks

The Cmdlet unregisters either specific or all vault events and returns the unregistered events as a result.
That means, when a specific vault event is raised, the registered *VaultEvent* will not be executed any more.

To unregister a *specific registration* the *result* from the *Register-VaultEvent* cmdlet can be used to detach the registration via its *SourceIdentifier*.
When all registrations should be detached from a *specific vault event*, only the *EventName* of the Vault event can be used.

Additionally it is possible to unregister from all *Vault API events*, Vault Client *Tab actions* and *Menu item Actions* by *not specifying any parameters*.

Examples

Unregister from a specific registration:

```
$event = Register-VaultEvent -EventName UpdateFileStates_Post -Action { param($files) }

Unregister-VaultEvent -SourceIdentifier $event.SourceIdentifier
```

Unregister from the registration after the State of a file become updated:

```
Register-VaultEvent -EventName UpdateFileStates_Post -SourceIdentifier 'AddLog_on_
↪UpdateFileStatesPost' -Action "Add-Log"

function Add-Log($file) {
        Write-Host -Object "$($file._Name)"
        Unregister-VaultEvent -SourceIdentifier 'AddLog_on_UpdateFileStatesPost'
}
```

Unregisters all the registrations from the AddFile_Post event:

```
Register-VaultEvent -EventName AddFile_Post -Action {
  param( $file )

  Write-Host -Object "Added file $($file._Name)"

}

function DoNothing($file) {}
Register-VaultEvent -EventName AddFile_Post -Action "DoNothing"

Unregister-VaultEvent -EventName AddFile_Post
```

Unregisters all event, tab action and menu item action registrations:

```
Register-VaultEvent -EventName EditItems_Restrictions -Action { ; }
Register-VaultEvent -EventName UpdateFileStates_Post -Action { ; }
Register-VaultEvent -EventName MoveFile_Pre -Action { ; }

Add-VaultTab -Name "File Tab" -EntityType 'File' - Action { ; }
Add-VaultTab -Name "Item Tab" -EntityType 'Item' - Action { ; }
Add-VaultMenuItem -Location ToolsMenu -Name 'Tools menu item' -Action { ; }
```

(continues on next page)

(continued from previous page)

```
Unregister-VaultEvent
```

5.2 Objects

5.2.1 Event

The Event object is of type *PsObject* and represents the registration to a specific *Vault API event* or Vault Client *Tab action*.

Syntax

```
$event.SourceIdentifier
```

Following properties are available :

Type	Name	Description
String	Name	The name of the registered event. For Vault API events this is the corresponding *VaultEvent* name, for tabs *'TabSelectionChanged'* is returned.
String	SourceIdentifier	Unique string which represents the registered event. Required for *Unregister-VaultEvent*.
String / ScriptBlock	Command	The PowerShell script which is executed when the event is fired.

Examples

Registered event with Command specified as function name

```
Name             : UpdateFileStates_Restrictions
SourceIdentifier : 6090dbfa-bbb4-4d31-becf-4b63c8111a4f
Command          : CanTriggerDwfJob
```

Properties of an event with Command specified as script block

```
Name             : UpdateFileStates_Post
SourceIdentifier : f4e99f68-fcfb-4d1c-b0df-4ae5611de2b6

Command          : {
                   param()
                   write-host 'This script block is executed when the Vault event is␣
↪raised!'
                   }
```

5.2.2 VaultEvent

The Enum of type *powerEvents.Cmdlets.VaultEvent* provides a list of all the supported Vault events.

Syntax

```
[powerEvents.Cmdlets.VaultEvent]::AddFile_Post
```

Remarks

Each of the following Vault events has its own set of **available arguments**:

Connection Events

LoginVault

EventNames:

- LoginVault_Post

Remarks:

- If the connection was established, the automatic variable `$vaultConnection`(provided by *powerVault*) can be used in the event action to retrieve the connection information:

```
$vaultConnection.Server # localhost
$vaultConnection.UserName # Administrator
$vaultConnection.Vault # Vault
```

Examples:
LoginVault_Post:

```
Register-VaultEvent -EventName LoginVault_Post -Action 'PostLoginVault'

function PostLoginVault() {
 #Write event code here
}
```

EventNames:

- LoginVault_Post

Remarks:

- If the connection was established, the automatic variable `$vaultConnection`(provided by *powerVault*) can be used in the event action to retrieve the connection information:

```
$vaultConnection.Server # localhost
$vaultConnection.UserName # Administrator
$vaultConnection.Vault # Vault
```

Examples:
LoginVault_Post:

```
Register-VaultEvent -EventName LoginVault_Post -Action 'PostLoginVault'

function PostLoginVault() {
 #Write event code here
}
```

Change Order Events

AddChangeOrder

EventNames:

- AddChangeOrder_Restrictions
- AddChangeOrder_Pre
- AddChangeOrder_Post

Parameters

Type	Nam	Description
Chan Orde	chan Orde	The changeOrder which should get / is added.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the all the properties available on the object like `_NewNumber`, `_NewTitle(Item,CO)`, `_NewDescription(Item,CO)`, `_NewApproveDeadline` and `NewRouting`.**Post:**In *POST* you have the full powerVaultChangeOrder object, because at this point the Change Order exists.
File[]	files	The Files to be tracked by the Change Order.
Item[	items	The Items to be tracked by the Change Order.*User Defined Link properties* are added and removed directly on the items.
File[]	at-tach-ment	Files to be attached to the Change Order.
Com-ment	com-ment	Multiple comments for the ChangeOrder including there attached files.
Emai	email	Multiple emails to send out upon completion.
bool	suc-cess-ful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

AddChangeOrder event is raised when clicking in Vault Client on *New Change Order...* -> *Save*.
Afterwards it fires **EditChangeOrder**.
No CommitChangeOrder is called.

The argument **files** contains only the files that are directly linked to the ChangeOrder, and not the primary-links of the linked Items!

The argument **comments** contains the newly added comments.
A negative Id value is returned in Pre events and even the values for all other properties except `Subject (CO)`, `Message` and `Attachments` can only be retrieved in in Post events.

Examples:
AddChangeOrder_Restrictions:

```
Register-VaultEvent -EventName AddChangeOrder_Restrictions -Action
↪'RestrictAddChangeOrder'

function RestrictAddChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪$emails) {
        #Write event code here
}
```

AddChangeOrder_Pre:

```
Register-VaultEvent -EventName AddChangeOrder_Pre -Action 'PreAddChangeOrder'

function PreAddChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪$emails) {
        #Write event code here
}
```

AddChangeOrder_Post:

```
Register-VaultEvent -EventName AddChangeOrder_Post -Action 'PostAddChangeOrder'

function PostAddChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪$emails, $successful) {
        #Write event code here
}
```

CommitChangeOrder

EventNames:

- CommitChangeOrder_Restrictions
- CommitChangeOrder_Pre
- CommitChangeOrder_Post

Parameters

Type	Nam	Description
Chan Orde	chang Orde	The changeOrder which should get / is commited.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewNumber`, `_NewTitle(Item,CO)`, `_NewDescription(Item,CO)`, `_NewApproveDeadline` and `NewRouting`.**Post:**In *POST* you have the previous properties available through `_OldNumber`, `_OldTitle(Item,CO)`, `_OldDescription(Item,CO)`, `_OldApproveDeadline` and `OldRouting`.
File[]	files	The Files to be tracked by the Change Order.
Item[	items	The Items to be tracked by the Change Order.*User Defined Link properties* are added and removed directly on the items.
File[]	at-tach-ment	Files to be attached to the Change Order.
Com-ment	com-ment	Multiple comments for the ChangeOrder including there attached files.
User[	routi gUse	All the users with the associated routing roles for this Change Order.
Emai	email	Multiple emails to send out upon completion.
bool	suc-cess-ful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

CommitChangeOrder event is raised when clicking in the Change Order dialog on *Save*. Afterwards it fires **EditChangeOrder**.

The argument **files** contains only the files that are directly linked to the ChangeOrder, and not the primary-links of the linked Items!

The argument **emails** only contains the emails of the current commit, and not the once added before!

The argument **comments** contains the current comments and the newly added comments.
A negative Id value is returned in Pre events and even the values for all other properties except `Subject (CO)`, `Message` and `Attachments` can only be retrieved in in Post events.

Examples:
CommitChangeOrder_Restrictions:

```
Register-VaultEvent -EventName CommitChangeOrder_Restrictions -Action
↪'RestrictCommitChangeOrder'

function RestrictCommitChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪ $routingUsers, $emails) {
        #Write event code here
}
```

CommitChangeOrder_Pre:

```
Register-VaultEvent -EventName CommitChangeOrder_Pre -Action 'PreCommitChangeOrder'

function PreCommitChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪$routingUsers, $emails) {
```

(continues on next page)

(continued from previous page)

```
        #Write event code here
}
```

CommitChangeOrder_Post:

```
Register-VaultEvent -EventName CommitChangeOrder_Post -Action 'PostCommitChangeOrder'

function PostCommitChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪$routingUsers, $emails, $successful) {
        #Write event code here
}
```

DeleteChangeOrders

EventNames:

- DeleteChangeOrders_Restrictions
- DeleteChangeOrders_Pre
- DeleteChangeOrders_Post

Parameters

Type	Name	Description
ChangeOrder[]	changeOrders (deletedChangeOrders in POST)	The changeOrders which should get / are deleted.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
DeleteChangeOrders_Restrictions:

```
Register-VaultEvent -EventName DeleteChangeOrders_Restrictions -Action
↪'RestrictDeleteChangeOrders'

function RestrictDeleteChangeOrders($changeOrders) {
        #Write event code here
}
```

DeleteChangeOrders_Pre:

```
Register-VaultEvent -EventName DeleteChangeOrders_Pre -Action 'PreDeleteChangeOrders'

function PreDeleteChangeOrders($changeOrders) {
        #Write event code here
}
```

DeleteChangeOrders_Post:

```
Register-VaultEvent -EventName DeleteChangeOrders_Post -Action 'PostDeleteChangeOrders'
```

(continues on next page)

(continued from previous page)

```
function PostDeleteChangeOrders($deletedChangeOrders, $successful) {
        #Write event code here
}
```

EditChangeOrder

EventNames:

- EditChangeOrder_Restrictions
- EditChangeOrder_Pre
- EditChangeOrder_Post

Parameters

Type	Name	Description
ChangeOrder	changeOrder	The Change Order which should get / is in edit mode.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

EditChangeOrder event is raised when selecting a Change Order within the Vault Client and cliching on *Edit*.

Examples:
EditChangeOrder_Restrictions:

```
Register-VaultEvent -EventName EditChangeOrder_Restrictions -Action
↪'RestrictEditChangeOrder'

function RestrictEditChangeOrder($changeOrder) {
        #Write event code here
}
```

EditChangeOrder_Pre:

```
Register-VaultEvent -EventName EditChangeOrder_Pre -Action 'PreEditChangeOrder'

function PreEditChangeOrder($changeOrder) {
        #Write event code here
}
```

EditChangeOrder_Post:

```
Register-VaultEvent -EventName EditChangeOrder_Post -Action 'PostEditChangeOrder'

function PostEditChangeOrder($changeOrder, $successful) {
        #Write event code here
}
```

UpdateChangeOrderState

EventNames:

- UpdateChangeOrderState_Restrictions
- UpdateChangeOrderState_Pre
- UpdateChangeOrderState_Post

Parameters

Type	Name	Description
Chang Order	chang Order	The updated / updating Change Order for a LifeCycle.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewStateEntered`.**Post:**In *POST* you can retrieve the previous information by using the properties `_OldState` and `_OldStateEntered`.
string	ac- tiv- ity	The activity being completed.
Com- ment[	com- ments	Multiple comments for the ChangeOrder including there attached files.
Email	email	Multiple emails to send out upon completion.
bool	suc- cess- ful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

UpdateChangeOrderState event is raised when clicking on *Submit/Approve*, in general when the State/Activity is changed.

When fired by **Vault Client** then the property `_NewStateEntered` from the argument **changeOrder** is the StateEntered of the current ChangeOrder!

Each comment in the argument **comments** provide negative Ids and only `Subject (CO)`, `Message` and `Attachments` values can be retrieved. All other data is unavailable, even in Post events.

Examples:
UpdateChangeOrderState_Restrictions:

```
Register-VaultEvent -EventName UpdateChangeOrderState_Restrictions -Action
↪'RestrictUpdateChangeOrderState'

function RestrictUpdateChangeOrderState($changeOrder, $activity, $comments, $emails) {
        #Write event code here
}
```

UpdateChangeOrderState_Pre:

```
Register-VaultEvent -EventName UpdateChangeOrderState_Pre -Action
↪'PreUpdateChangeOrderState'

function PreUpdateChangeOrderState($changeOrder, $activity, $comments, $emails) {
```

(continues on next page)

(continued from previous page)

```
        #Write event code here
}
```

UpdateChangeOrderState_Post:

```
Register-VaultEvent -EventName UpdateChangeOrderState_Post -Action
↪'PostUpdateChangeOrderState'

function PostUpdateChangeOrderState($changeOrder, $activity, $comments, $emails,
↪$successful) {
        #Write event code here
}
```

AddChangeOrder

EventNames:

- AddChangeOrder_Restrictions
- AddChangeOrder_Pre
- AddChangeOrder_Post

Parameters

Type	Nam	Description
Chan Orde	chan Orde	The changeOrder which should get / is added.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the all the properties available on the object like `_NewNumber`, `_NewTitle(Item,CO)`, `_NewDescription(Item,CO)`, `_NewApproveDeadline` and `NewRouting`.**Post:**In *POST* you have the full powerVaultChangeOrder object, because at this point the Change Order exists.
File[]	files	The Files to be tracked by the Change Order.
Item[	items	The Items to be tracked by the Change Order.*User Defined Link properties* are added and removed directly on the items.
File[]	at-tach-ment	Files to be attached to the Change Order.
Com-ment	com-ment	Multiple comments for the ChangeOrder including there attached files.
Emai	email	Multiple emails to send out upon completion.
bool	suc-cess-ful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

AddChangeOrder event is raised when clicking in Vault Client on *New Change Order...* -> *Save*.
Afterwards it fires **EditChangeOrder**.
No CommitChangeOrder is called.

The argument **files** contains only the files that are directly linked to the ChangeOrder, and not the primary-links of the linked Items!

The argument **comments** contains the newly added comments.
A negative Id value is returned in Pre events and even the values for all other properties except `Subject (CO)`, `Message` and `Attachments` can only be retrieved in in Post events.

Examples:
AddChangeOrder_Restrictions:

```
Register-VaultEvent -EventName AddChangeOrder_Restrictions -Action
↪'RestrictAddChangeOrder'

function RestrictAddChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪$emails) {
        #Write event code here
}
```

AddChangeOrder_Pre:

```
Register-VaultEvent -EventName AddChangeOrder_Pre -Action 'PreAddChangeOrder'

function PreAddChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪$emails) {
        #Write event code here
}
```

AddChangeOrder_Post:

```
Register-VaultEvent -EventName AddChangeOrder_Post -Action 'PostAddChangeOrder'

function PostAddChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪$emails, $successful) {
        #Write event code here
}
```

EditChangeOrder

EventNames:

- EditChangeOrder_Restrictions
- EditChangeOrder_Pre
- EditChangeOrder_Post

Parameters

Type	Name	Description
Change-Order	change-Order	The Change Order which should get / is in edit mode.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

EditChangeOrder event is raised when selecting a Change Order within the Vault Client and cliching on *Edit*.

Examples:
EditChangeOrder_Restrictions:

```
Register-VaultEvent -EventName EditChangeOrder_Restrictions -Action
↪'RestrictEditChangeOrder'

function RestrictEditChangeOrder($changeOrder) {
        #Write event code here
}
```

EditChangeOrder_Pre:

```
Register-VaultEvent -EventName EditChangeOrder_Pre -Action 'PreEditChangeOrder'

function PreEditChangeOrder($changeOrder) {
        #Write event code here
}
```

EditChangeOrder_Post:

```
Register-VaultEvent -EventName EditChangeOrder_Post -Action 'PostEditChangeOrder'

function PostEditChangeOrder($changeOrder, $successful) {
        #Write event code here
}
```

CommitChangeOrder

EventNames:

- CommitChangeOrder_Restrictions
- CommitChangeOrder_Pre
- CommitChangeOrder_Post

Parameters

Type	Nam	Description
Chan Orde	chang Orde	The changeOrder which should get / is commited.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewNumber`, `_NewTitle(Item,CO)`, `_NewDescription(Item,CO)`, `_NewApproveDeadline` and `NewRouting`.**Post:**In *POST* you have the previous properties available through `_OldNumber`, `_OldTitle(Item,CO)`, `_OldDescription(Item,CO)`, `_OldApproveDeadline` and `OldRouting`.
File[]	files	The Files to be tracked by the Change Order.
Item[	items	The Items to be tracked by the Change Order.*User Defined Link properties* are added and removed directly on the items.
File[]	at-tach-ments	Files to be attached to the Change Order.
Com-ment	com-ments	Multiple comments for the ChangeOrder including there attached files.
User[	routi gUse	All the users with the associated routing roles for this Change Order.
Emai	email	Multiple emails to send out upon completion.
bool	suc-cess-ful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

CommitChangeOrder event is raised when clicking in the Change Order dialog on *Save*. Afterwards it fires **EditChangeOrder**.

The argument **files** contains only the files that are directly linked to the ChangeOrder, and not the primary-links of the linked Items!

The argument **emails** only contains the emails of the current commit, and not the once added before!

The argument **comments** contains the current comments and the newly added comments.
A negative Id value is returned in Pre events and even the values for all other properties except `Subject (CO)`, `Message` and `Attachments` can only be retrieved in in Post events.

Examples:
CommitChangeOrder_Restrictions:

```
Register-VaultEvent -EventName CommitChangeOrder_Restrictions -Action
↪'RestrictCommitChangeOrder'

function RestrictCommitChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪ $routingUsers, $emails) {
        #Write event code here
}
```

CommitChangeOrder_Pre:

```
Register-VaultEvent -EventName CommitChangeOrder_Pre -Action 'PreCommitChangeOrder'

function PreCommitChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪$routingUsers, $emails) {
```

(continues on next page)

(continued from previous page)

```
        #Write event code here
}
```

CommitChangeOrder_Post:

```
Register-VaultEvent -EventName CommitChangeOrder_Post -Action 'PostCommitChangeOrder'

function PostCommitChangeOrder($changeOrder, $files, $items, $attachments, $comments,
↪$routingUsers, $emails, $successful) {
        #Write event code here
}
```

DeleteChangeOrders

EventNames:

- DeleteChangeOrders_Restrictions
- DeleteChangeOrders_Pre
- DeleteChangeOrders_Post

Parameters

Type	Name	Description
ChangeOrder[]	changeOrders (deletedChangeOrders in POST)	The changeOrders which should get / are deleted.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
DeleteChangeOrders_Restrictions:

```
Register-VaultEvent -EventName DeleteChangeOrders_Restrictions -Action
↪'RestrictDeleteChangeOrders'

function RestrictDeleteChangeOrders($changeOrders) {
        #Write event code here
}
```

DeleteChangeOrders_Pre:

```
Register-VaultEvent -EventName DeleteChangeOrders_Pre -Action 'PreDeleteChangeOrders'

function PreDeleteChangeOrders($changeOrders) {
        #Write event code here
}
```

DeleteChangeOrders_Post:

```
Register-VaultEvent -EventName DeleteChangeOrders_Post -Action 'PostDeleteChangeOrders'
```

(continues on next page)

(continued from previous page)

```
function PostDeleteChangeOrders($deletedChangeOrders, $successful) {
        #Write event code here
}
```

UpdateChangeOrderState

EventNames:

- UpdateChangeOrderState_Restrictions
- UpdateChangeOrderState_Pre
- UpdateChangeOrderState_Post

Parameters

Type	Name	Description
Chang Order	chang Order	The updated / updating Change Order for a LifeCycle.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewStateEntered`.**Post:**In *POST* you can retrieve the previous information by using the properties `_OldState` and `_OldStateEntered`.
string	activity	The activity being completed.
Comment[	comments	Multiple comments for the ChangeOrder including there attached files.
Email	emails	Multiple emails to send out upon completion.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

UpdateChangeOrderState event is raised when clicking on *Submit/Approve*, in general when the State/Activity is changed.

When fired by **Vault Client** then the property `_NewStateEntered` from the argument **changeOrder** is the StateEntered of the current ChangeOrder!

Each comment in the argument **comments** provide negative Ids and only `Subject (CO)`, `Message` and `Attachments` values can be retrieved. All other data is unavailable, even in Post events.

Examples:
UpdateChangeOrderState_Restrictions:

```
Register-VaultEvent -EventName UpdateChangeOrderState_Restrictions -Action
↪'RestrictUpdateChangeOrderState'

function RestrictUpdateChangeOrderState($changeOrder, $activity, $comments, $emails) {
        #Write event code here
}
```

UpdateChangeOrderState_Pre:

```
Register-VaultEvent -EventName UpdateChangeOrderState_Pre -Action
↪'PreUpdateChangeOrderState'

function PreUpdateChangeOrderState($changeOrder, $activity, $comments, $emails) {
        #Write event code here
}
```

UpdateChangeOrderState_Post:

```
Register-VaultEvent -EventName UpdateChangeOrderState_Post -Action
↪'PostUpdateChangeOrderState'

function PostUpdateChangeOrderState($changeOrder, $activity, $comments, $emails,
↪$successful) {
        #Write event code here
}
```

Custom Entity Events

UpdateCustomEntityStates

EventNames:

- UpdateCustomEntityStates_Restrictions
- UpdateCustomEntityStates_Pre
- UpdateCustomEntityStates_Post

Parameters

Type	Name	Description
CustomObject[]	customObjects	The updated / updating customObjects for a LifeCycle.**Restrictions and Pre:** In *Restrictions* and *Pre* you could get the future changes by the special properties `_NewState`, `_NewLifeCycleDefinition` and `_NewComment`.**Post:**In *Post* you could get the old information by the special properties `_OldState`, `_OldLifeCycleDefinition` and `_OldComment`
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

For the **UpdateCustomEntityStates_Post event** the property `_OldComment` will always be empty because the VaultAPI currently does not support Comments on CustomObjects (it is reserved for future use).

Examples:
UpdateCustomEntityStates_Restrictions:

```
Register-VaultEvent -EventName UpdateCustomEntityStates_Restrictions -Action
↪'RestrictUpdateCustomEntityStates'
```

(continues on next page)

(continued from previous page)

```
function RestrictUpdateCustomEntityStates($customObjects) {
        #Write event code here
}
```

UpdateCustomEntityStates_Pre:

```
Register-VaultEvent -EventName UpdateCustomEntityStates_Pre -Action
↪'PreUpdateCustomEntityStates'

function PreUpdateCustomEntityStates($customObjects) {
        #Write event code here
}
```

UpdateCustomEntityStates_Post:

```
Register-VaultEvent -EventName UpdateCustomEntityStates_Post -Action
↪'PostUpdateCustomEntityStates'

function PostUpdateCustomEntityStates($customObjects, $successful) {
#Write event code here
}
```

UpdateCustomEntityStates

EventNames:

- UpdateCustomEntityStates_Restrictions
- UpdateCustomEntityStates_Pre
- UpdateCustomEntityStates_Post

Parameters

Type	Nam	Description
Cus-to-mOb-ject[]	cus-to-mOb-jects	The updated / updating customObjects for a LifeCycle.**Restrictions and Pre:** In *Restrictions* and *Pre* you could get the future changes by the special properties `_NewState`, `_NewLifeCycleDefinition` and `_NewComment`.**Post:**In *Post* you could get the old information by the special properties `_OldState`, `_OldLifeCycleDefinition` and `_OldComment`
bool	suc-cess-ful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

For the **UpdateCustomEntityStates_Post event** the property `_OldComment` will always be empty because the VaultAPI currently does not support Comments on CustomObjects (it is reserved for future use).

Examples:
UpdateCustomEntityStates_Restrictions:

```
Register-VaultEvent -EventName UpdateCustomEntityStates_Restrictions -Action
↪'RestrictUpdateCustomEntityStates'

function RestrictUpdateCustomEntityStates($customObjects) {
        #Write event code here
}
```

UpdateCustomEntityStates_Pre:

```
Register-VaultEvent -EventName UpdateCustomEntityStates_Pre -Action
↪'PreUpdateCustomEntityStates'

function PreUpdateCustomEntityStates($customObjects) {
        #Write event code here
}
```

UpdateCustomEntityStates_Post:

```
Register-VaultEvent -EventName UpdateCustomEntityStates_Post -Action
↪'PostUpdateCustomEntityStates'

function PostUpdateCustomEntityStates($customObjects, $successful) {
#Write event code here
}
```

File Events

AddFile

EventNames:

- AddFile_Restrictions
- AddFile_Pre
- AddFile_Post

Parameters

Type	Nam	Description
File	file	The file which is / was added.**Restrictions and Pre:** In *Restrictions* and *Pre* you have only very limited data set on this object like `_NewName`, `_NewComment`, `_NewModDate`, `_NewClassification` and `_NewHidden`, because the file does not exist yet.**Post:**In *Post* it is a powerVaultFile object, because here the file exists.
Folder	parentFolde	The folder where the new file is located.
File[]	dependencies	The file dependencies what this new file has.
File[]	attachments	The attached files what this new file has.
FileBomRow[]	fileBom	The Bill of Materials of the newly checked-in file.For Inventor files with multiple Model States only the BOM data of the *Master* model state is available.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
AddFile_Restrictions:

```
Register-VaultEvent -EventName AddFile_Restrictions -Action 'RestrictAddFile'

function RestrictAddFile($file, $parentFolder, $dependencies, $attachments, $fileBom) {
        #Write event code here
}
```

AddFile_Pre:

```
Register-VaultEvent -EventName AddFile_Pre -Action 'PreAddFile'

function PreAddFile($file, $parentFolder, $dependencies, $attachments, $fileBom) {
        #Write event code here
}
```

AddFile_Post:

```
Register-VaultEvent -EventName AddFile_Post -Action 'PostAddFile'

function PostAddFile($file, $parentFolder, $dependencies, $attachments, $fileBom,
↪$successful) {
        #Write event code here
}
```

CheckinFile

EventNames:

- CheckinFile_Restrictions
- CheckinFile_Pre
- CheckinFile_Post

Parameters

Type	Nam	Description
File	file	The file which is / was checked in.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewName`, `_NewComment`, `_NewModDate`, `_NewClassification`, `_NewCheckedOut` and `_NewHidden`.**Post:**In *POST* you can retrieve the information from previous events by using the properties `_OldName`, `_OldComment`, `_OldModDate`, `_OldClassification`, `_OldCheckedOut` and `_OldHidden`.
File[]	dependencies	The file dependencies what this new checked-in file has.
File[]	attachments	The attachments what this new checked-in file has.
FileBomRow[	fileBom	The Bill of Materials of the checked-in file.For Inventor files with multiple Model States only the BOM data of the *Master* model state is available.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

CheckinFile event is not raised when a file is added to Vault for the first time. Instead the *AddFile* event is raised.

Examples:
CheckinFile_Restrictions:

```
Register-VaultEvent -EventName CheckinFile_Restrictions -Action 'RestrictCheckinFile'
function RestrictCheckinFile($file, $dependencies, $attachments, $fileBom) {
        #Write event code here
}
```

CheckinFile_Pre:

```
Register-VaultEvent -EventName CheckinFile_Pre -Action 'PreCheckinFile'
function PreCheckinFile($file, $dependencies, $attachments, $fileBom) {
        #Write event code here
}
```

CheckinFile_Post:

```
Register-VaultEvent -EventName CheckinFile_Post -Action 'PostCheckinFile'
function PostCheckinFile($file, $dependencies, $attachments, $fileBom, $successful) {
        #Write event code here
}
```

CheckoutFile

EventNames:

- CheckoutFile_Restrictions
- CheckoutFile_Pre
- CheckoutFile_Post

Parameters

Type	Nam	Description
File	file	The file which should get / is checked=out.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewComment` and `_NewCheckoutMachine`.**Post:**In *POST* you can retrieve the information from the previous events by using the properties `_OldComment` and `_OldCheckoutMachine`.
string	local-Path	The local path where the file should / is checked=out
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
CheckoutFile_Restrictions:

```
Register-VaultEvent -EventName CheckoutFile_Restrictions -Action 'RestrictCheckoutFile'
function RestrictCheckoutFile($file, $localPath) {
        #Write event code here
}
```

CheckoutFile_Pre:

```
Register-VaultEvent -EventName CheckoutFile_Pre -Action 'PreCheckoutFile'
function PreCheckoutFile($file, $localPath) {
        #Write event code here
}
```

CheckoutFile_Post:

```
Register-VaultEvent -EventName CheckoutFile_Post -Action 'PostCheckoutFile'
function PostCheckoutFile($file, $localPath, $successful) {
        #Write event code here
}
```

DeleteFiles

EventNames:

- DeleteFiles_Restrictions
- DeleteFiles_Pre
- DeleteFiles_Post

Parameters

Type	Name	Description
File[]	files (deletedFiles in POST)	The files which should be deleted or are already deleted.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
DeleteFiles_Restrictions:

```
Register-VaultEvent -EventName DeleteFiles_Restrictions -Action 'RestrictDeleteFiles'

function RestrictDeleteFiles($files) {
        #Write event code here
}
```

DeleteFiles_Pre:

```
Register-VaultEvent -EventName DeleteFiles_Pre -Action 'PreDeleteFiles'

function PreDeleteFiles($files) {
        #Write event code here
}
```

DeleteFiles_Post:

```
Register-VaultEvent -EventName DeleteFiles_Post -Action 'PostDeleteFiles'

function PostDeleteFiles($deletedFiles, $successful) {
        #Write event code here
}
```

DownloadFiles

EventNames:

- DownloadFiles_Restrictions
- DownloadFiles_Pre
- DownloadFiles_Post

Parameters

Type	Name	Description
File[]	files	The files which should get / are downloaded.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
DownloadFiles_Restrictions:

```
Register-VaultEvent -EventName DownloadFiles_Restrictions -Action 'RestrictDownloadFiles'

function RestrictDownloadFiles($files) {
        #Write event code here
}
```

DownloadFiles_Pre:

```
Register-VaultEvent -EventName DownloadFiles_Pre -Action 'PreDownloadFiles'

function PreDownloadFiles($files) {
        #Write event code here
}
```

DownloadFiles_Post:

```
Register-VaultEvent -EventName DownloadFiles_Post -Action 'PostDownloadFiles'

function PostDownloadFiles($files, $successful) {
        #Write event code here
}
```

MoveFile

EventNames:

- MoveFile_Restrictions
- MoveFile_Pre
- MoveFile_Post

Parameters

Type	Name	Description
File	file	The file which should get / are moved.**Restrictions and Pre:** In *Restrictions* and *Pre* you could get the new path by the property `_NewFullPath`.**Post:**In *POST* you you could get the old path by the property `_OldFullPath`.
Folde	parentFolder	The parent folder where the file should get / is moved.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
MoveFile_Restrictions:

```
Register-VaultEvent -EventName MoveFile_Restrictions -Action 'RestrictMoveFile'

function RestrictMoveFile($file, $parentFolder) {
        #Write event code here
}
```

MoveFile_Pre:

```
Register-VaultEvent -EventName MoveFile_Pre -Action 'PreMoveFile'

function PreMoveFile($file, $parentFolder) {
        #Write event code here
}
```

MoveFile_Post:

```
Register-VaultEvent -EventName MoveFile_Post -Action 'PostMoveFile'

function PostMoveFile($file, $parentFolder, $successful) {
        #Write event code here
}
```

UpdateFileStates

EventNames:

- UpdateFileStates_Restrictions
- UpdateFileStates_Pre
- UpdateFileStates_Post

Parameters

Type	Nam	Description
File[]	files	The updated / updating files for a LifeCycle.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewComment`, `_NewLifeCycleDefinition` and `_NewState`.**Post:**In *POST* you can retrieve the information from previous events by using the properties `_OldComment`, `_OldState` and `_OldLifeCycleDefinition`.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
UpdateFileStates_Restrictions:

```
Register-VaultEvent -EventName UpdateFileStates_Restrictions -Action
↪'RestrictUpdateFileStates'

function RestrictUpdateFileStates($files) {
```

(continues on next page)

(continued from previous page)

```
        #Write event code here
}
```

UpdateFileStates_Pre:

```
Register-VaultEvent -EventName UpdateFileStates_Pre -Action 'PreUpdateFileStates'

function PreUpdateFileStates($files) {
        #Write event code here
}
```

UpdateFileStates_Post:

```
Register-VaultEvent -EventName UpdateFileStates_Post -Action 'PostUpdateFileStates'

function PostUpdateFileStates($files, $successful) {
        #Write event code here
}
```

AddFile

EventNames:

- AddFile_Restrictions
- AddFile_Pre
- AddFile_Post

Parameters

Type	Nam	Description
File	file	The file which is / was added.**Restrictions and Pre:** In *Restrictions* and *Pre* you have only very limited data set on this object like `_NewName`, `_NewComment`, `_NewModDate`, `_NewClassification` and `_NewHidden`, because the file does not exist yet.**Post:**In *Post* it is a powerVaultFile object, because here the file exists.
Folde	parentFolde	The folder where the new file is located.
File[]	dependencies	The file dependencies what this new file has.
File[]	attachments	The attached files what this new file has.
FileBomRow[]	fileBom	The Bill of Materials of the newly checked-in file.For Inventor files with multiple Model States only the BOM data of the *Master* model state is available.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
AddFile_Restrictions:

```
Register-VaultEvent -EventName AddFile_Restrictions -Action 'RestrictAddFile'

function RestrictAddFile($file, $parentFolder, $dependencies, $attachments, $fileBom) {
        #Write event code here
}
```

AddFile_Pre:

```
Register-VaultEvent -EventName AddFile_Pre -Action 'PreAddFile'

function PreAddFile($file, $parentFolder, $dependencies, $attachments, $fileBom) {
        #Write event code here
}
```

AddFile_Post:

```
Register-VaultEvent -EventName AddFile_Post -Action 'PostAddFile'

function PostAddFile($file, $parentFolder, $dependencies, $attachments, $fileBom,
↪$successful) {
        #Write event code here
}
```

CheckinFile

EventNames:

- CheckinFile_Restrictions
- CheckinFile_Pre
- CheckinFile_Post

Parameters

Type	Nam	Description
File	file	The file which is / was checked in.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewName`, `_NewComment`, `_NewModDate`, `_NewClassification`, `_NewCheckedOut` and `_NewHidden`.**Post:**In *POST* you can retrieve the information from previous events by using the properties `_OldName`, `_OldComment`, `_OldModDate`, `_OldClassification`, `_OldCheckedOut` and `_OldHidden`.
File[]	dependencies	The file dependencies what this new checked-in file has.
File[]	attachments	The attachments what this new checked-in file has.
FileBomRow[	fileBom	The Bill of Materials of the checked-in file.For Inventor files with multiple Model States only the BOM data of the *Master* model state is available.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

CheckinFile event is not raised when a file is added to Vault for the first time. Instead the *AddFile* event is raised.

Examples:
CheckinFile_Restrictions:

```
Register-VaultEvent -EventName CheckinFile_Restrictions -Action 'RestrictCheckinFile'
function RestrictCheckinFile($file, $dependencies, $attachments, $fileBom) {
        #Write event code here
}
```

CheckinFile_Pre:

```
Register-VaultEvent -EventName CheckinFile_Pre -Action 'PreCheckinFile'
function PreCheckinFile($file, $dependencies, $attachments, $fileBom) {
        #Write event code here
}
```

CheckinFile_Post:

```
Register-VaultEvent -EventName CheckinFile_Post -Action 'PostCheckinFile'
function PostCheckinFile($file, $dependencies, $attachments, $fileBom, $successful) {
        #Write event code here
}
```

CheckoutFile

EventNames:

- CheckoutFile_Restrictions
- CheckoutFile_Pre
- CheckoutFile_Post

Parameters

Type	Nam	Description
File	file	The file which should get / is checked=out.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewComment` and `_NewCheckoutMachine`.**Post:**In *POST* you can retrieve the information from the previous events by using the properties `_OldComment` and `_OldCheckoutMachine`.
string	local-Path	The local path where the file should / is checked=out
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
CheckoutFile_Restrictions:

```
Register-VaultEvent -EventName CheckoutFile_Restrictions -Action 'RestrictCheckoutFile'
function RestrictCheckoutFile($file, $localPath) {
        #Write event code here
}
```

CheckoutFile_Pre:

```
Register-VaultEvent -EventName CheckoutFile_Pre -Action 'PreCheckoutFile'
function PreCheckoutFile($file, $localPath) {
        #Write event code here
}
```

CheckoutFile_Post:

```
Register-VaultEvent -EventName CheckoutFile_Post -Action 'PostCheckoutFile'
function PostCheckoutFile($file, $localPath, $successful) {
        #Write event code here
}
```

DeleteFiles

EventNames:

- DeleteFiles_Restrictions
- DeleteFiles_Pre
- DeleteFiles_Post

Parameters

Type	Name	Description
File[]	files (deletedFiles in POST)	The files which should be deleted or are already deleted.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
DeleteFiles_Restrictions:

```
Register-VaultEvent -EventName DeleteFiles_Restrictions -Action 'RestrictDeleteFiles'

function RestrictDeleteFiles($files) {
        #Write event code here
}
```

DeleteFiles_Pre:

```
Register-VaultEvent -EventName DeleteFiles_Pre -Action 'PreDeleteFiles'

function PreDeleteFiles($files) {
        #Write event code here
}
```

DeleteFiles_Post:

```
Register-VaultEvent -EventName DeleteFiles_Post -Action 'PostDeleteFiles'

function PostDeleteFiles($deletedFiles, $successful) {
        #Write event code here
}
```

DownloadFiles

EventNames:

- DownloadFiles_Restrictions
- DownloadFiles_Pre
- DownloadFiles_Post

Parameters

Type	Name	Description
File[]	files	The files which should get / are downloaded.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
DownloadFiles_Restrictions:

```
Register-VaultEvent -EventName DownloadFiles_Restrictions -Action 'RestrictDownloadFiles'

function RestrictDownloadFiles($files) {
        #Write event code here
}
```

DownloadFiles_Pre:

```
Register-VaultEvent -EventName DownloadFiles_Pre -Action 'PreDownloadFiles'

function PreDownloadFiles($files) {
        #Write event code here
}
```

DownloadFiles_Post:

```
Register-VaultEvent -EventName DownloadFiles_Post -Action 'PostDownloadFiles'

function PostDownloadFiles($files, $successful) {
        #Write event code here
}
```

MoveFile

EventNames:

- MoveFile_Restrictions
- MoveFile_Pre
- MoveFile_Post

Parameters

Type	Name	Description
File	file	The file which should get / are moved.**Restrictions and Pre:** In *Restrictions* and *Pre* you could get the new path by the property `_NewFullPath`.**Post:**In *POST* you you could get the old path by the property `_OldFullPath`.
Folde	parentFolder	The parent folder where the file should get / is moved.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
MoveFile_Restrictions:

```
Register-VaultEvent -EventName MoveFile_Restrictions -Action 'RestrictMoveFile'

function RestrictMoveFile($file, $parentFolder) {
        #Write event code here
}
```

MoveFile_Pre:

```
Register-VaultEvent -EventName MoveFile_Pre -Action 'PreMoveFile'

function PreMoveFile($file, $parentFolder) {
        #Write event code here
}
```

MoveFile_Post:

```
Register-VaultEvent -EventName MoveFile_Post -Action 'PostMoveFile'

function PostMoveFile($file, $parentFolder, $successful) {
        #Write event code here
}
```

UpdateFileStates

EventNames:

- UpdateFileStates_Restrictions
- UpdateFileStates_Pre
- UpdateFileStates_Post

Parameters

Type	Nam	Description
File[]	files	The updated / updating files for a LifeCycle.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewComment`, `_NewLifeCycleDefinition` and `_NewState`.**Post:**In *POST* you can retrieve the information from previous events by using the properties `_OldComment`, `_OldState` and `_OldLifeCycleDefinition`.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
UpdateFileStates_Restrictions:

```
Register-VaultEvent -EventName UpdateFileStates_Restrictions -Action
↪'RestrictUpdateFileStates'

function RestrictUpdateFileStates($files) {
```

(continues on next page)

(continued from previous page)

```
        #Write event code here
}
```

UpdateFileStates_Pre:

```
Register-VaultEvent -EventName UpdateFileStates_Pre -Action 'PreUpdateFileStates'

function PreUpdateFileStates($files) {
        #Write event code here
}
```

UpdateFileStates_Post:

```
Register-VaultEvent -EventName UpdateFileStates_Post -Action 'PostUpdateFileStates'

function PostUpdateFileStates($files, $successful) {
        #Write event code here
}
```

Folder Events

AddFolder

EventNames:

- AddFolder_Restrictions
- AddFolder_Pre
- AddFolder_Post

Parameters

Type	Name	Description
Folde	folder	The folder which should get / is added.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the very limited data set on this object like `_NewName` and `_NewLibrary`.**Post:**In *POST* its an usual powerVaultFolder object, because at this point the folder exists.
Folde	parentFolder	The parent folder where the new folder will be / is added.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
AddFolder_Restrictions:

```
Register-VaultEvent -EventName AddFolder_Restrictions -Action 'RestrictAddFolder'

function RestrictAddFolder($folder, $parentFolder) {
        #Write event code here
}
```

AddFolder_Pre:

```
Register-VaultEvent -EventName AddFolder_Pre -Action 'PreAddFolder'

function PreAddFolder($folder, $parentFolder) {
        #Write event code here
}
```

AddFolder_Post:

```
Register-VaultEvent -EventName AddFolder_Post -Action 'PostAddFolder'

function PostAddFolder($folder, $parentFolder, $successful) {
        #Write event code here
}
```

DeleteFolder

EventNames:

- DeleteFolder_Restrictions
- DeleteFolder_Pre
- DeleteFolder_Post

Parameters

Type	Name	Description
Folder	folder (deletedFolder in POST)	The folder which should get / is deleted.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
DeleteFolder_Restrictions:

```
Register-VaultEvent -EventName DeleteFolder_Restrictions -Action 'RestrictDeleteFolder'

function RestrictDeleteFolder($folder) {
        #Write event code here
}
```

DeleteFolder_Pre:

```
Register-VaultEvent -EventName DeleteFolder_Pre -Action 'PreDeleteFolder'

function PreDeleteFolder($folder) {
        #Write event code here
}
```

DeleteFolder_Post:

```
Register-VaultEvent -EventName DeleteFolder_Post -Action 'PostDeleteFolder'

function PostDeleteFolder($deletedFolder, $successful) {
        #Write event code here
}
```

MoveFolder

EventNames:

- MoveFolder_Restrictions
- MoveFolder_Pre
- MoveFolder_Post

Parameters

Type	Name	Description
Folde	folder	The folder which should get / is moved.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewFullPath`.**Post:**In *POST* you can retrieve the previous information by using the properties `_OldFullPath`.
Folde	parentFolder	The new parent folder where the folder should / is moved
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
MoveFolder_Restrictions:

```
Register-VaultEvent -EventName MoveFolder_Restrictions -Action 'RestrictMoveFolder'

function RestrictMoveFolder($folder, $parentFolder) {
        #Write event code here
}
```

MoveFolder_Pre:

```
Register-VaultEvent -EventName MoveFolder_Pre -Action 'PreMoveFolder'

function PreMoveFolder($folder, $parentFolder) {
        #Write event code here
}
```

MoveFolder_Post:

```
Register-VaultEvent -EventName MoveFolder_Post -Action 'PostMoveFolder'

function PostMoveFolder($folder, $parentFolder, $successful) {
        #Write event code here
}
```

AddFolder

EventNames:

- AddFolder_Restrictions
- AddFolder_Pre
- AddFolder_Post

Parameters

Type	Name	Description
Folde	folder	The folder which should get / is added.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the very limited data set on this object like `_NewName` and `_NewLibrary`.**Post:**In *POST* its an usual powerVaultFolder object, because at this point the folder exists.
Folde	parentFolder	The parent folder where the new folder will be / is added.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
AddFolder_Restrictions:

```
Register-VaultEvent -EventName AddFolder_Restrictions -Action 'RestrictAddFolder'

function RestrictAddFolder($folder, $parentFolder) {
        #Write event code here
}
```

AddFolder_Pre:

```
Register-VaultEvent -EventName AddFolder_Pre -Action 'PreAddFolder'

function PreAddFolder($folder, $parentFolder) {
        #Write event code here
}
```

AddFolder_Post:

```
Register-VaultEvent -EventName AddFolder_Post -Action 'PostAddFolder'

function PostAddFolder($folder, $parentFolder, $successful) {
        #Write event code here
}
```

MoveFolder

EventNames:

- MoveFolder_Restrictions
- MoveFolder_Pre
- MoveFolder_Post

Parameters

Type	Name	Description
Folde	folder	The folder which should get / is moved.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewFullPath`.**Post:**In *POST* you can retrieve the previous information by using the properties `_OldFullPath`.
Folde	par-ent-Folder	The new parent folder where the folder should / is moved
bool	suc-cess-ful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
MoveFolder_Restrictions:

```
Register-VaultEvent -EventName MoveFolder_Restrictions -Action 'RestrictMoveFolder'

function RestrictMoveFolder($folder, $parentFolder) {
        #Write event code here
}
```

MoveFolder_Pre:

```
Register-VaultEvent -EventName MoveFolder_Pre -Action 'PreMoveFolder'

function PreMoveFolder($folder, $parentFolder) {
        #Write event code here
}
```

MoveFolder_Post:

```
Register-VaultEvent -EventName MoveFolder_Post -Action 'PostMoveFolder'

function PostMoveFolder($folder, $parentFolder, $successful) {
        #Write event code here
}
```

DeleteFolder

EventNames:

- DeleteFolder_Restrictions
- DeleteFolder_Pre
- DeleteFolder_Post

Parameters

Type	Name	Description
Folder	folder (deletedFolder in POST)	The folder which should get / is deleted.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
DeleteFolder_Restrictions:

```
Register-VaultEvent -EventName DeleteFolder_Restrictions -Action 'RestrictDeleteFolder'

function RestrictDeleteFolder($folder) {
        #Write event code here
}
```

DeleteFolder_Pre:

```
Register-VaultEvent -EventName DeleteFolder_Pre -Action 'PreDeleteFolder'

function PreDeleteFolder($folder) {
        #Write event code here
}
```

DeleteFolder_Post:

```
Register-VaultEvent -EventName DeleteFolder_Post -Action 'PostDeleteFolder'

function PostDeleteFolder($deletedFolder, $successful) {
        #Write event code here
}
```

Item Events

AddItem

EventNames:

- AddItem_Restrictions
- AddItem_Pre
- AddItem_Post

Parameters

Type	Nam	Description
Item	item	The item which should get / is added.**Restrictions and Pre:** In *Restrictions* and *Pre* you have very limited data, because the item does not exist so far.Only the property `_NewCategoryName` is set as long as a valid category Id was passed.**Post:**In *POST* you have a full powerVaultItem object available, because at this point the item exists.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

AddItem event is raised when clicking in Vault Client on *New Item...* -> *Selecting Category* -> *Ok*. Therefore you have *only* the data about the selected Category available and seriously not more.

Examples:
AddItem_Restrictions:

```
Register-VaultEvent -EventName AddItem_Restrictions -Action 'RestrictAddItem'

function RestrictAddItem($item) {
        #Write event code here
}
```

AddItem_Pre:

```
Register-VaultEvent -EventName AddItem_Pre -Action 'PreAddItem'

function PreAddItem($item) {
        #Write event code here
}
```

AddItem_Post:

```
Register-VaultEvent -EventName AddItem_Post -Action 'PostAddItem'

function PostAddItem($item, $successful) {
        #Write event code here
}
```

CommitItems

EventNames:

- CommitItems_Restrictions
- CommitItems_Pre
- CommitItems_Post

Parameters

Type	Nam	Description
Item[	items	The items which should get / is comited.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewNumber`, `_NewTitle(Item,CO)`, `_NewDescription(Item,CO)`, `_NewComment` and `_NewUnits`.**Post:**In *POST* you can retrieve the previous information by using the properties `_OldNumber`, `_OldTitle(Item,CO)`, `_OldDescription(Item,CO)`, `_OldComment` and `_OldUnits`.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

CommitItems: is raised when clicking in the item dialog on *Save* or *Save and Close*.

When changing Item number via the Vault Client, the properties `_Number`, `_NewNumber` and `_OldNumber` have the same value.
This happens because the API call **ItemService.CommitItemNumbers** already changes the number on the item that is still in edit-mode.

When committing an Item which has a *Comment*, the properties `_Comment` and `_OldComment` will be empty because the comment will be automatically cleared from the previous API call **ItemService.EditItems**.

Examples:
CommitItems_Restrictions:

```
Register-VaultEvent -EventName CommitItems_Restrictions -Action 'RestrictCommitItems'

function RestrictCommitItems($items) {
        #Write event code here
}
```

CommitItems_Pre:

```
Register-VaultEvent -EventName CommitItems_Pre -Action 'PreCommitItems'

function PreCommitItems($items) {
        #Write event code here
}
```

CommitItems_Post:

```
Register-VaultEvent -EventName CommitItems_Post -Action 'PostCommitItems'

function PostCommitItems($items, $successful) {
        #Write event code here
}
```

DeleteItems

EventNames:

- DeleteItems_Restrictions
- DeleteItems_Pre
- DeleteItems_Post

Parameters

Type	Name	Description
Item[]	items (deletedItems in POST)	The items which should get / are deleted.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
DeleteItems_Restrictions:

```
Register-VaultEvent -EventName DeleteItems_Restrictions -Action 'RestrictDeleteItems'

function RestrictDeleteItems($items) {
        #Write event code here
}
```

DeleteItems_Pre:

```
Register-VaultEvent -EventName DeleteItems_Pre -Action 'PreDeleteItems'

function PreDeleteItems($items) {
        #Write event code here
}
```

DeleteItems_Post:

```
Register-VaultEvent -EventName DeleteItems_Post -Action 'PostDeleteItems'

function PostDeleteItems($deletedItems, $successful) {
        #Write event code here
}
```

EditItems

EventNames:

- EditItems_Restrictions
- EditItems_Pre
- EditItems_Post

Parameters

Type	Name	Description
Item[]	items	The items which should get / are in edit mode.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

The **EditItems event** is raised when:

- clicking on *Open -> Edit* or directly "*Update* "for the item.
- clicking on *Save* (not save and close) in the dialog then first CommitItem is fired, and **then EditItem**
- activating in the the Bill of Materials Tab of the Item dialog a new item

Examples:
EditItems_Restrictions:

```
Register-VaultEvent -EventName EditItems_Restrictions -Action 'RestrictEditItems'

function RestrictEditItems($items) {
        #Write event code here
}
```

EditItems_Pre:

```
Register-VaultEvent -EventName EditItems_Pre -Action 'PreEditItems'

function PreEditItems($items) {
        #Write event code here
}
```

EditItems_Post:

```
Register-VaultEvent -EventName EditItems_Post -Action 'PostEditItems'

function PostEditItems($items, $successful) {
        #Write event code here
}
```

PromoteItems

EventNames:

- PromoteItems_Restrictions
- PromoteItems_Pre
- PromoteItems_Post

Parameters

Type	Name	Description
File[]	files	The files which should get / are assigned to the items.
Item[]	items	The items which should get / are assigned to the files.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

PromoteItems event is raised when clicking "*Assign / Update Item*" on a file or "*Update…*" on a item and afterwards *EditItem* event is fired

- When "*Assign / Update Item*" on a file is clicked wheter the item exists or not, you will *only get data about the files*
- When "*Update…*" on an item is clicked, you will *only get data about the items*
- Changing item category does not fire any event.

Examples:
PromoteItems_Restrictions:

```
Register-VaultEvent -EventName PromoteItems_Restrictions -Action 'RestrictPromoteItems'

function RestrictPromoteItems($files, $items) {
        #Write event code here
}
```

PromoteItems_Pre:

```
Register-VaultEvent -EventName PromoteItems_Pre -Action 'PrePromoteItems'

function PrePromoteItems($files, $items) {
        #Write event code here
}
```

PromoteItems_Post:

```
Register-VaultEvent -EventName PromoteItems_Post -Action 'PostPromoteItems'

function PostPromoteItems($files, $items, $successful) {
        #Write event code here
}
```

RollbackItemState

EventNames:

- RollbackItemState_Restrictions
- RollbackItemState_Pre
- RollbackItemState_Post

Parameters

Type	Nam	Description
Item	item	The item which should get / is reverted for a LifeCycle.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewState` and `_NewLifeCycleDefinition`.**Post:**In *POST* you can retrieve the old information by the properties `_OldState` and `_OldLifeCycleDefinition`.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
RollbackItemState_Restrictions:

```
Register-VaultEvent -EventName RollbackItemState_Restrictions -Action
↪'RestrictRollbackItemState'

function RestrictRollbackItemState($item) {
        #Write event code here
}
```

RollbackItemState_Pre:

```
Register-VaultEvent -EventName RollbackItemState_Pre -Action 'PreRollbackItemState'

function PreRollbackItemState($item) {
        #Write event code here
}
```

RollbackItemState_Post:

```
Register-VaultEvent -EventName RollbackItemState_Post -Action 'PostRollbackItemState'

function PostRollbackItemState($item, $successful) {
        #Write event code here
}
```

UpdateItemStates

EventNames:

- UpdateItemStates_Restrictions
- UpdateItemStates_Pre
- UpdateItemStates_Post

Parameters

Type	Nam	Description
Item[	items	The updated / updating items for a LifeCycle.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewComment`, `_NewLifeCycleDefinition` and `_NewState`.**Post:**In *POST* you can retrieve the previous information by using the properties `_OldComment`, `_OldLifeCycleDefinition` and `_OldState`.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
UpdateItemStates_Restrictions:

```
Register-VaultEvent -EventName UpdateItemStates_Restrictions -Action
↪'RestrictUpdateItemStates'

function RestrictUpdateItemStates($items) {
        #Write event code here
}
```

UpdateItemStates_Pre:

```
Register-VaultEvent -EventName UpdateItemStates_Pre -Action 'PreUpdateItemStates'

function PreUpdateItemStates($items) {
        #Write event code here
}
```

UpdateItemStates_Post:

```
Register-VaultEvent -EventName UpdateItemStates_Post -Action 'PostUpdateItemStates'

function PostUpdateItemStates($items, $successful) {
        #Write event code here
}
```

AddItem

EventNames:

- AddItem_Restrictions
- AddItem_Pre
- AddItem_Post

Parameters

Type	Nam	Description
Item	item	The item which should get / is added.**Restrictions and Pre:** In *Restrictions* and *Pre* you have very limited data, because the item does not exist so far.Only the property `_NewCategoryName` is set as long as a valid category Id was passed.**Post:**In *POST* you have a full powerVaultItem object available, because at this point the item exists.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

AddItem event is raised when clicking in Vault Client on *New Item...* -> *Selecting Category* -> *Ok*. Therefore you have *only* the data about the selected Category available and seriously not more.

Examples:
AddItem_Restrictions:

```
Register-VaultEvent -EventName AddItem_Restrictions -Action 'RestrictAddItem'

function RestrictAddItem($item) {
        #Write event code here
}
```

AddItem_Pre:

```
Register-VaultEvent -EventName AddItem_Pre -Action 'PreAddItem'

function PreAddItem($item) {
        #Write event code here
}
```

AddItem_Post:

```
Register-VaultEvent -EventName AddItem_Post -Action 'PostAddItem'

function PostAddItem($item, $successful) {
        #Write event code here
}
```

CommitItems

EventNames:

- CommitItems_Restrictions
- CommitItems_Pre
- CommitItems_Post

Parameters

Type	Nam	Description
Item[	items	The items which should get / is comited.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewNumber`, `_NewTitle(Item,CO)`, `_NewDescription(Item,CO)`, `_NewComment` and `_NewUnits`.**Post:**In *POST* you can retrieve the previous information by using the properties `_OldNumber`, `_OldTitle(Item,CO)`, `_OldDescription(Item,CO)`, `_OldComment` and `_OldUnits`.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

CommitItems: is raised when clicking in the item dialog on *Save* or *Save and Close*.

When changing Item number via the Vault Client, the properties `_Number`, `_NewNumber` and `_OldNumber` have the same value.
This happens because the API call **ItemService.CommitItemNumbers** already changes the number on the item that is still in edit-mode.

When committing an Item which has a *Comment*, the properties `_Comment` and `_OldComment` will be empty because the comment will be automatically cleared from the previous API call **ItemService.EditItems**.

Examples:
CommitItems_Restrictions:

```
Register-VaultEvent -EventName CommitItems_Restrictions -Action 'RestrictCommitItems'

function RestrictCommitItems($items) {
        #Write event code here
}
```

CommitItems_Pre:

```
Register-VaultEvent -EventName CommitItems_Pre -Action 'PreCommitItems'

function PreCommitItems($items) {
        #Write event code here
}
```

CommitItems_Post:

```
Register-VaultEvent -EventName CommitItems_Post -Action 'PostCommitItems'

function PostCommitItems($items, $successful) {
        #Write event code here
}
```

DeleteItems

EventNames:

- DeleteItems_Restrictions
- DeleteItems_Pre
- DeleteItems_Post

Parameters

Type	Name	Description
Item[]	items (deletedItems in POST)	The items which should get / are deleted.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
DeleteItems_Restrictions:

```
Register-VaultEvent -EventName DeleteItems_Restrictions -Action 'RestrictDeleteItems'

function RestrictDeleteItems($items) {
        #Write event code here
}
```

DeleteItems_Pre:

```
Register-VaultEvent -EventName DeleteItems_Pre -Action 'PreDeleteItems'

function PreDeleteItems($items) {
        #Write event code here
}
```

DeleteItems_Post:

```
Register-VaultEvent -EventName DeleteItems_Post -Action 'PostDeleteItems'

function PostDeleteItems($deletedItems, $successful) {
        #Write event code here
}
```

EditItems

EventNames:

- EditItems_Restrictions
- EditItems_Pre
- EditItems_Post

Parameters

Type	Name	Description
Item[]	items	The items which should get / are in edit mode.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

The **EditItems event** is raised when:

- clicking on *Open -> Edit* or directly "*Update* "for the item.
- clicking on *Save* (not save and close) in the dialog then first CommitItem is fired, and **then EditItem**
- activating in the the Bill of Materials Tab of the Item dialog a new item

Examples:
EditItems_Restrictions:

```
Register-VaultEvent -EventName EditItems_Restrictions -Action 'RestrictEditItems'

function RestrictEditItems($items) {
        #Write event code here
}
```

EditItems_Pre:

```
Register-VaultEvent -EventName EditItems_Pre -Action 'PreEditItems'

function PreEditItems($items) {
        #Write event code here
}
```

EditItems_Post:

```
Register-VaultEvent -EventName EditItems_Post -Action 'PostEditItems'

function PostEditItems($items, $successful) {
        #Write event code here
}
```

PromoteItems

EventNames:

- PromoteItems_Restrictions
- PromoteItems_Pre
- PromoteItems_Post

Parameters

Type	Name	Description
File[]	files	The files which should get / are assigned to the items.
Item[]	items	The items which should get / are assigned to the files.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Unexpected Behaviour

PromoteItems event is raised when clicking "*Assign / Update Item*" on a file or "*Update…*" on a item and afterwards *EditItem* event is fired

- When "*Assign / Update Item*" on a file is clicked wheter the item exists or not, you will *only get data about the files*
- When "*Update…*" on an item is clicked, you will *only get data about the items*
- Changing item category does not fire any event.

Examples:
PromoteItems_Restrictions:

```
Register-VaultEvent -EventName PromoteItems_Restrictions -Action 'RestrictPromoteItems'

function RestrictPromoteItems($files, $items) {
        #Write event code here
}
```

PromoteItems_Pre:

```
Register-VaultEvent -EventName PromoteItems_Pre -Action 'PrePromoteItems'

function PrePromoteItems($files, $items) {
        #Write event code here
}
```

PromoteItems_Post:

```
Register-VaultEvent -EventName PromoteItems_Post -Action 'PostPromoteItems'

function PostPromoteItems($files, $items, $successful) {
        #Write event code here
}
```

UpdateItemStates

EventNames:

- UpdateItemStates_Restrictions
- UpdateItemStates_Pre
- UpdateItemStates_Post

Parameters

Type	Nam	Description
Item[	items	The updated / updating items for a LifeCycle.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewComment`, `_NewLifeCycleDefinition` and `_NewState`.**Post:**In *POST* you can retrieve the previous information by using the properties `_OldComment`, `_OldLifeCycleDefinition` and `_OldState`.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
UpdateItemStates_Restrictions:

```
Register-VaultEvent -EventName UpdateItemStates_Restrictions -Action
↪'RestrictUpdateItemStates'

function RestrictUpdateItemStates($items) {
        #Write event code here
}
```

UpdateItemStates_Pre:

```
Register-VaultEvent -EventName UpdateItemStates_Pre -Action 'PreUpdateItemStates'

function PreUpdateItemStates($items) {
        #Write event code here
}
```

UpdateItemStates_Post:

```
Register-VaultEvent -EventName UpdateItemStates_Post -Action 'PostUpdateItemStates'

function PostUpdateItemStates($items, $successful) {
        #Write event code here
}
```

RollbackItemState

EventNames:

- RollbackItemState_Restrictions
- RollbackItemState_Pre
- RollbackItemState_Post

Parameters

Type	Nam	Description
Item	item	The item which should get / is reverted for a LifeCycle.**Restrictions and Pre:** In *Restrictions* and *Pre* you have the future data set on this object like `_NewState` and `_NewLifeCycleDefinition`.**Post:**In *POST* you can retrieve the old information by the properties `_OldState` and `_OldLifeCycleDefinition`.
bool	successful	**Post**:The information whether the Web Service call was successful or not is only available in *Post* events.

Examples:
RollbackItemState_Restrictions:

```
Register-VaultEvent -EventName RollbackItemState_Restrictions -Action
↪'RestrictRollbackItemState'

function RestrictRollbackItemState($item) {
        #Write event code here
}
```

RollbackItemState_Pre:

```
Register-VaultEvent -EventName RollbackItemState_Pre -Action 'PreRollbackItemState'

function PreRollbackItemState($item) {
        #Write event code here
}
```

RollbackItemState_Post:

```
Register-VaultEvent -EventName RollbackItemState_Post -Action 'PostRollbackItemState'

function PostRollbackItemState($item, $successful) {
        #Write event code here
}
```

They can be used as parameters in *registered* functions or script blocks.

Tip: You are not forced to use all the defined arguments, since they are all **optional**.
Therefore you can pass only the *parameters you really need* into your script blocks.

Warning: The arguments are case sensitive and must be **named exactly** the same as described in the linked documentation.

5.2.3 Host

The automatic variable $Host is of type PSHost and represents the current host application for PowerShell.
Within the context of powerEvents this can either be the Vault *WebServiceExtension* running an extended Host version or your *PowerShell IDE*.

Syntax

```
$Host.Name
```

See also:

For the complete list of properties and methods and for more information see: PSHost Class.

Following properties are always available :

Type	Nam	Description	Access type
Strin	Name	An identifier for the PowerShell hosting application. Within *Client Customizations* this should be *'powerEvents Webservice Extension'*.	read-only
PSOb ject	Private-Data	Each Host can provide private data that is editable. For example the property ErrorBackgroundColor that is available in several PowerShell IDEs for changing the background color of error messages.The powerEvents Host allows manipulating the way how Vault users are notified about Terminating Errors using the `OnTerminatingError` property.	read-write

Remarks

In contrast to your PowerShell IDE the powerEvents Host redirects the written output into the *logfile*.
The format of the output provided by the Write-Host cmdlet can be customised within the according *Logging* sections.

Error Notifications

By default the powerEvents host is configured to write to the logfile and display a modal Vault restrictions dialog to notify users *about erroneous* scripts, modules and exceptions that were thrown by registered Vault events.
The way how users are notified about terminating errors can be changed using the **PrivateData** and its `OnTerminatingError` property that provides all the relevant error details:

```
$global:Host.PrivateData.OnTerminatingError = [Action[System.Management.Automation.
↪RuntimeException]] {
        param($exception)
                $errorType = $exception.GetType().FullName
                $errorMessage = $exception.Message
                $errorStackTrace = $exception.ErrorRecord.ScriptStackTrace
```

(continues on next page)

(continued from previous page)

```
                if($exception.Data['RegisteredEvent'])
                {
                        $script = $exception.Data['RegisteredEvent'].Script.FullName
                        $vaultEventName = $exception.Data['RegisteredEvent'].Name
                        Write-Host "An unexpected error has occurred in the script '
↪$script' for Vault event '$vaultEventName'.`r`n$errorType : $errorMessage`r`n
↪$errorStackTrace"
                }
                else
                {
                        Write-Host "An unexpected error has occurred in the script or␣
↪module '$($exception.ErrorRecord.InvocationInfo.ScriptName)'`r`n$errorType :
↪$errorMessage`r`n$errorStackTrace"
                }
}
```

Automatic Script reloading

Changes made in the PowerShell files located in the *Events* or *Modules* folder are loaded by the powerEvents Host at *every start*.
Also when the files are modified while they are loaded, powerEvents **automatically executes them again**. Restarting your application is not required.
Adding or removing *Scripts* or *Modules* also causes powerEvents to automatically reload the whole configuration.

Please be aware that long running operations or modal dialog boxes can block this script execution (e.g. *error dialogs*). Changes that are saved in the meantime may then not take effect for the current script execution, and the original script execution will continue after such dialogs are closed. However, the changes will take effect immediately on the next run.
Cmdlets such as Show-Inspector or Show-BOMWindow are an exception. They block the current script execution with the windows they show, but they still allow nested reloading of scripts (but again, PowerShell operations are only executed sequentially).

Note: Similarly, also simple Form.ShowDialog or System.Windows.Forms.MessageBox.Show usages, as well as Window.ShowDialog or System.Windows.MessageBox.Show calls may not block the main window thread from processing automatic reloads.
If these functions are used - by mistake without owner window assignment - changes at runtime might not be supported and can lead to misbehavior.
So, as long as these dialogs are open, no automatic reload should be triggered, or a restart of the application may be necessary for all changes to take effect correctly.

We're still working on the feature and the documentation to detail what edits are supported.

Keep this functionality in mind when developing new client customizations or when saving changes in your PowerShell IDE.
In order to disable the automatic realoding mechanism you can use the global flag `$powerEvents_ReloadPsScripts` in the *Common.psm1 Module*:

```
$global:powerEvents_ReloadPsScripts = $false
```

After enabling the functionality again, you need to restart the application.

5.2.4 Inventor

The automatic variable $Inventor is of type *Application Object* and gives direct access to the **Inventor's API**.

It is the topmost object in the API hierarchy and supports methods and properties that let you control the Inventor Application. Most importantly, it provides access to the properties and information of the active document.

Syntax

```
$Inventor.ActiveDocument
```

See also:

For the complete list of properties and methods and for more information see: Inventor API Reference Manual.

CHAPTER

SIX

LOGGING

powerEvents uses Apache log4net as core logging library, and additionally PostSharp Diagnostics for extended Debug logging.
By default, all the logs are stored in a logfile located in *'C:\Users\{USER}\AppData\Local\coolOrange\powerEvents\Logs\powerEvents.log* and it contains only Infos, Warnings and Errors.
The log4net settings file is located in *C:\Program Files\coolOrange\Modules\powerEvents\powerEvents.log4net*.
Further information about log4Net Configurations can be found here.

6.1 When to change the logging behavior?

When you have issues with failing script executions or when you want to get a more detailed knowledge about what powerEvents or powerVault cmdlets are doing, you can increase the loglevel.

Note: When changing the loglevel to *DEBUG* PostSharp Diagnostics will be enabled and will log all the function calls into the log files.
This could cause performance issues.

6.2 LogFile

You can see, that there are multiple logging-Appenders used. If you want to change the outputpath or the name of the logfile, please visit following appender:

```
<appender name="FileAppender"
          type="log4net.Appender.RollingFileAppender">
        <param name="File" value="${LOCALAPPDATA}\coolOrange\powerEvents\Logs\
↪powerEvents.log" />
```

The logging level for the logfile can be changed in the following lines:

```
<root>
        <level value="INFO" />
        <appender-ref ref="FileAppender"/>
        <appender-ref ref="ColoredConsoleAppender" />
</root>
```

By changing the level to "DEBUG", log entries wih all the levels between the range Debug and Fatal will be written to the logfile.

This way you can get more details about errors that occur during:

- the execution of client customization *scripts* (within the powerEvents *Host*)
- during the execution of *Cmdlets* or their *-Action* parameters (also if in external PSHosts)

To control the logging behaviour of *powerVault Cmdlets*, the following section is used:

```
<logger name="powerVault.Cmdlets">
        <level value="INFO" />
</logger>
```

Note: When powerVault cmdlet's are getting used *before importing the powerEvents module*, not all the powerVault logs can be redirected to the logger configured in the previously mentioned *section*.

6.3 PowerShell IDE

PowerShell IDEs like PowerShell console (and PowerShell ISE) are configured to show the logging levels in a different color.
In order to customize the logging level in the console window, visit following appender:

6.3.1 ColoredConsoleAppender

ColoredConsoleAppenders are working for PowerShell IDE's that support console windows.

```
<appender name="ColoredConsoleAppender" type="log4net.Appender.
↪ManagedColoredConsoleAppender">
```

In the lines

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

you can configure the required logging level. You could set the minimal filter level to "DEBUG", than all the levels between the range Debug and Fatal will be logged.
This appender allows changing even the colors of the messages, depending on their log level:

```
<mapping>
        <level value="DEBUG" />
        <foreColor value="Black" />
        <backColor value="White" />
</mapping>
<mapping>
        <level value="INFO" />
        <backColor value="DarkGreen" />
</mapping>
<mapping>
        <level value="WARN" />
        <backColor value="DarkYellow" />
```

(continues on next page)

(continued from previous page)

```
</mapping>
<mapping>
        <level value="ERROR" />
        <backColor value="Red" />
</mapping>
<mapping>
        <level value="FATAL" />
        <backColor value="DarkRed" />
</mapping>
```

CHAPTER

SEVEN

CHANGE LOGS

7.1 powerEvents v24

7.1.1 v24.0.13

`05-06-2024`

Fixed

- Improved *Add-VaultMenuItem -Location ToolsMenu* parameter, which failed to add menu items to the Vault "Tools" menu after incomplete updates to Vault Client 2024.3.
  This fixes *MissingMethodExceptions* caused by older DevExpress 21.1.5 versions that remain installed in the GAC, e.g. if the recommended CAD application updates were forgotten or the necessary Vault Server 2024.3 update was overlooked on test-environments with additional ADMS installation.

7.1.2 v24.0.12

`05-03-2024`

Fixed

- Occasional issue with registered *LoginVault_Post* events where the Vault Client 2021 freezes on login.
- Issue where loading powerEvents fails when logging into Vault via an application that lacks the required Vault SDK references (e.g. *Autodesk.Connectivity.Explorer.ExtensibilityTools.dll*).
- Issue where *Client Customizations* scripts are not executed when logging into Vault via an application without a user interface (e.g. *.NET Console Application*).

7.1.3 v24.0.11

`23-02-2024`

General

- Updated powerVault to version: 24.0.8

Fixed

- Issue where Vault Client and Inventor freeze when registered *LoginVault_Post* event *-Action* makes use of *Connection.FileManager.AcquireFiles* calls.
  Since this VDF function is also used by Save-VaultFile, Add-VaultFile and Update-VaultFile, these cmdlets can also cause the deadlock when they perform a download or a check out.

7.1.4 v24.0.9

24-01-2024

Fixed

- Issue that *LoginVault_Post* event *-Action* was executed too often,
  sometimes e.g. twice after logins in Inventor (rarely observed in Vault Client), or again after *automatically reloaded script changes*

7.1.5 v24.0.8

21-12-2023

Fixed

- Compatibility Issue with powerJobs Client that prevented the use of powerVault Cmdlets (and VDF *Property-Manager* functions) in *LoginVault_Post* actions.
  Unfortunately, when used, also other cmdlet *-Action* parameters experienced *NullReferenceExceptions*, which completely prevented their execution or caused them to terminate unexpectedly.

7.1.6 v24.0.7

16-10-2023

General

- Updated powerVault to version: 24.0.7

7.1.7 v24.0.6

30-08-2023

General

- Updated powerVault to version: 24.0.5

7.1.8 v24.0.5

21-08-2023

Fixed

- Issue where Vault Client and Inventor freeze when client customizations register *CheckoutFile* events that are triggered by custom *Connection.FileManager.AcquireFiles* calls.
  Since this VDF function is also used by Add-VaultFile and Update-VaultFile, these cmdlets can also cause the deadlock when they perform a check out (on main UI thread).

7.1.9 v24.0.4

11-08-2023

Features

- Folder structure for *Client Customizations* has been simplified
- The PowerShell runspace, that loads these client customizations, got a friendly name: "coolOrange".
  This makes it easier to *debug script* executions by attaching a Powershell debugger to the Vault application.
- Client Customizations can now be easily *distributed* from any environment, whether it's a development machine or a productive workstation

General

- Removed the non-intuitive "Events" directory, as it also allows PowerShell scripts to extend the user interface of Vault Client, Inventor or realizing entire ERP integrations.
  Therefore, *Scripts* are now placed directly in the directory *"C:\ProgramData\coolOrange\Client Customizations"* and *Modules* in the subdirectory of the same name.
- Minor adjustments in text references to "event scripts" in messages for logs and Error Message Boxes
- Windows Permissions for directory *"C:\ProgramData\coolOrange"* were changed to *Read*, *Write* & *Delete* for *Everyone* to allow synchronizing scripts and modules
- The *Publish-Customizations.ps1* script has been moved to *"C:\ProgramData\coolOrange"* but still only supports the *distribution* of client customizations
- During Vault application startup, if *distribution* problems occur with client customizations, the Error Message Box now displays additional details that help in resolving these issues more effectively

Breaking Changes

Change paths in Customization distribution mechanisms

If you or your IT department use a distribution mechanism to automatically install or update scripts and modules on all workstations, note that the paths where they are stored must be changed!

Previous	Now
`C:\ProgramData\coolOrange\powerEvents\Events`	`C:\ProgramData\coolOrange\Client Customizations`
`C:\ProgramData\coolOrange\powerEvents\Modules`	`C:\ProgramData\coolOrange\Client Customizations\Modules`

When *updating* to this version (or newer), all scripts and modules from the old directory *"C:\ProgramData\coolOrange\powerEvents"* are automatically moved to the correct structure.
In addition, powerEvents reminds and assists in loading customizations also from this old directory (for another two major versions).
However, please be aware that related functionalities - such as the *automatic reloading* of scripts and the build-in *distribution mechanism* - unfortunately no longer work for the obsolete directory.

7.1.10 v24.0.3

30-06-2023

General

- Updated Licensing to version: 18.3.1
  This significantly improves the performance of loading powerEvents during Vault logins when a Standalone license is registered for the product.

7.1.11 v24.0.2

09-06-2023

Features

- Script changes can now be saved while modal dialogs are open, resulting in a smoother debugging and development experience.
  For example, if changes are saved while an *Error Message Box* or other blocking windows are displayed, the dialog box can be closed and the current changes will automatically take effect at the next execution.
- For active Vault Client tabs created with *Add-VaultTab*, the *-Action* parameter is now automatically re-executed when changes are *automatically reloaded*.

Fixed

- Issue where the *automatic reloading* of script changes stopped working if a reload did not complete. Reasons for this could be concurrent PowerShell executions or blocked runspaces.
  Restarting the host application is now no longer required because it ensures that all recent changes are reloaded, even if modal dialogs are open when saving.
- Changes to customization scripts are no longer *reloaded* on MTA background threads, which prevents unexpected *"System.InvalidOperationException: The calling thread must be STA ..."* errors when running scripts that use WPF elements.
- Issue where *Error Message Boxes* appeared in the background of the host application

7.1.12 v24.0.1

21-04-2023

General

- Added support for Vault 2024
- Updated Licensing to version: 18.2.29
- End User License Agreement (EULA) has changed

7.2 powerEvents v23

7.2.1 v23.0.20

`17-04-2023`

General

- Updated powerVault to version: 23.0.15.

Fixed

- Issue when using `finally` statements in script blocks or PowerShell functions, where the execution of *-Action* parameters in *Register-VaultEvent*, *Add-VaultTab*, *Add-VaultMenuItem* and *Add-InventorMenuItem* cmdlets failed with *System.NullReferenceException* even before the actual finally block could execute
- Display issue in the Error Message Box where *System.Management.Automation.RuntimeException* was displayed for all *terminating errors* instead of their actual exception type

7.2.2 v23.0.19

`06-03-2023`

Features

- **New cmdlet**: *Add-InventorMenuItem* to register menu items in Inventor ribbons with custom Actions that are invoked when the menu items are clicked.

Fixed

- Issue where Vault Data Standard (VDS) customizations executed significantly slower, **after** registered *VaultEvents* were triggered from the VDS customization

7.2.3 v23.0.17

`22-02-2023`

General

- Updated powerVault to version: 23.0.14.

Fixed

- Issue in the *Add-VaultTab* cmdlet where the Vault Client crashed when terminating exceptions occurred in UI events, such as ScriptBlocks invoked on `$button.add_Click({ ... })` events

7.2.4 v23.0.16

`14-02-2023`

General

- Updated powerVault to version: 23.0.11.

Features

- *Add-VaultMenuItem*: Added support to add menu items to the context menu of Vault **Folders**

Fixed

- Issue where context menu items were not displayed when the *Add-VaultMenuItem* cmdlet was used for multiple different *locations*
- Issue in *UpdateFileStates_Restrictions* and *_Pre* events where properties of `$files` parameter instances return either empty or wrong values when accessed from outside the registered function or Scriptblock.
- Issue where uninstalling product also uninstalled powerVault, although it was still needed by other products, e.g. powerJobs Processor.

7.2.5 v23.0.14

`01-02-2023`

General

- Significant performance improvements when large Inventor assemblies are Checked-in to Vault and *AddFile*- or *CheckinFile* events are registered.
  For these events, unncecessary Vault API calls will be prevented when `$dependencies`,`$attachments` or `$fileBom` parameters are not used.
- Product-wide performance improvements by configuring the log level 'INFO' for the *<root> logger* to avoid automatically generated DEBUG logs if not configured.
- Updated powerVault to version: 23.0.10.

7.2.6 v23.0.13

`12-01-2023`

Fixed

- Issue where new "low-level Vault connection" was created against Vault Data Server (ADMS), when working in environments with dedicated/separated Vault File Server (AVFS).

7.2.7 v23.0.12

`15-12-2022`

General

- Behavior of `Add-VaultMenuItem` when called multiple times for same menu item has changed: Instead of ignoring subsequent calls for same menu item, *Add-VaultMenuItem* cmdlet now updates the passed `-Action` for the menu item.

Features

- *Add-VaultMenuItem*: Added support to add menu items to the Vault Explorer's **Tools menu**

7.2.8 v23.0.11

07-12-2022

Features

- **New cmdlet**: *Add-VaultTab* to register a new tab for a specific entity type in the Vault Explorer with an Action that is invoked when the tab is clicked.
- Simplified *enabling and disabling* customization scripts by providing a *"./Disabled"* subdirectory where scripts can be moved to, instead of uncommenting or commenting the contained `Register-VaultEvent ...` lines.

Fixed

- Issue that manually disabled/enabled Sample- and standard *scripts* are automatically reverted after installing *Updates*.
  Note: This will only affect future versions. If sample scripts have been manually enabled (uncommented `Register-VaultEvent ...` lines), they can be moved to *%ProgramData%/coolOrange/powerEvents/Events* directory after updating from *v23.0.9* or earlier.
- Issue that *LoginVault_Post* event actions were not executed before those of other event registrations
- Issue where updating a property mapped to an AutoCAD property with Update-VaultFile in an event causes the Vault Client to freeze.

General

- Changed type of property *Name* on *Event* results from *'powerEvents.Cmdlets.VaultEvent'* to *'String'*.

7.2.9 v23.0.9

27-10-2022

General

- Updated powerVault to version: 23.0.8.

Features

- **New VaultEvent**: *LoginVault_Post* which gets raised after successfully signing in to Vault.

Fixed

- Issue with *AddFile*- and *CheckinFile* events not getting executed when files with empty BOMs (and *copyBom* parameter disabled) are checked into Vault.
  This applies to all non-modelling files (e.g. drawings, presentation files or automatically generated design visualization files) that are checked in through Inventor or AutoCAD.

7.2.10 v23.0.7

29-09-2022

General

- Updated powerVault to version: 23.0.7.
- The setup has been extended to provide a dependency key that is required for other product setups that depend on powerEvents

Fixed

- Issue where repair via setup was not possible if powerVault was previously uninstalled

- Issue where powerVault was listed twice in Programs and Features after upgrading powerVault

7.2.11 v23.0.6

`23-08-2022`

Features

- The `-Action` parameter of the *Add-VaultMenuItem* cmdlet is invoked with powerVault entities as parameter, which provide all entity properties without having to implement custom logic to retrieve them

7.2.12 v23.0.5

`28-07-2022`

Features

- **New cmdlet**: *Add-VaultMenuItem* to register menu items with custom actions in the Vault Explorer
- **New client customization**: *SubmitJobsOnVaultMenuItemClick.ps1* to submit jobs from the context-menu (right-click menu) in Vault when configured in the powerJobs Settings Dialog.

General

- Updated powerVault to version: 23.0.6.
  This allows custom applications to execute events registered in customization *scripts* again when using powerVault v23.0.4 or later (see fix *v23.0.1*).
  Also, warning logs like *"Creating a low level connection to the server"* should no longer appear.

7.2.13 v23.0.1

`20-04-2022`

Features

- Added support for Vault 2023
- Added support for *console logs* in **PowerShell ISE**

General

- Updated powerVault to version: 23.0.1.
  Memory usage is reduced when a large number or frequently executed Vault events are *registered* (see 22.0.9).

Fixed

- Vulnerability in *Logging* configuration files by updating *log4net* to v2.0.14 (CVE-2018-1285)
- Issue with *ColoredConsoleAppender* that caused powershell remote hosts to crash when appender was logging to console
- Issue where the *Action* from the *Register-VaultEvent* was not executed when Vault event was triggered from custom application
- "Page not found" error page opens after clicking Help button in Control Panel → Add or Remove Programs

7.3 powerEvents v22

7.3.1 v22.0.5

02-03-2022

Features

- New client customization ***SubmitJobsOnLifecycleTransition.ps1*** to submit jobs on lifecycle transitions configured in the powerJobs Settings Dialog.
  - Configuration for Vault UDP to submit the *Sample.CreateDXF&STEPfromSheetmetal* job only for Sheet Metal parts is no longer required and is now determined by the script.
- *Technical Preview*: Client customizations (scripts and modules() can be easily *distributed* and will be automatically downloaded to all workstations upon application startup

General

- Updated Licensing to version: 18.2.27
- Removed *Sample.SubmitPublishingJobsOnRelease.ps1* sample as the functionality is replaced by *SubmitJobsOnLifecycleTransition.ps1*
- Directory of *Client Customizations*: Windows Permissions changed to **Read**, **Write** & **Delete** for **Everyone** to allow synchronizing scripts and modules
- Removed support for Custom Objects in the *Sample.ValidateProperties.ps1* script as it was not working properly.

Fixed

- Fixed and issue in the *Sample.ValidateProperties.ps1* script which caused it to mistakenly show a restriction when the vault displayname differed from the account name for the same user.

7.3.2 v22.0.4

08-10-2021

Features

- New client customization script *Sample.SubmitPublishingJobsOnRelease* replaces the script *Sample.TriggerDwfJobOnRelease* and handles the publishing of PDF files on the release of Inventor and AutoCAD drawings, and of DXF and STEP files after releasing sheet metal parts.
  Simple configurable settings allow further customization.
- New client customization script *Sample.RestrictDisturbingSubmittedJobs.ps1* to restrict changing the Lifecycle state of files as long as not all their jobs have been processed successfully

General

- Updated Licensing to version: 18.2.26

7.3.3 v22.0.1

28-04-2021

Features

- Added support for Vault 2022

General

- Updated Licensing to version: 18.1.24
- End User License Agreement (EULA) has changed

Breaking Changes

Linked properties of Items passed in the ChangeOrder events have a prefix

Linked properties of ChangeOrder Items have to be retrieved by using the **'Record_'** prefix within the *AddChangeOrder* and *CommitChangeOrder* event actions.

7.4 powerEvents v21

7.4.1 v21.0.7

04-02-2021

Features

- *Logfile* contains more details about erroneous scripts and provides the filename and line number of PowerShell errors together with the name of the failed Vault event

General

- Improved informations displayed in the *Error notification* for Vault users, even because his preconfigured processes may not have been fully executed in error situations
- Updated Licensing to version: 18.1.22

Fixed

- Issues with missing filenames and wrong line numbers in StackTraces and confusing "at <ScriptBlock>" entries beeing provided in *Error notifications* when the execution of registered Vault events fails

7.4.2 v21.0.6

21-12-2020

General

- Updated Licensing to version: 18.1.21

Fixed

- Issue that errors were *logged* multiple times
- Issue that led to an unusable machine and failing Jobs after running JobProcessor for a long time

7.4.3 v21.0.4

06-11-2020

Features

- *Error notification* informs about erroneous scripts and when the registered Vault events are raised, even about failed script block or PowerShell functions

General

- Updated Licensing to version: 18.1.19
- Copyright notices have changed

7.4.4 v21.0.3

08-06-2020

Fixed

- Compatibility-Issue with other coolOrange products using an older *Logging* version

7.4.5 v21.0.1

27-04-2020

Features

- Added support for Vault 2021

General

- End User License Agreement (EULA) has changed
- Updated Licensing to version: 18.1.17
- Added *powerEvents Information* shortcut to startmenu
- Removed *powerEvents Help* shortcut from startmenu as it can be accessed via powerEvents Information shortcut
- Removed Splashscreen

7.5 powerEvents v20

7.5.1 v20.0.7

05-11-2019

General

- Updated Licensing to version: 18.0.10

7.5.2 v20.0.6

25-10-2019

General

- Updated Licensing to version: 18.0.9

7.5.3 v20.0.5

20-08-2019

General

- Updated Licensing to version: 18.0.8

7.5.4 v20.0.4

29-05-2019

- Official Release

Features

- Added support for *Token Licensing*
- Added support for *Stand-Alone Licensing*

General

- End User License Agreement (EULA) has changed
- Updated Licensing to version: 18.0.7
- Assemblies *coolorange.licensing* and *coolorange.Utils.UI.v18.0* now gets installed in the GAC
- *Trial mode* expires after 30 days
- Splashscreen is shown when connecting to Vault

Fixed

- Issue when using *Register-VaultEvent* and *Unregister-VaultEvent* in different runspaces (e.g. in PowerShell ISE) did not work properly

7.5.5 v20.0.1-beta

14-01-2019

Features

- Added support for Vault, Inventor and AutoCAD 2020 (BETA)

7.6 powerEvents v19

7.6.1 v19.0.8

24-04-2018

- Official Release

Features

- powerEvents *cmdlets* can be used in every IDE

General

- Updated to PowerShell 4.0
- replaced Log4PostSharp with PostSharp Diagnostics for extended Debug logging
- Re-enabled extended Debug *Logging* for Vault 2019
- Added *powerEvents ISE* shortcut in startmenu
- Assembly *coolOrange.VaultServices_[Vault Version]* gets installed into the GAC

Fixed

- Issue where logging did not work when powerEvents was used in a 32-Bit process

7.6.2 v19.0.1beta

24-01-2018

Features

- Added support for Vault, Inventor and AutoCAD 2019 (BETA)

Note: Extended Debug *Logging* is disabled for Vault 2019

7.7 powerEvents v18

Vault 2016 and earlier

CommitItems event is raised when changing the LifeCycleState of items **CommitItems** event is fired after the UpdateItemLifecycleState event

---

7.7.1 v18.0.15

15-12-2017

General

- Updated powerVault to version 18.0.19

Fixed

- Vault events are not getting executed because of compatibility issue with powerVault 18.0.19 and later

7.7.2 v18.0.14

13-10-2017

Fixed

- Issue that "low-level Vault connection" was created against Autodesk Data Management Server (AMDS), when working with SSL connections

7.7.3 v18.0.13

25-08-2017

General

- Updated powerVault to version: 18.0.17

Fixed

- Disabled Sample.ValidateProperties.ps1 by default, by commenting *Register-VaultEvent* calls in script

7.7.4 v18.0.11

18-08-2017

General

- Removed property *SourceObject* from *Event*

Features

- New cmdlet: *Unregister-VaultEvent*
- Scripts and Modules become *automatically reloaded* at runtime, when developping new scripts or modifying them
- *CustomObject*- Events are now supported *read more*
- *ChangeOrder* - Events are now supported *read more*

Fixed

- Issue where PowerShell Host depending Cmdlets like Out-File or Out-Host were throwing an exception when used in Vault *Event Actions*
- Issue that "low-level Vault connection" was created against Autodesk Data Management Server (ADMS), when working in environments where ADMS and Autodesk Vault Filestore Server (AVFS) are located on different environments

7.7.5 v18.0.7

17-07-2017

- Official Release

General

- All *File- Item-* and *Folder-* Events are supported *read more*
- New cmdlet: *Register-VaultEvent*
- New cmdlet: *Add-VaultRestriction*
- Added *standard logging* with installation into GAC
- Added client customization *samples*

Fixed

- poweEvents did not load in AutoCAD and Inventor, now it works in both applications
- Performance issues where too much memory was consumed, because multiple PowerShell sessions were created

powerEvents is a Vault extension, which allows to take advantage of the Vault Client events, such as check-in/-out, lifecycle change and the like.
It also provides functionality to programmatically extend the UI of the Vault Client with Context Menu Items or Tabs and the Inventor with Menu Items in the Inventor Ribbons.

You can write custom code that will be executed on the according event via PowerShell. Let's say you would like to fill or clear some properties on a lifecycle state change,
automatically rename files when imported via drag&drop or prevent a lifecycle state transition when a combination of properties does not match your rules, etc.
The product makes it easy to create such logic in a simple PowerShell script and, if desired, distribute this customization directly to other Vault environments.